GU0001

UNIX at Fermilab

Computing Division
Fermi National Accelerator Laboratory

September 30, 1996

ABSTRACT

UNIX at Fermilab provides an overview of UNIX as implemented in the Fermi UNIX Environment (FUE). It is intended primarily for new UNIX users and for users who are familiar with UNIX but new to the Fermilab implementations.

Revision Record

August 1990	Original Draft
June 1991	Revised
September 1995	Revised
October 1995	Revised
November 6, 1995	Revised
November 20, 1995	Revised
January 3, 1996	Revised
February 20, 1996	Revised
March 15, 1996	Revised
August 7, 1996	Revised
September 30, 1996	Revised (Significant changes in sections 1.5, 4.4, 5.1, 5.3.3, and 7.1)

This document and associated documents and programs, and the material and data contained therein, were developed under the sponsorship of an agency of the United States government, under D.O.E. Contract Number EY-76-C-02-3000 or revision thereof. Neither the United States Government nor the Universities Research Association, Inc. nor Fermilab, nor any of their employees, nor their respective contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights. Mention of any specific commercial product, process, or service by trade name, trademark, manufacturer, supplier, or otherwise, shall not, nor is it intended to, imply fitness for any particular use, or constitute or imply endorsement, recommendation, approval or disapproval by the United States Government or URA or Fermilab. A royalty-free, non-exclusive right to use and disseminate same for any purpose whatsoever is expressly reserved to the U.S. and the U.R.A. Any further distribution of this software or documentation, parts thereof, or other software or documentation based substantially on this software or parts thereof will acknowledge its source as Fermilab, and include verbatim the entire contents of this Disclaimer, including this sentence.

Acknowledgments

The Fermilab Computing Division's VMS Migration Task Force has been responsible for updating *UNIX at Fermilab* in time for the mass migration from FNALV in 1996. The task force includes: Pushpa Bhat, Joel Butler, Lauri Loebel Carpenter, Mike Diesburg, Dave Fagan, Ken Fidler, Steve Hanson, Rob Harris, Joy Hathaway, Anne Heavey, Art Kreymer, Paul Lebrun, Frank Nagy, Judith Nicholls, Dane Skow, Mike Stolz, Roy Thatcher, Al Thomas, and Steve Wolbers.

UNIX at Fermilab in its present form may be described as loosely based on its previous release in June 1991, although so much new and updated information has been included that the resemblance is now minimal. The *CERN UNIX User Guide*, after having lifted information from the early release of *UNIX at Fermilab*, was in turn used to provide much of that information back to us appropriately updated! Naturally, we lifted additional useful information from CERN's well-constructed manual. The CERN document *Shell Choice, A shell comparison* was also used.

Other contributors/reviewers include Liz Buckley-Geer, Chuck DeBaun, Lynn Garren, Randy Herber, Peter Kasper, Marc Mengel, Clyde Moseberry, Gene Oleynik, Ruth Pordes, Marilyn Schweitzer, Jonathan Streets, and Margaret Votava.

We'd like to thank the readers who have taken the time to send us suggestions about the manual.

Table of Contents

Chapter 1: Introduction 1-1
1.1 The *UNIX at Fermilab* Manual 1-1
1.1.1 Conventions Used in this Manual 1-2
1.1.2 Summary of Chapters and Appendices 1-3
1.2 About UNIX 1-6
1.2.1 A Brief History 1-6
1.2.2 A Word About Features and Components 1-7
1.2.3 The Advantages of UNIX 1-7
1.3 The Fermi UNIX Environment 1-9
1.4 UNIX Support at Fermilab 1-9
1.5 UNIX References 1-10
1.5.1 Fermilab Documents 1-10
1.5.2 Commercially Available Texts 1-10
Chapter 2: Getting Started on a UNIX System 2-1
2.1 Logging In 2-1
2.1.1 C Shell Family 2-2
2.1.2 Bourne Shell Family 2-2
2.2 Logging Out 2-2
2.2.1 C Shell Family 2-2
2.2.2 Bourne Shell Family 2-2
2.3 The UNIX Prompt 2-3
2.4 Special Keys 2-3
2.5 Special Characters (Metacharacters) 2-4
2.6 File Systems: Standard UNIX and AFS 2-5
2.7 Information Distribution System: NIS 2-6
2.8 Changing Your Password 2-6
2.8.1 Standard UNIX Password 2-6
2.8.2 AFS (Kerberos) Password 2-7
Chapter 3: Information Resources 3-1
3.1 UNIX On-Line Help 3-1
3.1.1 man Pages 3-1
3.1.2 Finding the Right Command 3-3
3.1.3 Vendor Product Documentation 3-4
3.2 The Internet 3-4
3.2.1 World Wide Web (WWW) 3-4

3.2.2 UNIX Help on WWW 3-6
3.2.3 Newsgroups 3-6
3.3 The Info Utility 3-7
3.4 Other Users: WWW Directories, finger and who 3-7
Chapter 4: Shells 4-1
4.1 Introduction to Shells 4-1
4.1.1 Determining Your Current Shell 4-1
4.1.2 Starting a Shell 4-2
4.1.3 Exiting a Shell 4-2
4.2 Features of Available Shells 4-3
4.3 Supported/Recommended Shells at Fermilab 4-4
4.4 Shell Scripts 4-4
4.5 Other Interpretive Programming Languages 4-6
Chapter 5: UNIX Commands and Concepts 5-1
5.1 Processing Environment 5-1
5.1.1 Programs, Commands and Processes 5-1
5.1.2 Command Interpretation by the Shell 5-2
5.2 Using Commands 5-3
5.2.1 Command Entry 5-3
5.2.2 Command Recall 5-4
5.3 Important Concepts 5-6
5.3.1 Path 5-7
5.3.2 Standard Input and Output Redirection 5-7
5.3.3 Pipes 5-9
5.3.4 Filters 5-10
5.3.5 Regular Expressions 5-13
5.4 Job Control 5-14
5.4.1 Priority 5-15
5.4.2 Background, Foreground, and Suspended Jobs 5-15
5.4.3 Scheduling Jobs: at and cron 5-17
Chapter 6: The UNIX File System 6-1
6.1 Directory Structure 6-1
6.1.1 Pathnames 6-1
6.1.2 The Home Directory 6-2
6.1.3 Command Line Directory Shortcuts 6-3
6.1.4 Directories and Executables 6-3
6.2 Files 6-4
6.2.1 Filenames 6-4
6.2.2 Filename Expansion and Wildcard Characters 6-4
6.3 Manipulating Files 6-5
6.3.1 List Directory Contents: ls 6-6
6.3.2 List File Contents: cat, less, more, head, and tail 6-7
6.3.3 Copy a File: cp 6-8
6.3.4 Move (Rename) a File: mv 6-9
6.3.5 Reference a file: ln 6-9

6.3.6 Remove a File: rm 6-10
6.3.7 Compress or Expand a File: gzip, gunzip 6-10
6.4 Information About Files 6-11
6.4.1 Find a File: find 6-11
6.4.2 Search for a Pattern: grep 6-12
6.4.3 Count a File: wc 6-13
6.4.4 Dump a File: od 6-13
6.4.5 Determine File Type: file 6-14
6.5 Manipulating Directories 6-14
6.5.1 Print Working Directory: pwd 6-14
6.5.2 List Directory Contents: ls 6-14
6.5.3 Change Directory: cd 6-15
6.5.4 Make a Directory: mkdir 6-15
6.5.5 Copy a Directory 6-15
6.5.6 Move (Rename) a Directory: mv or mvdir 6-16
6.5.7 Remove a Directory: rmdir 6-16
6.6 File and Directory Permissions 6-16
6.6.1 File Access Permissions 6-16
6.6.2 AFS File Permissions 6-18
6.6.3 Directory Permissions 6-19
6.7 Temporary Directories 6-19

Chapter 7: Printing 7-1

7.1 The FUE Print Command: flpr 7-1
7.2 Pre-Printing Options 7-3
7.2.1 a2ps 7-3
7.2.2 psnup 7-3
7.2.3 Setting Duplex Mode 7-4
7.3 Other Print Utilities 7-5

Chapter 8: Working Environment 8-1

8.1 Shell Variables and Environment Variables 8-1
8.1.1 C Shell Family 8-1
8.1.2 Bourne Shell Family 8-2
8.2 Some Important Variables 8-3
8.3 The Alias Command 8-4
8.3.1 C Shell Family 8-4
8.3.2 Bourne Shell Family 8-5
8.4 Tailoring Your Environment 8-5
8.4.1 C Shell Family Fermi Files 8-5
8.4.2 Bourne Shell Family Fermi Files 8-7
8.4.3 Storing Customized Code 8-8
8.5 Multimedia Support 8-8
8.5.1 The .mailcap File 8-8
8.5.2 The .mime.types File 8-9
8.6 Terminal Characteristics 8-9

Chapter 9: UPS Products and Applications 9-1
9.1 Product Support Overview 9-1
9.2 The UNIX Software Database 9-2
9.3 Accessing and Using Products 9-2
9.4 Obtaining Products 9-3
9.4.1 upd 9-4
9.4.2 Anonymous ftp 9-5
Chapter 10: Editors 10-1
10.1 The Available Editors 10-1
10.2 Comparison of Editors 10-1
10.3 Getting Started with the Editors 10-3
10.3.1 vi 10-3
10.3.2 emacs 10-4
10.3.3 NEdit 10-6
10.3.4 nu/TPU 10-6
10.3.5 fermitpu 10-7
10.3.6 EDT+ 10-7
Chapter 11: UNIX Mail Systems 11-1
11.1 Mail Forwarding 11-1
11.1.1 The Fermilab Mail Server: FNAL 11-1
11.1.2 Forwarding on File-Sharing UNIX "Clusters" 11-2
11.1.3 Recommended Forwarding Procedure 11-2
11.2 Overview of Mail Systems Available at Fermilab 11-4
11.2.1 MH Graphical Interface: exmh 11-4
11.2.2 MH Line-Mode Interface: mh 11-4
11.2.3 pine 11-5
11.2.4 Berkeley Mail 11-7
11.3 The exmh and mh Mail Handlers 11-7
11.3.1 Running Setup and Invoking the Application 11-8
11.3.2 Composing and Sending Messages 11-10
11.3.3 Incorporating and Reading Incoming Messages 11-13
11.3.4 Replying to Messages 11-14
11.3.5 Forwarding Messages 11-15
11.3.6 Printing Messages 11-16
11.3.7 Extracting Messages 11-17
11.3.8 Removing Messages 11-17
11.3.9 Creating, Changing and Removing Folders 11-18
11.3.10 Refiling Messages 11-19
11.3.11 Searching for Messages 11-19
11.4 Basic Configuration for MH 11-20
11.4.1 Configuration Files 11-20
11.4.2 MH Mail Folders 11-21
11.4.3 Incorporation of Incoming Mail into Folders 11-22
11.4.4 Signature Lines 11-23
11.4.5 Mail Aliases 11-23

11.4.6 Folder Order and Header Display (exmh) 11-24
11.5 Berkeley Mail 11-24
11.5.1 Sending Messages and Files 11-24
11.5.2 Reading Messages 11-24
Chapter 12: Connecting to Remote Systems 12-1
12.1 Transferring Files 12-1
12.1.1 ftp 12-1
12.1.2 rcp 12-2
12.1.3 The .rhosts File 12-3
12.2 Logging in to Other Systems 12-4
12.2.1 telnet 12-4
12.2.2 rlogin 12-5
12.3 Executing Commands Remotely: rsh 12-5
Chapter 13: Software Development 13-1
13.1 Overview of Programming Languages and Tools 13-1
13.2 Introduction to C and FORTRAN on UNIX 13-4
13.2.1 The C Compiler: cc 13-4
13.2.2 The FORTRAN Compiler: f77 13-4
13.2.3 C and FORTRAN Compiling Basics 13-4
13.2.4 Linking Order 13-5
13.2.5 Displaying Active Options 13-5
13.2.6 Option Passing 13-5
13.3 Introduction to C++ on UNIX 13-6
13.4 C and FORTRAN Compiler Options 13-6
13.4.1 Commonly-Used Options 13-6
13.4.2 Recommended Options for General Use 13-7
13.4.3 Debugging Option 13-8
13.4.4 Portability Option for AIX 13-8
13.4.5 Speed Optimization Options 13-8
13.4.6 Load Map Option 13-9
13.4.7 Special FORTRAN Compiler Options 13-9
13.5 FORTRAN Programming 13-11
13.5.1 External Reference and Entry Point Names 13-11
13.5.2 Separate Compilation of FORTRAN Subprograms: fsplit . . . 13-11
13.5.3 AIX-Specific Issues 13-11
13.5.4 Loading Block Data Modules 13-11
13.5.5 Program Control 13-12
13.5.6 FORTRAN Source Code Analyzer: FLINT 13-12
13.5.7 Future FORTRAN Enhancements 13-13
13.6 Obsolete Programming Features 13-13
13.7 C and FORTRAN I/O 13-14
13.7.1 Records 13-14
13.7.2 Tapes 13-15
13.7.3 Standard Input and Output 13-15
13.8 Performance Tuning for C and FORTRAN 13-15

13.8.1 Optimization 13-15
13.8.2 Word Length 13-16
13.8.3 Feedback 13-16
13.8.4 Inlining 13-16
13.9 C and FORTRAN Mixed Programming 13-16
13.9.1 Variable Types 13-17
13.9.2 Array Indexing 13-17
13.9.3 External Names 13-17
13.9.4 Arguments 13-18
13.9.5 Commons 13-18
13.9.6 I/O 13-18
13.9.7 Linking 13-18
13.10 Executing a Program 13-18
13.11 Debugging 13-19
13.11.1 dbx 13-19
13.11.2 purify 13-20
13.11.3 CASEVision 13-21
Chapter 14: The make Utility 14-1
14.1 An Overview of the make Utility 14-1
14.2 The Makefile and its Components 14-2
14.2.1 Macros 14-2
14.2.2 Targets 14-3
14.2.3 Suffix Rules 14-4
14.2.4 Suffix Definitions 14-5
14.2.5 Control Files within a Makefile 14-5
14.3 Running make 14-5
14.3.1 General Usage 14-5
14.3.2 Usage without Specifying Target 14-6
14.3.3 Usage without a Makefile 14-6
14.4 “Housekeeping” Targets 14-6
14.5 Portability 14-7
14.6 make’s Built-in Rules 14-8
14.7 A Few Caveats... 14-8
Chapter 15: Code Management 15-1
15.1 CVS 15-1
15.1.1 Accessing CVS and Obtaining the Manual 15-1
15.1.2 Basic CVS Commands 15-2
15.2 UCM 15-2
15.2.1 Accessing UCM and Obtaining the Manual 15-3
15.2.2 Basic UCM Commands 15-3
15.2.3 kpp 15-3
Chapter 16: Tape Handling 16-1
16.1 Operator Communications Software (OCS) 16-1
16.1.1 Accessing OCS and Obtaining the Manual 16-1
16.1.2 OCS Basics 16-2

16.1.3 Reference for Selected OCS Commands 16-2
16.1.4 The OCS X Interfaces 16-6
16.1.5 Using Provided Examples to Get Started 16-7
16.2 Raw Buffered I/O (RBIO) 16-7
16.3 DAta From Tape (DAFT) 16-7
Chapter 17: Computing Resources and Projects 17-1
17.1 General Purpose UNIX Resource: FNALU 17-1
17.1.1 Intended Uses 17-1
17.1.2 Hardware 17-2
17.1.3 File System and Application Software 17-2
17.1.4 AFS on FNALU 17-2
17.1.5 FNALU Batch System: LoadLeveler 17-3
17.1.6 Tapes: OCS System 17-4
17.2 Clustered Large UNIX Batch System (CLUBS) 17-4
17.2.1 Integration of FNALU and CLUBS 17-4
17.2.2 CLUBS Components 17-5
17.2.3 CLUBS Batch Process Overview 17-5
17.2.4 Elements of Using CLUBS 17-6
17.3 Data Acquisition System (DART) 17-7
17.3.1 DART Functionality and Supported Interfaces 17-7
17.3.2 More about DART 17-8
Appendix A. VMS Migration for the Impatient A-1
A.1 The Two Necessary Commands A-1
A.2 OK, What's the Catch? A-1
A.3 Whoa! Too Fast! A-3
Appendix B. FUE Conformance Levels B-1
Appendix C. awk's Programming Model C-1
Appendix D. VMS to UNIX Command Reference D-1
D.1 UNIX Equivalents for VMS Commands D-1
D.2 Shell Scripts for Copying/Renaming Many Files D-4
D.3 Unpacking VMS Backup Save-sets D-5
Appendix E. Fermi Login Files E-1
E.1 C Shell Family E-1
E.1.1 .cshrc E-1
E.1.2 fermi.cshrc E-2
E.1.3 setpath.csh E-3
E.1.4 setups.csh E-4
E.1.5 .login E-6
E.1.6 fermi.login E-7
E.2 Bourne Shell Family E-11
E.2.1 .profile E-11
E.2.2 fermi.profile E-12
E.2.3 setpath.sh E-16
E.2.4 .shrc E-17

E.2.5 fermi.shrc E-19
E.2.6 setups.sh E-21
Appendix F. mh and exmh Customization F-1
F.1 Forwarding and Notification F-1
F.1.1 Forwarding Address F-1
F.1.2 Mail Notification F-2
F.2 Customizing mh and exmh F-3
F.2.1 .mh_profile F-3
F.2.2 components F-4
F.2.3 replcomps F-4
F.2.4 forwcomps F-5
F.2.5 scan-form and inc-form F-5
F.2.6 .maildelivery F-6
F.3 Unattended Autoincorporation F-8
F.4 Unattended Autoincorporation in AFS Environment F-9
Appendix G. mh Command Reference G-1
Appendix H. Mail Conversion from VMS H-1
H.1 Preparation for Conversion H-1
H.2 Choosing the Process to Use H-2
H.3 Using the Semi-Automatic Process H-2
H.4 Using the Automatic Process H-2
Appendix I. Programming Examples I-1
I.1 Interfacing C and FORTRAN I-1
I.2 Makefiles and the make Process I-3
I.2.1 A Simple make Process I-3
I.2.2 A Physics Makefile I-5

Chapter 1: Introduction

This chapter provides an introduction to the manual itself as well as to UNIX and its implementations at Fermilab. In particular, it covers:

- the purpose and intended audience of *UNIX at Fermilab*
- where to send comments and questions about the manual
- where to obtain additional copies of the manual
- the typeface conventions and symbols used throughout the manual
- a summary of the contents of the chapters and appendices
- a very brief history of the UNIX operating system and a discussion of its advantages
- a listing of UNIX references

1.1 The *UNIX at Fermilab* Manual

This manual is intended for all Fermilab users who are new to UNIX and for those of you who just need a reference once in a while. Users who are familiar with UNIX but new to the Fermilab implementations will also find useful information here.

> There are many different UNIX systems installed at Fermilab, with differing levels of conformance to Fermilab standards. Therefore, some of the information contained in this manual may not be valid for your system. You will very likely need to obtain supplementary documentation and information specific to your system.
>
> In particular, CDF and D0 have created documentation for their systems and made it available on the World Wide Web (this utility is described in section 3.2.1). You can access this information at the following locations:
>
> **CDF**
>
> **`http://www-cdf.fnal.gov/offline/cdfsga/cdfsga.html`**
>
> **D0**
>
> **`http://d0wop.fnal.gov/d0unix/d0unix.html`**

The purpose of this manual is to help you get started as quickly and painlessly as possible in the UNIX environment here at Fermilab, whatever your particular needs. We discuss the most commonly used UNIX concepts, commands and tools, and provide enough information to allow you to execute commands and perform tasks typically required by the Fermilab user community. We refer you to commercially available sources of information for details on commands and features that will allow you to exploit the more sophisticated features of the UNIX operating system not covered in this document. We also describe the Fermi UNIX Environment, which includes Fermilab-specific programs, libraries, tools, and usages, and occasionally refer you to other Fermilab documents.

We try to describe here a rather "standard UNIX" and recommend that you try not to use the "value-added" features that the individual vendors provide which may cause you trouble in conversion to another platform. Commands and features for both families of UNIX command interpreters, C shell and Bourne shell, are included when they differ. Since some of the systems at Fermilab will have the AFS file system installed (see sections 2.6 and 2.8.2), we address some issues concerning AFS.

It is *not* the purpose of this manual to teach you UNIX or to give you a complete description of UNIX. For that purpose we recommend that you obtain and read one of many good books on UNIX. See section 1.5 for some suggestions. This manual is also not intended as a system administration reference.

For those readers who wish to convert from VMS to UNIX quickly without learning about UNIX ahead of time, see Appendix A for the bare minimum needed to accomplish this task!

UNIX at Fermilab is bound to contain some errors, however we endeavor to minimize the error count! We encourage all the readers of this manual to report back to us:

- errors or inconsistencies that we have overlooked
- any parts of the manual that are confusing or unhelpful -- please offer *constructive* suggestions!
- other topics to include (keeping in mind the purpose of the manual)
- tricks that other Fermilab users might find helpful

Send your comments via email to *cdlibrary@fnal.gov.*

Copies of *UNIX at Fermilab* can be obtained from the following sources:

WWW[1]	PostScript files of the individual chapters and appendices are available for viewing and/or printing under the URL: `http://www.fnal.gov/docs/UNIX/unix_at_fermilab` You can reach this from the Fermilab Computing Division home page, under *UNIX Resources* (see section 1.5.1). An HTML version of this document is also available from the *UNIX Resources* page.
Paper Copies	Computing Division library, Wilson Hall, 8th floor, NE; document number GU0001.
AFS	PostScript files of the individual chapters and appendices are available at `/afs/fnal/files/docs/UNIX/unix_at_fermilab/ps`

1.1.1 Conventions Used in this Manual

The following notational conventions are used in this manual:

bold	Used for shells (e.g., **csh**) and product names. Used in text to emphasize an important new word or concept.
italic	Used to emphasize a word or concept in the text; many terms found in the index are italicized in the text. Also used to indicate variables and reference book titles.
`typewriter`	Used for filenames, path names, contents of files, output of commands.

1. See section 3.2.1 for information on the World Wide Web (WWW).

SMALLCAPS	Used to indicate keys (on keyboard) and "buttons" on graphical applications.
`typewriter-bold`	In text, used to indicate commands and prompts. In command formats, indicates what the user types "as is".
bold-italic	In command formats, indicates variables for which the user must make context-specific substitutions.
<CTRL-*char*>	Indicates a control character. To enter a control character, hold down the control key (labeled Ctrl, probably) while pressing the key specified by ***char***.
[ ]	In command formats, indicates **optional** command arguments and options.
|	When shown in between brackets (**`[x|y|z]`**) in a command line, separates a series of options from which one must be chosen. In UNIX command formats, used to "pipe" output of preceding command to the following one (see section 5.3.3).
' ... '	Single vertical quotes indicate apostrophes in commands.
` ... `	Single backquotes in UNIX commands have a special meaning. Don't confuse them with apostrophes.
. . .	Means that a repetition of the preceding parameter or argument is allowed.

All command examples are followed by an implicit carriage return key.

The following symbols are used throughout *UNIX at Fermilab* to draw your attention to specific items in the text:

A UNIX "bomb"; this refers to something important you need to know about UNIX in order to avoid a pitfall.

A VMS "bomb"; this is used to point out common pitfalls of VMS users.

This symbol indicates information for AFS systems (see section 2.6).

This symbol is intended to draw your attention to a useful hint.

This is a reminder.

We refer you to other documentation sources for further information where you see this symbol.

1.1.2 Summary of Chapters and Appendices

The organization of this manual reflects our compromise between providing a tutorial and a reference manual. You may find it useful to skip around a little rather than reading straight through. We recommend that you get a UNIX account and begin to work with the system as you read.

Chapters:

2: Getting Started on a UNIX System

This chapter describes the login and logout procedures, and discusses some basic UNIX features including the prompt, special keys, and special characters. We present a brief discussion of the file systems you are likely to encounter at Fermilab, and how to change your password according to your file system and installation.

3: Information Resources

This chapter introduces you to the information facilities available from UNIX. Standard UNIX on-line help is available via the man pages. We discuss the World Wide Web and newsgroups, which are very rich sources of information on a virtually unlimited set of topics. A few utilities that allow you to get information about vendor products, other users, and the Fermilab computing systems are also covered.

4: Shells

This chapter discusses the concept of a UNIX shell, and how to manipulate shells. It includes information on the available and recommended shells and their features. The concept of a shell as an interpretive programming language is introduced.

5: UNIX Commands and Concepts

This chapter introduces you to the UNIX command structure, and to many important commands and concepts. The features introduced in this chapter constitute the core of the UNIX operating system, and many of these tools are quite powerful and flexible. Some of the features are shell-specific, and we provide the distinctions where necessary.

6: The UNIX File System

The UNIX file system has a hierarchical or tree-like structure with the directory called *root* (**/**) as its source. The system is essentially composed of *files* and *directories*. In this chapter we describe techniques for manipulating files and directories, and commands designed to provide information about them.

7: Printing

This chapter covers the standard FUE print utility **flpr**, as well as filter programs and techniques available for formatting the output prior to printing. **flpr** is the Fermi implementation of the standard UNIX **lpr** utility. Most software applications supplied by the Computing Division use **flpr** as a default.

8: Working Environment

This chapter describes the methods used to set up your working environment in UNIX. Some of these are standard UNIX (e.g., shell and environment variables), and some are provided and/or customized by FUE (e.g., the login scripts).

9: UPS Products and Applications

This chapter first discusses the Fermilab product support structure, and shows you how to get information about supported products (referred to as **ups** products) and applications. We then describe how to use available **ups** products, and how to obtain ones not yet installed on your system.

10: Editors

Several text editors are available at Fermilab. In this chapter we present our view of the advantages and disadvantages of the available editors, and we provide some basic information on the setup and use of each one. You will learn how to invoke each editor, and how to create, edit, and save a file in each one using a small subset of commands and features. We include only minimal usage information for the VMS-style editors.

11: UNIX Mail Systems

This chapter is designed to introduce you to the available mail handling systems at Fermilab. We concentrate on the UNIX **MH** (Message Handling) system which the Computing Division recommends and fully supports. Fermilab-supported **MH** products include both the standard line mode mail reader (referred to as **mh**), and a GUI version of **MH** called **exmh**. In this chapter, our goal is to get you up and running in **mh** or **exmh** as quickly and easily as possible, with a minimum of up-front configuration.

The Computing Division has recently begun supporting an additional non-graphical UNIX mail utility, **pine**. **pine** functionality is not documented in the present manual, but we provide some start-up information for you, and point out where you can get more information. You'll also find a brief discussion of **Berkeley mail** at the end of the chapter, only because it is a mail reader available on all UNIX machines.

12: Connecting to Remote Systems

Several utilities are available to enable you to transfer files between systems, to log into other systems on which you have an account, and to execute commands remotely. These features are described in this chapter.

13: Software Development

This chapter gives an introduction to UNIX software development tools in common use at Fermilab, providing information on:

- Supported languages
- Compiling and linking in C, C++ and FORTRAN
- Debugging

We do not include a discussion of general programming here, but rather, aspects of software development particular to UNIX.

14: The make Utility

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. Although **make** can be used in a wide variety of applications, in this chapter we concentrate on its use in the area of software development. We describe how to define relationships between source, object, library and executable files for use by **make**, and how to invoke **make** in its simplest and slightly more complex forms.

15: Code Management

This chapter introduces the two recommended code management solutions for UNIX, **CVS** (Concurrent Versions System) and **UCM** (UNIX Code Management). Both use **RCS** (Revision Control System) as the underlying protocol. We provide basic information only, and refer you to the complete manuals for these utilities for detailed information.

16: Tape Handling

In this chapter we introduce you to the principal tape handling software and facilities available at Fermilab, and refer you to detailed documents on each subject for more information. Since many groups and experiments have standardized and/or customized their tape-handling routines, some of the options described here may not be available or appropriate for you. Contact your group leader to find out what procedures have been established for your group.

17: Computing Resources and Projects

It is our intention in this chapter to make you aware of resources available through the Fermilab Computing Division, and projects of general interest to Fermilab UNIX users. In particular, we discuss FNALU, CLUBS and DART. We provide references in case you wish to investigate any of these topics further.

Appendices:

A: VMS Migration for the Impatient

So, you've decided you're ready to convert (or you've run up against a deadline!), but you don't know the first thing about UNIX. Here's enough information to get you moved over. You can use the rest of the manual to learn about UNIX afterwards.

B: FUE Conformance Levels

The Fermilab Computing Division has defined a set of FUE conformance levels. We describe them in this appendix.

C: awk's Programming Model

This appendix is adapted from a section of the same name in the book *sed & awk*, published by O'Reilly & Associates. It describes the generic structure and organization of an **awk** program.

D: VMS to UNIX Command Reference

This appendix is intended as a convenient UNIX command reference for the migrating VMS user. The UNIX equivalents of commonly used VMS commands are listed in tabular format. No information on syntax, options, or arguments is presented here, however some of the listed commands are described elsewhere in *UNIX at Fermilab.*

E: Fermi Login Files

This appendix contains file listings of the FUE-customized default login files (the "Fermi Files") used to set up your UNIX environment. The files are reprinted in their entirety except for the copyright disclaimers.

F: mh and exmh Customization

This appendix contains information for further customizing the **mh** and/or **exmh** mail readers. Given the multitude of customizable features in these mail readers, we cannot provide you with a comprehensive treatment of the subject here. The information presented in Chapter 11 and this appendix should be sufficient for most users.

G: mh Command Reference

This appendix provides an alphabetical reference to the subset of **mh** commands discussed in Chapter 11.

H: Mail Conversion from VMS

This appendix is intended to guide you through a mail conversion process from VMS to the UNIX **MH** (Message Handling) system. Two options for conversion are presented.

I: Programming Examples

This appendix contains examples of programs that illustrate information presented in Chapters 13 and 14.

1.2 About UNIX

1.2.1 A Brief History

The UNIX operating system was developed at AT&T Bell Laboratories in Murray Hill, New Jersey in the late 1960's. Universities and colleges have played a major role in popularizing UNIX. The Computer Science Department at the University of California at Berkeley made so many popular

changes to it that one of the two most popular versions in use today is named the Berkeley Software Distribution (BSD) of the UNIX system. The other major version is AT&T's UNIX System V. Many implementations incorporate features of both systems.

1.2.2 A Word About Features and Components

Because UNIX was originally designed by programmers to support their own projects, one of its strongest points is that it provides an excellent software development environment. UNIX has a large set of powerful utility programs and tools that allow users to easily build systems and applications. It also has several command interpreters, called *shells* that can also be used as high-level programming languages. Keep an open mind to the powerful features of UNIX that may be quite different from systems you are familiar with. Some of the important concepts are introduced in Section 5.3 (e.g., pipes and filters, and device-independence).

Bear in mind that there are many UNIX utilities not described in this guide.

The UNIX operating system has four basic components:

- The *kernel* constitutes the nucleus of the operating system and coordinates the internals such as allocating system resources.
- The *file system*, which is part of the kernel, controls the storage and access of data. It is similar to the VMS file system in that the structure is hierarchical.
- *Commands* are programs that you request the computer to execute.
- Programs (commands) called *shells* serve as command interpreters. They act as links between you and the computer, interpreting and executing commands. They are also high-level interpretive programming languages.

Although there are many implementations of UNIX, there are two families of shells. The Bourne shell family and the Berkeley/C shell family. The shell families and the individual shells are discussed in more detail in Chapter 4.

1.2.3 The Advantages of UNIX

New Technologies

UNIX allows Fermilab to achieve better alignment with the mainstream of computing. The "developmental momentum" is with UNIX and personal computers, and it is primarily for these environments that the new technologies, software products, and software methods are emerging. Other HEP laboratories have also recognized these trends and are moving in this direction.

Multi-Platform Support

There are many UNIX platforms, and while there are certainly variations among them, the differences tend to be small. This allows the Computing Division to more effectively support platforms supplied by many vendors, thus eliminating the link to a single vendor whose pricing policies and technology directions are outside of our control.

Availability of Popular Applications

The general administrative and much of the technical computing load has migrated to MACs and PCs due to the availability of commercial applications. There is also a growing demand from HEP researchers for these applications. The ability to use personal computers as X-terminals offers a real possibility of providing a single platform that allows access to popular applications for word-processing, preparation of presentations, charting, statistical analysis, etc., as well as access to powerful UNIX systems for data analysis and program development.

Choice of Machine and Platform

You first need to determine if you are going to purchase your own desktop UNIX system, get an account on a workgroup UNIX system, or get an account on the central Fermilab UNIX system, FNALU. At this point, you may or may not have a choice of platform; for instance on FNALU you can choose between IBM RS/6000, Silicon Graphics, Sun SPARCserver, and Digital UNIX Alpha. UNIX exists in different *flavors* on the different vendor platforms, meaning that the UNIX operating system is somewhat customized to each platform. The differences between flavors tend to be small.

Customizable Environment

The "brave new world" of UNIX allows you to set up your work environment and choose your tools to suit your own individual needs. At Fermilab we have been careful to implement and support what we consider to be the most useful of the wide range of available UNIX utilities appropriate for our environment. Still, you will find a much larger array of options than you may be accustomed to. We provide you with defaults for most options, thereby allowing you to get used to the new system gradually, and build your knowledge and proficiency at your own speed.

In order to take full advantage of the flexibility offered, you need to be aware of the options, and you need to make some choices. Two choices you as an individual user need to make initially are the editor and the mail system you want to use. Of course you can change to other ones later. After that, you have the choice of delving further into the marvels of UNIX, or accepting the provided defaults so you can just get to work. UNIX is in fact usable by ordinary mortals as well as by sophisticated hackers! Throughout this manual we give some guidance for making choices.

The available editors come in three basic types:

- modern point-and-click style editors (for graphics-capable terminals) similar to those available for PC Windows and Macintosh
- UNIX editors that provide powerful features for sophisticated editing needs, but that can also be used for simple editing tasks using a small subset of commands
- VMS-style editors

The types of mail systems offered include:

- basic UNIX mail readers
- more fully functional "traditional" mail systems
- modern point-and-click mail readers

Instead of a single command interpreter and user interface, several shells have been developed for UNIX, a number of which we provide and support. Until you get a good idea of what the advantages are of one shell over another, we recommend that you just use the default shell provided. You can change it later.

The choices that you need to make should not be overwhelming. Most people find the range of choices empowering once they're familiar with the system.

1.3 The Fermi UNIX Environment

The on-site computer systems are used for a wide range of tasks; for example general purpose interactive use, batch jobs, farms[1], and data acquisition systems. In order to most efficiently accommodate the needs of our user community, the Computing Division provides a single platform-independent UNIX environment known as the *Fermi UNIX Environment,* or *FUE.*

The Fermi UNIX Environment (FUE) is a set of products installed on most UNIX systems at Fermilab that defines the computing environment. One of the main goals of FUE is to provide as much as possible the same environment on the different UNIX platforms. A second important goal is to provide a standard product support methodology.

The FUE project team is composed of people from several different departments within the Computing Division, and some researchers from other divisions. Since the FUE project's inception as an outgrowth of the 1990 UNIX work station SEED project, it has been able to provide a variety of services and products to the UNIX user community at Fermilab. The FUE project established the UNIX product distribution area (**upd**, see section 9.4.1), and developed a process for distributing proprietary products. It provides a certification process for new operating system and compiler releases, coordination of multiple shell support, and recommendations on UNIX code management.

FUE can be divided into three main areas:

- A set of login scripts, called the *Fermi files* that provide a common environment. These are described in section 8.4 and in Appendix E.
- Infrastructure to allow access to software products: **ups** (UNIX Product Support) and **upd** (UNIX Product Distribution). These are described in Chapter 9.
- General utilities, system administration tools, and software products in **ups** format.

UNIX systems at the lab or at users' home institutions can be set up to be FUE-compliant. There are several levels of FUE compliance available; these levels are described in Appendix B.

1.4 UNIX Support at Fermilab

The Fermilab Computing Division supports many different flavors of UNIX operating systems. The Computing Division currently supports the following platform and operating system combinations[2]:

- Silicon Graphics running IRIX
- IBM running AIX
- Digital running OSF1
- Sun Microsystems running SunOS

For the currently supported releases of the operating systems, C compilers and FORTRAN compilers, refer to the document DR0010 (see section 1.5.1 for information on how to locate Fermilab documents). This information changes frequently.

1. Farms are clusters of RISC-based workstations for which parallel processing software has been developed to support activities such as event reconstruction and Monte Carlo.
2. This information is taken from document DR0010 updated 9/13/95.

1.5 UNIX References

1.5.1 Fermilab Documents

Unless stated otherwise, the Fermilab documents that we reference throughout this manual can be found in the Computing Division library, Wilson Hall, 8th floor, NE. Many of the documents are now also available on the World Wide Web (WWW), an on-line information resource to which we provide an introduction in section 3.2.1.

As a general rule throughout this manual, we do not provide the URL for each referenced document due to the volatility of the addresses. Navigation through the Computing Division's WWW pages is fairly straightforward. From the *Fermilab at Work* page, at URL:

`http://www.fnal.gov/faw/fermilab_at_work.html`

go to the heading **Divisions & Sections** and select *Computing Division* as the place to start your search. The Computing Division home page can also be accessed directly at URL:

`http://www.fnal.gov/cd/`

UNIX Documentation

Most of the documents to which we refer you can be found from the Computing Division home page under the heading **Information Resources**. Under *General Documentation and information about offline applications* you'll find the *Computing Division Offline Document Database* which is a general repository of Computing Division documents, both UNIX-related and non-UNIX. Its URL is:

`http://www.fnal.gov/cd/OfflineDocs/general.html`

Most of these documents are available for viewing and/or printing in either HTML (see section 3.2.1) or PostScript format. This includes document numbers starting with GG (General Documentation), GU (General UNIX), DR (Division Recommendations) and some other designations.

Lots of information has been gathered under *UNIX Resources*, organized by topic. The URL for *UNIX Resources* is:

`http://www.fnal.gov/cd/UNIX/UnixResources.html`

Software Products Information

Documentation for many Fermilab software products is maintained in an Oracle database accessible from the *Computing Division Offline Document Database* under the heading **Product and Application Information**. The document numbers correspond to the product numbers, and start with PM (Physics and Math Programs), PU (Utility Programs), and other designations. See section 9.2 for more information on this database.

Alternatively, you can go to URL `http://www.fnal.gov/docs/products/` to find UNIX product documentation from a variety of sources directly by product name.

1.5.2 Commercially Available Texts

There are several good books on UNIX. The vendors also have UNIX documentation, for example the Silicon Graphics *IRIS-4D User's Guide* and Sun's beginner's guides. Printed reference documentation for the commands is typically just hard copy of the UNIX on-line help, known as *man pages*, described in section 3.1. Some popular books are:

- *UNIX System V: A Practical Guide* by Mark Sobell, published by Benjamin/Cummings
- *UNIX in a Nutshell* by Daniel Gilly et al, published by O'Reilly & Associates, Inc.

- *UNIX for the Impatient* by Paul W. Abrahams and Bruce R. Larson, published by Addison-Wesley
- *UNIX for VMS Users* by Philip Bourne, published by Digital Press (Note that it has a predictable and detectable ULTRIX[1] bias!)

The O'Reilly & Associates, Inc. publishers provide specialized texts on many UNIX applications in addition to their general UNIX texts. These are generally regarded as excellent references.

From the *UNIX Resources* WWW page, see *Recommended Books* under **The UNIX Operating System** for information on what's available in the Computing Division library and the stockroom. These books can be found in many bookstores, too.

1. ULTRIX was Digital's early UNIX operating system. It has been replaced by OSF1.

Chapter 2: Getting Started on a UNIX System

This chapter describes the login and logout procedures, and discusses some basic UNIX features including the prompt, special keys, and special characters. We present a brief discussion of the file systems you are likely to encounter at Fermilab, and how to change your password according to your file system and installation.

2.1 Logging In

The following are a couple of ways that you can access a UNIX system:

- Log into the workstation console directly (in other words, it is on or under your own desk). Consult the instructions for your workstation.
- Connect to it over the network; enter[1]:

  telnet ***host***

The system will prompt for your *login name* and for your *password*. Below is a sample login session from an X terminal to node FSGI01 of the FNALU system (note that the system prompt is set to the node name here):

```
Connecting to host "fsgi01".....success.


IRIX System V.4 (fsgi01)

login: username
Password:(password is not shown)
IRIX Release 5.2 IP7 fsgi01
Last login: Tue Oct 24 09:27:14 by UNKNOWN@dcdx03.fnal.gov
Terminal Type is xterm


There are no available articles.
<fsgi01>
```

The login name must be all lower case characters, but the password may contain upper and lower case characters. Be sure to enter them in the correct case, because **UNIX is case sensitive**. Note that if you do use upper case to log in, UNIX may assume you have an upper-case-only terminal and you will have very limited capability. If you do so, either log out and log back in again, or enter the command:

```
% stty -lcase
```

---

1. **telnet** is described in section 12.2. Also described there is the utility **rlogin**, which can be used instead of **telnet**.

When you log in, a series of login scripts is run to define the functionality of your terminal and to set up your environment. These startup files have been customized under FUE and are called the *Fermi files*. They provide the common environment described in Chapter 8.

At login, the system will type out:

`Terminal Type is` *`termtype`*

where *`termtype`* represents the best guess depending on exactly how and from where you entered the system. To change this terminal type, see section 8.6.

2.1.1 C Shell Family

The C shell runs two files when you log in, first a file named `.cshrc` and then a file named `.login`. They are both located in your home directory. See Chapter 8 for details.

2.1.2 Bourne Shell Family

In the Bourne shell family the `.profile` and `.shrc` files, located in your home directory, are run when you log in.[1] See Chapter 8 for details.

2.2 Logging Out

The logout commands are slightly different for the two shell families. For either one, you can create a `.logout` script and use it in place of the standard commands.

2.2.1 C Shell Family

You can logout with either:

`% logout` or

`% exit`

If you have other processes running you will be informed that you have stopped jobs. You can continue to enter **`logout`** until all the processes are terminated.

The control character **eof**, which is usually set to **<CTRL-D>**, is the normal UNIX way of logging off. Since this is easily entered accidentally, the standard Fermi UNIX Environment disables this way of logging out by including the command **`set ignoreeof`** in the Fermi files.

2.2.2 Bourne Shell Family

You can logout with:

`$ exit` or

`$ <CTRL -D>`

If you have other processes running you will be informed that you have stopped jobs. You can continue to enter **`exit`** until all the processes are terminated.

1. In a non-FUE environment, the Bourne shell (**sh**) does not run `.shrc`.

The control character **eof**, which is usually set to **<CTRL-D>**, is the normal UNIX way of logging off. Since this is easily entered accidentally, the standard Fermi UNIX Environment includes the command **`set ignoreeof`** in the Fermi files to disable it (*ignoreeof* is not supported on all Bourne shells).

2.3 The UNIX Prompt

After you log in successfully, you will get a prompt. The default UNIX prompt usually indicates your default shell (see Chapter 4 for information on shells). Typically, a **`%`** indicates the C shell (**csh**) and a **`$`** indicates the Bourne shell (**sh**). The prompt on your machine may be set differently, however, and may not indicate the shell. The Fermi files set the prompt to the machine name.

You can change your default prompt by altering your startup files.

In the C shell, include in your `.login` file:

`set prompt='`*`newprompt`*`'`

In the Bourne shell, you need to set the value of the keyword shell variable *PS1*. This is just a variable that is declared and initialized by the shell at startup. Include in your `.profile` file:

`export PS1; PS1="`*`newprompt`*`"`

2.4 Special Keys

Like all systems, UNIX has a number of special keys that perform particular functions. Some important ones are the keys necessary to backspace over a character when entering a command, to delete the whole line being entered, and to interrupt execution. These are user-specifiable, and have different defaults based on the shell, the version of UNIX, and the login files described in section 8.4. If you are using the Fermi files, these special keys will be set as described in the table (notice the footnotes labelled *a* and *b* that follow the table):

Name	Control Char	Function
erase	**DEL** or **BACK-SPACE** [a]	Erase character. Backspace and erase one character (the key used depends on terminal setting).
werase	**<CTRL-W>**	Delete the rightmost word typed in.
kill	**<CTRL-X>**	Kill (erase) the line typed in so far.[b]
intr	**<CTRL-C>**	Interrupt the program currently running.
rprnt	**<CTRL-R>**	Reprint the line typed in so far.
flush	**<CTRL-O>**	Stops terminal output until you press a key.
susp	**<CTRL-Z>**	Suspend the program currently running and put it in the background. **This does not stop the process as it does in VMS!**
stop	**<CTRL-S>**	Stop the display. To resume, press the **start** key
start	**<CTRL-Q>**	Start the display after stop.

Name	Control Char	Function
eof	**<CTRL-D>**	Send the program an end-of-file character.

a. In some unusual circumstances of setup and keyboards you may also need to issue this set of commands to get the backspace key to work as expected:

```
% stty erase "^?"
% stty intr "^C"
% stty kill "^X"
```

b. If you prefer to use **<CTRL-U>** for this function, uncomment the line `#stty kill '^u'` in your `.login` file (C shell family), or `#stty kill` in your `.profile` file (Bourne shell family).

To display the current settings for your terminal, enter:

```
% stty -a
```

If the keys don't seem to work as described here or you want to change them, refer to section 8.6.

UNIX relies on the hardware tabs of your terminal. If they are not set or if they are set in an unusual way, displays may appear strange on your terminal. You can set the tabs manually on your terminal, or you can use the **tabs** command to set them. The command with no arguments will set tabs in the usual UNIX way, 8 spaces apart.

```
% tabs
```

2.5 Special Characters (Metacharacters)

Slashes

The backslash (****) character is used on the UNIX command line to mask the special meaning of the character immediately following it (no spaces in-between) so that the command interpreter takes the character literally. For example:

% *command* \<*CR*>

causes the carriage return to be ignored. This is a useful way to continue commands to the following line.

The forward slash (**/**) character is the symbol for the root directory. In path names it acts as a separator between directories in the hierarchy, and between the last directory and the file, if one is specified. For example:

/rootdir/. . ./userdir/subdir1/subdir2/filename

Quotes and Parentheses

Different types of quotes have special meanings.

Normal single quotes (apostrophes) around a string (**'string'**) tell the command interpreter to take the string (**string**) literally. Double quotes around a string (**"string"**) also tell the command interpreter to take the string literally, but allow interpretation of variables that follow a **$** character (**$** preceding a variable name outputs the value of the variable; see section 8.1). Section 5.1.2 further explains single and double quotes in command interpretation, and provides an example.

Single backquotes around a command string (**`string`**) tell the interpreter to run the command(s) in the string, and to use the output of the command(s) in place of the string itself. This is useful for combining two commands into one, and for doing iterative tasks within shell scripts (shell scripts are introduced in section 4.4).

A string of commands enclosed in parentheses (**(string)**) is run in a subshell. In section 5.1.1 we discuss the difference between shell commands and non-shell commands. A non-shell command always runs in a separate shell (a subshell); when enclosed in parentheses, the command starts a second subshell.

Command Separators

The semicolon (**;**) character separates successive commands on a single command line. For example,

% *command1* ; *command2*

executes ***command1***, and when it finishes, ***command2*** gets executed.

The ampersand character (**&**) is similar to **;** but does not wait for ***command1*** to finish.

A double ampersand (**&&**) runs ***command2*** only if ***command1*** was successful.

Piping commands (the pipe symbol **|**) is discussed in section 5.3.3. A pipe tells ***command2*** to use the output of ***command1*** as input.

A double pipe (**||**) runs ***command2*** only if ***command1*** was *unsuccessful*.

Other Special Characters

Special characters are used in file expansion (section 6.2.2). Note that to prevent file expansion, these characters must be prefaced by a backslash (****). Metacharacters are also used in input/output redirection (section 5.3.2), and in regular expressions (section 5.3.5) as wildcards, delimiters, and other special pattern-matching characters. Refer to these sections for specific information.

2.6 File Systems: Standard UNIX and AFS

The standard UNIX file system is a hierarchy of directories descending from what is known as the *root directory.* UNIX allows parts of the directory hierarchy, also called file systems, to reside on separate storage devices or in separate disk partitions. These are accommodated by means of mount points. A *mount point* is a directory in a file system that corresponds to the root directory of some other file system. The primary file system is the one starting at the true root. The standard UNIX file system is described in Chapter 6.

If the UNIX machine that you work on is part of an integrated system of UNIX machines, for example a LAN (local area network), it is likely that a distributed file serving system has been installed on it. A distributed file system provides a common directory structure and thus the same view of the file system to all participating nodes. This overrides the standard UNIX file system.

Fermilab is using the AFS (Andrew File System) as a distributed file service model, and it is installed on several machines at Fermilab in a production environment, including the FNALU cluster. AFS is described in detail in the *FNALU User's Guide* (document number GU0008). For a local collection of information regarding AFS, look under **The UNIX Operating System** on the *UNIX Resources* WWW page.

A special note to CDF and D0 users: Your UNIX systems are *not* configured to use AFS.

Determining if AFS is installed on your machine can be accomplished in two steps. First enter **cd /afs** to attempt to change to the `/afs` root directory. If it's not there, AFS is not installed. If the directory is there, you still don't know if AFS is actually installed, or if it's running in *translator*

mode[1]. At this point, enter the command **`df |grep afs`**. If no output is returned, AFS is not installed on your machine. If output is returned, and the output line begins with `/afs`, for example:

```
/afs              (AFS              ):18000000 blocks  9000000 files
```

then AFS is installed on your machine. If the output line begins with a node name preceding `/afs`, for example:

```
fsut01:/afs                          144000000          0   144000000     0%
```

then AFS is not installed on your machine; it is running in translator mode.

2.7 Information Distribution System: NIS

NIS (Network Information System) is a system that distributes information throughout a cluster. You may also know this by the name *yellow pages*.

Note that the word *cluster*, as used in this manual in reference to UNIX, is not the same as a VAX cluster. We define a UNIX cluster as a group of machines that share both a common password file (or user database), and a common file system, especially for login directories. NIS is usually used to provide the common password file, and the common file system is typically NFS or AFS.

NIS is installed on FNALU and many other UNIX clusters at Fermilab. In order to determine if NIS is running on your system, execute the command:

```
% domainname
```

If it returns a value, then NIS is running on your cluster. If no output is returned, then it is not. Many UNIX clusters use NIS to share a common login area across several machines. Note that it is possible for both AFS and NIS to be installed on a single system.

2.8 Changing Your Password

For any password changes, you will be prompted for your old and new passwords. Your password must be at least six characters long and should not be your login name or any simple permutation of it. We recommend that you limit your password to eight characters, especially if it is used as an AFS password[2]. It is advisable to mix letters and digits in your password.

If your system is configured in a way not covered in the following sections, you may need to contact your local system manager to determine which password(s) to change.

Man pages are available for the commands described below: **`passwd`**, **`yppasswd`**, and **`kpasswd`**.

2.8.1 Standard UNIX Password

This section assumes that AFS is not running on your system.

Your UNIX password is used to log you into the system. This password is stored in one of two ways, depending on the configuration of your system.

---

1. In translator mode, a “translator machine” runs AFS and exports the AFS file tree to other systems.
2. This enforces consistent behavior in a multi-vendor AFS environment. On some platforms the login program may truncate the password after eight characters in which case the system can’t provide you access to the file system. (This is discussed in section 2.8.2).

If your system uses the standard UNIX file system, and NIS is not installed, your UNIX password is part of your local password file. In this case you need to execute the generic UNIX password-changing command to change your password:

```
% passwd
```

The system will prompt you for the necessary information. Your password is only changed for the machine, or node, on which you execute this command.

If NIS is running on your system, your UNIX password is stored in the NIS password file which is shared by all machines on the NIS cluster. In this case, you need to use the command **yppasswd** to change your password. The system will prompt you for the necessary information. When you change your password in this way, it changes for all the machines (nodes) that are part of your cluster.

2.8.2 AFS (Kerberos) Password

If AFS is installed, you are required to have what is known as a *Kerberos token* in order to access files in the AFS file system[1]. This is a security measure. You must obtain a token every time you log in and want to work with the file system. You will have an AFS password (also known as a *Kerberos* password), for any account on a system that uses the AFS file system, for example FNALU. The AFS password, which you enter at login time, allows two operations to proceed:

- You can log in to the system
- Your *Kerberos token* is automatically obtained via the **klog** program to allow you access to the file system

You can change your AFS password using the command:

```
% kpasswd
```

The system will prompt you for the necessary information.

This command changes your AFS password for all systems that run AFS on-site.

Under normal circumstances, the AFS password is the only password you need for an AFS system. A special circumstance would be if you log in via a method that doesn't understand AFS (Kerberos) passwords (e.g, MAC-X to some UNIX platforms). In this case, after login, you would need to run a separate program, **klog**, to obtain access to files. Type the command **klog**, and it prompts you for your AFS password.

Another situation in which you would need to run **klog** explicitly occurs if your token expires. You may need to run **klog** if you've been logged in for more than 24 hours, since tokens are set to expire at that point. Simply type **klog**, and enter your AFS password when prompted.

Depending on how the AFS file system was installed, you may or may not have a standard UNIX password in addition to your AFS one. On FNALU, for example, AFS was installed so that you only have an AFS password. If you have both, at login you should provide the AFS password so that you obtain your Kerberos token and are granted access to files. If instead you provide your standard UNIX password, the system will log you in, but you will not obtain a token and thus will not be able to access AFS files. If the passwords are the same, the AFS password automatically takes precedence.

You can find out if you have a standard UNIX password by attempting to change it via the standard UNIX command **passwd**. If it returns the lines:

```
passwd:  Changing password for username
passwd:  username does not exist.
```

---

1. Generally speaking, this includes files whose paths start at `/afs`.

where *username* is your login id, then you do not have one.

Chapter 3: Information Resources

This chapter introduces you to the information facilities available from UNIX. Standard UNIX on-line help is available via the man pages. We discuss the World Wide Web and newsgroups, which are very rich sources of information on a virtually unlimited set of topics. A few utilities that allow you to get information about vendor products, other users, and the Fermilab computing systems are also covered.

3.1 UNIX On-Line Help

3.1.1 man Pages

On-line help for UNIX system commands and utilities is in the form of *man pages* (*man* stands for manual) which consist of an on-line version of the UNIX documentation set (often called the UNIX Programmer's Reference Manual). You access the man pages with the **`man`** command.

Note that the man pages differ in many instances between UNIX platforms. Also note that to access the man pages for **ups** products (see Chapter 9), you may have to *setup* the product first.

The man Command

When you need help on a known command, use the general **`man`** command format:

`% man [`*part*`]` ***topic***

where ***topic*** is generally a UNIX command. **man** is really the on-line manual which is divided into several parts. ***part*** is a digit between 0 and 9. If you know in advance which part contains the information you want, you can speed the search by specifying it. More often than not you will just enter:

`% man` ***topic***

The word *print* in **man** entries usually means *display on the screen*. Don't be confused by this. Several options are available with the **man** utility, described under **`man man`**. The **`man`** command normally displays complete manual pages that you select by name. One-line summaries can be selected by either by keyword (**`-k`** option), or by the name of an associated file (**`-f`**). These options are described in section 3.1.2.

A typical initial **man** screen can be seen by issuing the command:

```
% man ls
```

where **`ls`** is the UNIX command to list files in a directory.

```
ls(1)                       User Commands                       ls(1)



NAME
     ls - list contents of directory

SYNOPSIS
     ls [ -abcCdfFgilLmnopqrRstux1 ] [ names ]

AVAILABILITY
     SUNWcsu

DESCRIPTION
     For each directory argument, ls lists the  contents  of  the
     directory;  for  each file argument, ls repeats its name and
     any other  information  requested.   The  output  is  sorted
     alphabetically  by  default.  When no argument is given, the
     current directory is listed.   When  several  arguments  are
     given,  the  arguments  are  first sorted appropriately, but
/tmp/mpa002Zf
```

Man pages are typically formatted with the UNIX text processing utility **nroff** (or **groff**). These utilities are covered in most UNIX texts. If you find that the **man** page is unformatted, run **`setup groff`**, and then rerun the **`man`** command.

Note that built-in shell commands are described under the topic corresponding to their shell.[1] (See section 5.1.1 for information on built-in commands.) For example, to get information on the command **`alias`** for your current shell, you would enter **`man`** ***shell*** (e.g., **`man bash`**) and search there for information on **`alias`** using the ***/pattern*** function described below.

There is an alternate, "quick and dirty" method to verify the format of a command and get a listing of its options. It doesn't work with all commands, but is usually worth a try. Simply enter the command with an illegal option (try **`/`** or **`?`** or **`.`**). For example,

```
% ls -/
```

will produce the output:

```
ls: illegal option -- /
usage: ls -RadCLHxmnlogrtucpFbqisf [files]
```

Manipulating man Pages

man displays the page using your *$PAGER* environment variable, which under FUE is set to **less** (see section 5.3.4). Therefore, **man** pages are normally piped to **less.** The command **`man less`** will give you more information about manipulating **man** via the **less** filter. **less** gives you one page at a time and lets you enter commands at the prompt to control what it does after each page. For example, you page forward with the **SPACEBAR** and page backward using **`b.`**

Once in the **man** environment, you can search for patterns by entering the ***/pattern*** option at the command line. The first instance of the string ***pattern*** will appear in the top line of the screen. To find additional instances of the pattern in the text, simply enter a slash (**`/`**).

To exit from **man** enter **`q`** (for quit).

---

1. Some platforms provide **man** pages for built-in commands, however in general you may find it easier to look in a reference book!

Printing man Pages

To print a **man** page, you can use the pipe feature (**|**) along with recommended print formatting and printing commands. These features are covered in Chapter 7. As a suggestion, pipe the output of the **man** command to **a2ps -m** (to convert **man** pages to PostScript format) and then pipe that output to the print command **flpr**:

% man *command* **| a2ps -m | flpr [***options***]**

This formats the output nicely in landscape, two pages to a sheet.

3.1.2 Finding the Right Command

If you don't know exactly what command you need, try the **-k** option with a keyword.

% man -k *keyword*

This displays the **man** page name, the section number in the UNIX documentation, and a short description for each **man** page whose name line contains ***keyword***. For example to find a search utility, enter:

% man -k search

The system will output several records similar to the following:

```
conflict(8) - search for alias/password conflicts /usr/local/lib/mh/conflict [-m
glookbib(1) - search bibliographic databases glookbib [ -v ] [ -istring ] [ -tn
ident(1) - identify files ident [ -q ] [ \&.\|.\|. ] ident searches for all occu
lkbib(1) - search bibliographic databases lkbib [ -v ] [ -ifields ] [ -pfilename
lsearch(l) - See if a list contains a particular element lsearch ?mode? list pat
lsearch(n) - See if a list contains a particular element lsearch ?mode? list pat
zgrep(1) - search possibly compressed files for a regular expression zgrep [ gre
..
..
```

Some UNIX systems have an additional utility, **apropos**, which can be used to locate commands by keyword lookup:

% apropos *keyword*

apropos ***keyword*** is equivalent to:

% man -k *keyword*

The **-f** ***filename*** option for **man** prints the manual entry summaries which might pertain to the given filename(s). Any leading pathname components are stripped from the filename before the filename is matched against the summaries. Here is an example using the **-f** option, followed by the output:

% man -f /etc/passwd

```
passwd (1)              - change login password and password attributes
passwd (4)              - password file
```

The summaries are gotten from the **whatis** database. You can run **whatis** ***command(s)*** to look up a given command and obtain the header line from the manual section. You can then run **man** to get more information on the command. If the line starts `name(section)...` you can do **man** ***section name*** to get the documentation for it.

3.1.3 Vendor Product Documentation

Most vendor product documentation is now available on-line, viewable with an X-terminal or workstation.

Platform:	Command:
AIX	**`info`**
IRIX	**`insight`**
OSF1	**`dxbook`**
SunOS	**`answerbook`** (only usable on Sun workstations at Fermilab)

3.2 The Internet

The Internet is a global network of networks that provides access to hundreds of thousands of computers around the world. As the reach of the network has grown, so has the number of services accessible. The main tools that allow the user to navigate through the Internet are:

telnet	to access remote hosts (see section 12.2)
ftp	to retrieve data files (see section 12.1.1)
Mail	to send mail (see Chapter 11)
WWW	to browse the World Wide Web (see section 3.2.1)
News	to scan the numerous Usenet news groups (see section 3.2.3)

There are two ways to reference an Internet host: an alphabetic name and a series of numbers. The alphabetic sequence is called the *host name* (e.g., fnsg01.fnal.gov) and the numeric one is called the *IP address* (e.g., 131.225.8.178). At Fermilab all host names end with *fnal.gov*, where this suffix is called the *domain name*. Since hosts may change their IP addresses, it is a good practice to always use the host name.

An excellent introduction to the Internet services is *The Whole INTERNET, User's Guide and Catalog*, published by O'Reilly & Associates.

3.2.1 World Wide Web (WWW)

Our primary way of delivering information to you, especially relatively static information, is via the World Wide Web (also known as WWW or "the Web"). There are a number of reasons why WWW has become the defacto standard for information delivery at Fermilab, within the HEP community, and even in the computer-literate parts of the world of business:

- WWW is very strong at pulling together information from different places, and of different formats and types (native **HTML**[1], PostScript, newsgroups, and even VAXNEWS), and making it appear as a seamless whole.
- It is relatively easy to make information available on WWW.
- Browsers (the programs you use to view WWW information) are available for every supported platform at Fermilab, making it an excellent fit to the distributed environment in which we now work.

Accessing WWW

In order to use WWW, enter (or put in your login script):

1. HTML stands for HyperText Markup Language, the standard language for documents accessed on WWW.

```
% setup www
```

WWW works best on workstations that support graphics, so if possible you should have on your desktop a workstation, X terminal, PC or Macintosh rather than a "dumb" terminal. If you have a graphics terminal, use either **Netscape** or **Mosaic**. To invoke them on an X terminal enter:

```
% mosaic [&]
```
or

```
% netscape [&]
```

The ampersand (**&**) is discussed in section 5.4.2. In order to use an X application, you must have defined your *$DISPLAY* variable correctly. See section 8.2.

If you don't have a graphics terminal, then you must use one of the line-mode programs, **www** or **lynx**. **lynx** is generally thought to be the better of the two. To invoke, enter:

```
% www
```
or

```
% lynx
```

This section is not intended as a detailed instruction on the use of any particular Web browser. Once you get any of them running (at least the graphical ones) there is more information than you will ever want available under **HELP**. In addition, from the *UNIX Resources* page under **WWW** you will find information both on using browsers and on the files that control what filetypes the browser recognizes. These files are called `.mailcap` and `mime.types`; see section 8.5 for an introduction to them.

WWW Basics

Web browsers find information based on URLs (Universal Resource Locators) which are like addresses and which take you to the top level of a Web site, often called a *home page*.

A home page usually refers to the first page of a commercial, educational, or personal Web site. Each home page can have several layers or pages that it links to, thereby creating a whole Web site. But the home page is generally the first place you would look. It is like looking at the cover of a book and its table of contents at the same time.

The native WWW addresses are of the form:

`http://address/`
or

`http://address/something.html`

The first part is the protocol, `http` in this case. (A protocol is a set of rules computers observe to exchange information. `http` stands for the HyperText Transfer Protocol; think of it as the identifier of a Web page.) Instead of `http` you might also see `ftp`, `file`, `gopher`, `mailto`, and `news`.

Next you'll see a colon and two forward slashes (except for `mailto`, where you'll see only the colon).

Next comes the computer's address in the format described in section 3.2. A computer address, or domain name, is used by computers in routing data across the many networks that make up the Net.

Finally, you often see a directory path at that computer plus a file at the end of the path. Web pages usually end in "`html`," for HyperText Markup Language, or sometimes just "`htm`". **HTML** is the simple scripting language that tells browsers how to display the various elements of a Web page such as links, body text, header text, inline graphics, and external files.

Many **HTML** files contain links to other documents. Sometimes links are text; sometimes they are images. If a link consists of text, it is underlined and may be in color. You can tell your cursor is on a link when a URL appears at the bottom of the screen. If a link consists of an image, you'll see a URL when you move your mouse pointer across it.

Several Web search "engines" can help you find information you are seeking, and they vary in the number of URLs they contain in their database, how deep they go into Web sites indexing information, what they index, and how frequently they "crawl" or "walk" the Web in surveying sites. There's a valuable Swiss site at URL `http://cuiwww.unige.ch/meta-index.html` that collects several WWW search tools.

Subject directories make it increasingly easy to find information about broad subjects. Two excellent directories are the Virtual Library at URL

`http://www.w3.org/hypertext/DataSources/bySubject/Virtual_libraries/Overview.html`

and something called Yahoo at URL `http://www.yahoo.com/`, a well-constructed directory of tens of thousands of Web pages.

Fermilab Home Page

Your browser is most likely configured to have either the *Fermilab Home Page* or the *Fermilab at Work* page as your default home page. If not, the *Fermilab Home Page* can be simply accessed as: `http://www.fnal.gov/`. You may want to change this to *Fermilab at Work*: `http://www.fnal.gov/faw/fermilab_at_work.html`. To do this, set your environment variable *WWW_HOME* as follows (environment variables are discussed in section 8.1.), depending on your shell:

C shell family:

```
% setenv WWW_HOME http://www.fnal.gov/faw/fermilab_at_work.html
```

Bourne shell family:

```
$ WWW_HOME="http://www.fnal.gov/faw/fermilab_at_work.html"; \
  export WWW_HOME
```

From *Fermilab at Work* you can find information on Fermilab's different divisions and experiments, activities, schedules, etc. Navigate to the Computing Division's pages to find information on UNIX.

3.2.2 UNIX Help on WWW

The man pages can sometimes be cryptic, unwieldy, or both. As an alternative, try keeping a WWW window open to *UNIXHelp*, accessible under the heading **The UNIX Operating System** on the *UNIX Resources* page. Here you'll find easy-to-follow instructions on the use of many UNIX features, organized into four categories: Tasks, Commands, Concepts, and Utilities.

Also from the *UNIX Resources* page you'll find *The UNIX Reference Desk* under the heading **Other UNIX Resources**. The resources in this document are divided into the following classes: General, Texinfo Pages, Applications, Programming, IBM AIX Systems, HP-UX Systems, Unix for PCs, Sun Systems, X Window System, Networking, Security, Humor.

3.2.3 Newsgroups

Usenet News (or NetNews) is a way of communicating "articles" among people world-wide. We have a server here at Fermilab which receives articles from elsewhere and posts the articles originating here. Fermilab has its own newsgroups named `fnal.`*xxx*. CERN's groups are prefixed with `cern` and SLAC's with `slac`. General information and especially important

information is posted in `fnal.announce`. NALCAL, seminar announcements, and the like are posted in `fnal.announce.seminars`. Computing Division forms are in `fnal.announce.forms`. UNIX discussion articles are posted in `fnal.comp.unix`. There are many more newsgroups, both Fermilab groups and others, that you might find of interest.

In order to make a newsreader available on UNIX, enter (or put in your login script):

```
% setup news
```

A number of readers will be made available to you. Line-mode browser commands are **nn**, **rn**, and **trn**. X-based browser commands are **xrn** and **knews**. **News** may also be read from most WWW browsers and some mail readers. The readers keep track of the newsgroups that you are interested in ("subscribed to") as well as which articles in each newsgroup you have read; all of the UNIX readers cooperatively maintain this information, so you can use different readers at different times without losing this information.

In general, information in newsgroups is volatile information, whereas information in Web pages is of longer term.

3.3 The Info Utility

Info is a facility available to the system support people to communicate with you about events regarding the Fermilab computing systems (shutdowns, for example), or other systems-related information that is newly available. The format of the **Info** command is:

```
% Info nametag
```

Note the capital I! The command without options results in a list of available items. To read an item, enter the command with the ***nametag*** of the item found on the left side of the **Info** list. If the item is more than one page, press the space bar to continue. Press **q** to quit.

3.4 Other Users: WWW Directories, finger and who

WWW Directories

From the Fermilab at Work page, directories are available to point you to information about Fermilab personnel and the high energy physics community at large. These directories typically contain general information such as email addresses, phone numbers, and office locations.

finger

finger is used to find out about other users. It searches for matching account names and first or last names, if known. Depending on the vendor implementation, it may display the name of the person associated with each account, the login name, the home directory and login shell, the contents of the file `.plan` in the person's home directory, and possibly other information such as waiting mail, and time of last login. If the person is logged in, it also may display information about his or her current sessions.

Note that each vendor has a different implementation of finger. In addition, for security reasons many sites disable the output of finger over the network. It is therefore unwise to rely on the format, content, or even the availability of finger as a tool for finding out about users or their accounts.

The format of the **finger** command is:

```
% finger [options] [name...]
```

where ***name*** can be a part of a personal name or a username. If you specify the option **`-m`**, then ***name*** is matched only to account name and not the rest of the personal name.

We encourage you to create a `.plan` file. It is just a text file in which you might include information such as your office location, phone numbers, mail station, home address, schedule, emergency numbers, and so on.

finger can often be used to look up users on a remote machine by specifying the name in the standard internet form *user@host*. This form works on VMS machines with **MultiNet** running, but in this case ***name*** must be the username; otherwise not much useful information is obtained.

As an alternative, point your favorite browser to the location `http://fnal.fnal.gov/finger/`.

who

The command **`who`** lists certain information about the users on your system.

```
% who
```

If used with the **`-q`** option, only the names of the logged in users and the number of users are displayed.

The **`who am i`** form identifies the invoking user. The command format is:

```
% who am i
```

There are a number of options which you can read about in the man pages.

Chapter 4: Shells

This chapter discusses the concept of a UNIX shell, and how to manipulate shells. It includes information on the available and recommended shells and their features. The concept of a shell as an interpretive programming language is introduced.

4.1 Introduction to Shells

The kernel is the real operating system and is loaded into memory at boot time. Typically the user never interacts directly with the kernel. The utilities are programs stored on disk, and loaded into memory by the kernel when invoked.

A shell is a utility. It is run in user mode, and does not have system privileges. You have a default shell, and you can invoke other shells. Invoking shells is discussed in section 4.1.2.

The shell is the interface between the operating system and the user. It interprets the commands you type and the keys you press in order to direct the operating system to take an action. Shell scripts, which are analogous to DCL command files, allow you to use the shell as an interpretive programming language. They are introduced in section 4.4, but a comprehensive treatment of scripts is beyond the scope of this manual.

There are two families of shells: one based on the Bourne shell (this family also includes the Korn (**ksh**) and Bourne-again (**bash**) shells), and the other based on the Berkeley/C shell. The shells themselves will be discussed and compared in section 4.2.

4.1.1 Determining Your Current Shell

There are several commands available in all the shells that furnish this information. We present four examples below with sample output for **csh**. The first three, **`echo`**, **`env`** and **`finger`**, will show only your login shell. If you have invoked another shell, these commands will not reflect the new shell. **`ps`** lists information about all your active processes.

`% echo $SHELL`

displays the value of the variable name that follows the **`$`**; example output: `/bin/csh`

`% env or printenv`

shows all defined environment variables, including SHELL; example output: `SHELL=/bin/csh`

`% finger` ***your_username***

shows user information and login shell; example output:

```
Login name: username                    In real life: your name
Directory: /afs/fnal.gov/files/home/room3/username Shell: /bin/csh
```

% ps

shows processes, including shell; example output:

```
PID TTY      TIME COMD
6264 pts/11   0:03 csh
```

4.1.2 Starting a Shell

A shell is started by a login process. A new shell is also started for each invocation of a terminal window or shell script (see section 4.4). Which shell gets invoked is determined by the last field in your entry in the password entry file. In a standard UNIX file system you can display your password entry by the command:

% grep ^*username* /etc/passwd

(**grep** is described in section 6.4.2.; the use of **^** is explained in section 5.3.5.) To display your password entry in an NIS environment, use the command:

% ypmatch *username* passwd

(The NIS command **ypmatch** is not described in this manual.) Sample **ypmatch** output from the FNALU system, for which the default shell has been set to **csh**, looks like:

```
aheavey:!:6302:1525:Name of User:/afs/fnal.gov/files/home/room3/aheavey:/bin/csh
```

When you log in, the login process invokes a shell program (e.g., `/usr/local/bin/tcsh` or `/usr/local/bin/bash`) and transfers control to it. The shell displays a prompt indicating it is ready for your input. The default UNIX prompts are symbols that indicate which shell is invoked (recall from section 2.3 that your prompt is likely to be set differently):

- **%** for the C shell family
- **$** for the Bourne or Korn shells

On FNALU the prompts are set to indicate the host machine, for example **<fsui01>**, or **<fsgi02>**. At any point in your session you can invoke another copy of the same shell or a different shell by typing the shell name at the prompt, for example:

% csh

invokes **csh** (C shell). This new shell, or "subshell", sits on top of your current shell. The execution of the original shell is then suspended (the shell is put to sleep), and the new shell takes control. Upon quitting the new one, the original shell wakes up and resumes control.

The average user at Fermilab does not have the privilege to change the password entry file. Therefore, to change your default shell you will need to ask your system manager.

4.1.3 Exiting a Shell

To exit a shell and return to the calling shell, type **exit** at the prompt. If you repeat the **exit** command once more, your terminal emulation is closed, and the terminal window disappears. Instead of **exit** you may need to enter **<CTRL-D>**.

4.2 Features of Available Shells

This section is excerpted from *Shell Choice, A shell comparison* (dated September 28, 1994) by Arnaud Taddei of CERN. His eleven-page document contains a brief description of the six major shells and provides an excellent comparison of features between the shells. It is available in the Computing Division library (document number TUN020) and on the Web at `http://consult.cern.ch/writeup/shellchoice`.

Of the six major shells, four are in the Bourne family: **sh**, **ksh**, **bash**, and **zsh**; and two are in the Berkeley/C family: **csh**, **tsch**.

The most up-to-date shells are **tcsh** (Berkeley/C), and **bash** and **zsh** (Bourne). These are also the three shells that are public domain (as opposed to vendor-supported). The public domain shells are the same on all platforms, which is not true of vendor shells. This is desirable when attempting to homogenize user environments. **Note that zsh is not supported at Fermilab.**

Some of the common features of these newer shells are:

- specific startup files
- startup files are the same for any platform
- specific shell variables
- specific built-in commands

The **tcsh** is essentially an enhanced **csh**. Some additional features of the **tcsh** are:

- enhanced completion[1] mechanism (programmable for commands, file names, variable names, user names, etc.)
- multiline editing capabilities (command line editing using **emacs** or **vi**-style key bindings)
- enhanced file expression syntax
- spelling correction (see section 5.2.2)
- enhanced prompt
- step up/down through history list

The following table should give you an idea of the virtues of each of the shells supported at Fermilab. It is adapted from one in Taddei's document referenced above. More complete feature lists for all the shells can be found there.

++	good
+	existing
-	weak
--	absent

Criteria	sh	ksh	bash	csh	tcsh
Configurability	-	+	++	+	++
Execution of commands	+	+	+	+	++
Completion	--	+	++	+	++
Line editing	-	+	++	-	++

1. This feature allows you to uniquely specify a file without typing in its whole name.

Criteria	sh	ksh	bash	csh	tcsh
Name substitution	+	+	++	+	++
History	--	+	++	+	++
Redirections and pipes	+	+	+	+	+
Spelling correction	--	--	--	--	+
Prompt settings	+	+	+	+	++
Job control	--	+	+	+	+
Execution control	+	+	+	+	+
Signal handling	+	+	+	-	-

4.3 Supported/Recommended Shells at Fermilab

On many systems at Fermilab, **tcsh** is used as the default shell. The Computing Division currently supports **csh**, **tcsh**, **sh**, **bash** and **ksh**. **tcsh** or **bash** is recommended for interactive use, and **sh** for scripts. (The C shell family is not recommended for scripts due to inconsistent syntax at different levels of nesting.) **zsh** is not currently supported. From the *UNIX Resources* WWW page you can access UNIX support information under **UNIX Products** to find out about any support policy changes.

4.4 Shell Scripts

As mentioned above, a UNIX shell can be used as an interpretive programming language. Besides executing shell commands within the script, you can:

- create and use variables
- process (read) arguments
- test, branch, and loop
- perform I/O

A *shell script* is a file containing a sequence of commands which can be executed by the shell, and flow control commands. The same syntax is used for commands within scripts as for interactive command entry. Section 5.1 explains briefly how the system runs and interprets shell scripts.

Although you can write complex programs using the shell language, you can also create simple shell scripts for running long commands or a series of commands that you use frequently.

In every shell script you write, include the special characters `#!` followed by the pathname of the shell as the first characters in the file. This indicates (a) that this is a script rather than a compiled executable, and (b) which shell to invoke to run the script. For example:

```
#!/usr/local/bin/bash
```

at the start of the script invokes **bash** to run it. A `#` found anywhere else in the script is interpreted as the beginning of a comment, and the shell ignores all characters between the `#` symbol and the next newline character.

A good introductory reference for script-writing with examples can be found in *UNIXHelp* on WWW. You can get to it via the *UNIX Resources* page under **The UNIX Operating System**.

Note that in order to execute the script, regardless of shell, the script file must have execute permission for the appropriate users (see section 6.6.1 for a discussion of permissions). After you set this permission, the shell will need to rebuild its "hash table" to include the new script. The hash table is a table of executables that the shell recognizes.

To complete these two operations, enter:

% chmod a+x ***filename***

% rehash[1]

To run a script, the shell must be able to locate it. Typing the full path of the filename is perhaps the most cumbersome way of telling the shell where the script is, but it will work. You may want to maintain a `$HOME/bin` directory for all your programs and shell scripts, and include this directory in your path[2]. We refer you to section 8.2 which discusses the *PATH* variable. The shell uses this variable to locate commands and other executables. You generally execute a script by typing the filename by itself, or the filename preceded by **./** on the command line, as shown below (The **./** explicitly tells the shell to look for the executable file in the current working directory.):

% ***filename*** or

% ***./filename*** or

% ***/full_path/.../filename***

The shell then interprets and executes the commands in the file one by one.

It is important to remember that, like all UNIX commands that are not part of the shell (see section 5.1.1 for an explanation of shell commands), the script file executes in a subshell forked[3] by the parent shell. This subshell retains any environment variables defined in the script as well as any shell variables defined in the file `.cshrc` or `.shrc` (one of these two files may be executed automatically prior to the script, depending on your shell; see section 8.4). At the end of the script, control returns to the parent shell, and any definitions made by the subprocess are *not* passed back to the parent process.

To execute a script for which you do want to pass back changes to the parent shell (for example, setting new shell variables), the syntax for execution differs. For the C shell family, execute the script by typing:

% source ***filename***

For the Bourne shell family, type:

$. ***filename***

For instance, after you make changes to your `.cshrc` or `.login` file, you can use **source** or **.** to execute it from within the login shell in order to put the changes into effect.

---

1. The command in **sh** is **hash**; not necessary in other shells.
2. Under FUE, the Fermi files add your `/bin` directory to your PATH.
3. Under UNIX, the term *fork* means create a new process.

4.5 Other Interpretive Programming Languages

We have mentioned that each UNIX shell can be used as the interpreter for its own programming language. Other interpretive languages supported at Fermilab are **perl** (provided in the FUE **shells** product), and **gawk** (a version of **awk**; see section 5.3.4). These languages are beyond the scope of this manual. The O'Reilly & Associates, Inc. publishers provide excellent reference texts on them.

Chapter 5: UNIX Commands and Concepts

This chapter introduces you to the UNIX command structure, and to many important commands and concepts. The features introduced in this chapter constitute the core of the UNIX operating system, and many of these tools are quite powerful and flexible. Some of the features are shell-specific, and we provide the distinctions where necessary.

5.1 Processing Environment

5.1.1 Programs, Commands and Processes

A *program* is an executable file. A program is invoked by entering its filename (which is the *command* associated with the executable), often followed by options, arguments, and/or parameters on the command line. The shell allows three types of commands:

- an executable file that contains object code produced by a compilation of source code
- an executable file that contains a sequence of shell command lines (a *shell script*)
- an internal shell command (*built-in* command)

The first two command types may include standard UNIX utilities, commercial products, and user-written programs. All the shells allow both *interactive* command entry in which the commands are typed at the keyboard and executed one by one, and *scripted* entry in which commands are put in a file, called a shell script, and executed sequentially when the script is run. See section 4.4 for a brief discussion of the uses of shell scripts and how to execute them.

Shells execute commands by means of *processes*. A process is an instance of a program in execution. A process can interact with the kernel by invoking a well defined set of *system calls*. The system calls instruct the kernel to perform particular operations for the calling program and they can exchange data between the kernel and the process. For example, a process can use system calls to create new processes and terminate running processes.

When a terminal session begins, the operating system starts a single *parent process*. Creating a new process from an existing process is called *forking*. This new process is called a *child process* or *subprocess*. Each process has a unique process identification number (PID). A subprocess can fork another process and become a parent. A process which is not receiving input from the terminal, either running or stopped, is said to be in the *background* (see section 5.4). The **`ps`** command can be used to print the status of active processes. See the man pages for information about its options.

When you give the shell a command associated with a compiled executable or shell script, the shell creates, or *forks*, a new process called a *subshell*. The new process runs the system call *exec* which invokes yet another program to execute the command in place of the current process (the subshell). Unless the subprocess runs in the background, the parent process remains dormant until its subprocess completes or is stopped. Then control returns to the parent.

To execute most built-in commands, the shell forks a subshell which executes the command directly (no *exec* system call). For the built-in commands **`cd`**, **`set`**, **`alias`** and **`source`**, the current shell executes the command; no subshell is forked. You can, however, cause the shell to

fork a process by enclosing the command in parentheses. The following example illustrates this (use of the semicolon is described in section 2.5; and the commands **cd** (change directory)and **pwd** (print working directory) are described in section 6.5):

% cd /dir1; pwd	displays **/dir1** (no subshell is forked)
% (cd /dir2; pwd)	due to the parentheses, a subshell is forked, then the commands are issued; displays **/dir2**. Control then returns to the parent process.
% pwd	displays **/dir1** since the current process was unaffected by the previous command line.

Most built-in commands exist in all shells, but there may be differences regarding arguments, options, or output format between the shell-specific versions of each command. Some commands for a given shell are not available on all platforms. Refer to a UNIX text for lists of built-in commands.

You do not need to distinguish between built-in and other commands to execute them. However in order to find help in the man pages, you do need to know which is which. Help on shell commands is usually found under the shell name, for example under **man tcsh** or **man bash**. Some platforms provide man pages for built-in commands, however in general you may find it easier to look in a reference book! Help on other commands is found directly under **man** ***command***.

5.1.2 Command Interpretation by the Shell

When the shell receives a command, it interprets it in a series of three (for Bourne shell family) or four (for C shell family) passes. Naturally, if the command is an alias (see section 8.3), it requires an additional pass up front for substitution.

- The first pass for the C shell family looks for the **!** character, and replaces it with the previous command (see section 5.2.2 for information on command recall).
- The next pass (the first pass for the Bourne shell family) replaces wildcards (used in filename expansion, redirection, and regular expressions; see sections 6.2.2, 5.3.2, and 5.3.5, respectively).
- The next pass looks for the **$** character in order to replace variable names with their values (see section 8.1).
- The final pass splits the command line elements by whitespace to arrive at the final, literal command that the shell must execute.

There are ways to prevent interpretation of special characters in each of these passes. Preceding a character with a backslash (****) works for all special characters; wildcards can be enclosed in single or double quotes; variables can be enclosed in single quotes; and whitespace is ignored if the argument containing the whitespace is enclosed in single or double quotes.

To illustrate the operations that take place in each pass, the following table presents a series of three examples using the **echo** command and the same string, first in single quotes, then double quotes, and finally with no quotes. The **echo** command writes the string to standard output. Assume that the files that match **q*** are `qq` and `qqq`, and the value of the variable *a* is **foo**.

Command -->	`echo 'q* $a x'`	`echo "q* $a x"`	`echo q* $a x`
After first pass[a], only wildcards are interpreted.	`echo 'q* $a x'` (no wildcard expansion due to quotes)	`echo "q* $a x"` (no wildcard expansion due to quotes)	`echo qq qqq $a x` (unquoted wildcard is expanded)
After second pass, unquoted or double quoted variables are replaced by their values.	`echo 'q* $a x'` (no variable replacement due to quotes)	`echo "q* foo x"` (double-quoted variable `$a` replaced by value)	`echo qq qqq foo x` (unquoted variable `$a` replaced by value)
After final pass, command string is broken up according to whitespace. The separate elements are listed vertically.	`echo` `'q* $a x'` (string treated as one argument due to quotes)	`echo` `q* foo x` (string treated as one aregument due to double quotes)	`echo` `qq` `qqq` `foo` `x` (no quotes; each argument treated separately)
When you type in the original command, the system returns the string:	`q* $a x`	`q* foo x`	`qq qqq foo x`

a. This would be the *second* pass for C shell family; there were no `!` characters to replace.

5.2 Using Commands

5.2.1 Command Entry

A UNIX command is either a built-in command or the name of an executable file which the operating system will load and execute. When you see the prompt, you can enter a command by typing the command name, any options and arguments, followed by a carriage return.

Recall, the formats displayed in this manual use **`this font style`** to indicate characters to be typed as is, and ***this font style*** to indicate arguments to be substituted. Arguments enclosed in square brackets, **`[...]`**, are optional.

You should be aware that UNIX commands are not noted for their consistency of format. Furthermore, commands, formats, arguments, and options may vary slightly from one UNIX platform to another. In this manual, we attempt to be as generic as possible, and describe options that are widely available.

UNIX commands are described on-line in the man pages (see section 3.1).

Command Format

The basic format of UNIX commands is:

`%` ***command –option(s) argument(s)***

where:

%	is the (default, non-FUE) **csh** prompt.[1]
command	is the UNIX command name of a utility or tool.
option(s)	modifies how the command runs; options are nearly always preceded by a dash and listed one after another. See example below.
argument(s)	specifies data or entities (usually files) on which the command is to operate; arguments are separated by blanks ("white space").

Remember, UNIX is case-sensitive. Therefore UNIX commands must be entered in the correct case. Most of the time commands are entered in lower case.

The components are separated by at least one blank space. If an argument contains a blank, enclose the argument in double quote marks. Normally, options can be grouped; e.g., the **-lw** and the **-l -w** option specifications are equivalent in the examples below (**wc** is a sample command; it lists line, word, and/or character count of one or more files.):

% wc -lw ***file1 file2***

% wc -l -w ***file1 file2***

Some options can have arguments, and there isn't consistency on whether there should be a blank space between the option and its argument. Check the man pages when you're not sure. In the next example which shows the FORTRAN command, ***outputfile*** is the argument of the option **-o**:

% f77 -o ***outputfile program*****.f**

Looping and conditional commands are also supported. These are more advanced shell commands and are not covered in this manual. Consult a UNIX text for information on these.

Miscellaneous command line features

- To correct typos you can use the erase key (**DELETE** or **BACKSPACE**) to erase character-by-character, or the **KILL** key to kill an entire line (See Section 2.4).
- More than one command can be entered on a line if the commands are separated by semicolons. The commands will be executed sequentially. See section 2.5 for more information on using multiple commands on one line.
- If you need to continue a command to a new line, you can either keep on typing (without doing a carriage return), or enter a backslash (****) followed directly by a carriage return (no space in-between) and then continue typing on the next line. (Recall the backslash is used to prevent a special character's meaning to be interpreted by the shell. See section 2.5.)
- You can use parentheses to group commands. Since a subshell is created for each group, this can be used to prevent changing the current environment. It can also be used to redirect all output from the commands considered as a group (see section 5.3.2).
- Type ahead works, even if the characters get interspersed with output.

5.2.2 Command Recall

Command recall is quite different in each shell. One common feature for all shells that support command recall is the **history** mechanism. It maintains a list of commands that have been entered and allows them to be reexecuted. The *history* variable, set to some number at login time in the

1. **$** is the non-FUE default for Bourne shell.

startup files, determines the number of commands that are saved in the list. The *savehist* variable specifies how many commands are to be saved for your next session after you log out. The **history** command displays the list of saved commands:

```
% history
```

We discuss the following shells separately: **csh**, **tcsh**, and **bash/ksh**. There is no command recall facility for **sh**.

csh

There is no command line editing native to **csh**. Before describing the standard **csh** command recall facility, we should mention a Fermilab product called **cedit** that we recommend for use with **csh** instead. It was designed to mimic VMS line editing, and turns out to provide similar command recall and editing functionality to **tcsh**. To use **cedit**, you need to set it up initially. Enter:

```
% setup cedit
```

To execute it, type:

```
% m
```

followed by **<RETURN>**. **m** stands for *modify*. Use the up or down arrow keys to scroll to the desired command. The right and left arrow keys and your backspace key allow you to edit the command before reexecuting it. There are several control characters that perform functions within **cedit**. Typing **<CTRL-I>** in **cedit** displays the available commands.

Recalling history commands using standard **csh** syntax is fairly easy. Use the commands listed below.

!!	Reexecute the previous command
!*n*	Reexecute command ***n*** from the history list
!*text*	Reexecute the most recent command beginning with ***text***
!?*text*?	Reexecute the most recent command containing ***text***

For example, to reexecute the 4th command from the history list, enter:

```
% !4
```

and to reexecute the last command starting with **ls**:

```
% !ls
```

The dollar sign (**$**) can be used to recall the last word of a command. **!$** causes substitution of the last word of the last command. For example, you can check the contents of `myfile.f` and then compile it using the following sequence of commands:

```
% less myfile.f
```

```
% f77 !$
```

A couple of nice features you can use with these reexecution commands are *preview* (**p**) and *substitute* (**s**). To substitute a string in the previous command and preview it before execution, use the syntax:

```
% !:p:s/oldstring/newstring
```

To do the same for the ***n***th command in the history list, use:

% !*n*:p:s/*oldstring*/*newstring*

To execute after previewing (and/or substituting), simply type:

% !!

tcsh

Recalling commands is easy if you are using **tcsh**. The up/down arrows on the keyboard can be used to recall commands and the left/right arrows can be used to move around within the command to edit it (VMS users will be familiar with this concept).

A command line correction algorithm is available in **tcsh**. To enable it, enter:

% set correct=all

This causes all words on the command line to be checked. If any part gets corrected, the system notifies you, and gives you a chance to accept or reject it. For example, say you type in:

% lz /usr/bin

The system will return with:

```
CORRECT ls /usr/bin (y|n|e|a|)?
```

Where `y`=yes, `n`=no, `e`=edit, and `a`=abort. You must provide one of these responses.

To turn off command line correction, enter:

% set correct=none

ksh

Two styles of command recall are supported; **emacs** and **vi**. The style is determined in one of two ways:

- include the line **`set -o`** ***editor*** in either your `.profile` or `.shrc` file, where ***editor*** is either **emacs** or **vi** (this takes precedence if variables below are set differently)
- set either the *EDITOR* or *VISUAL* environment variable to one of these editors

When set to **emacs**, use the usual **emacs** commands to display and modify previous commands, for instance **<CTRL-P>** for previous line. When set to **vi**, command recall is initiated by typing the **ESCAPE** (or **<CTRL-[>**) key. Then all the standard **vi** commands can be used. Some of the basic **vi** and **emacs** commands are listed in section 10.3.

bash

Both **csh** and **ksh**-style recall are supported.

5.3 Important Concepts

This section attempts to provide an overview of a few of the important concepts in UNIX which are very different from other systems and may therefore be confusing to the novice user. In order to be able to make effective use of UNIX, these concepts need to be understood.

5.3.1 Path

When you issue a command, the shell program parses the command line and either processes it directly or searches for an executable file with that name in any of the directories specified in your *search path*, which is controlled by the variable *PATH*. If the file is not found in any of the directories in your search path, the shell reports that the command was not found. The file may well be on the disk somewhere, but it is **not in your path**.[1]

FUE attempts to provide an appropriate path, and we recommend that you not change this basic path. However, feel free to *add* directories to it. For **csh**, your `.login` file contains a `set path` line. Uncomment this line (remove the `#`) and include additional directories in the shown format:

```
#set path=($path /dir1 /dir2... )
```

For **sh**, uncomment and add directories to the `PATH` line in your `.profile` file:

```
# PATH=$PATH:/dir1:/dir2...
```

See section 8.2 for information on the *PATH* variable.

As an aside, if you add a command to one of the directories in your search path, it may be necessary for you to either log out and log back in, or to recreate the internal tables used by the shell with the **`rehash`** (**csh**) or **`hash`** (**sh**) command (see section 4.4).

5.3.2 Standard Input and Output Redirection

The shell and many UNIX commands take their input from *standard input* (`stdin`), write output to *standard output* (`stdout`), and write error output to *standard error* (`stderr`). By default, standard input is connected to the terminal keyboard and standard output and error to the terminal screen. VMS users would know these as the logical devices SYS$INPUT, SYS$OUTPUT, and SYS$ERROR.

The way of indicating an end-of-file on the default standard input, a terminal, is usually **<CTRL-D>**.

Redirection of I/O, for example to a file, is accomplished by specifying the destination on the command line using a *redirection metacharacter* followed by the desired destination.

C Shell Family

Some of the forms of redirection for the C shell family are:

Character	Action
>	Redirect standard output
>&	Redirect standard output and standard error
<	Redirect standard input
>!	Redirect standard output; overwrite file if it exists
>&!	Redirect standard output and standard error; overwrite file if it exists
|	Redirect standard output to another command (pipe)

1. This concept will be familiar to users of MS-DOS.

Character	Action
>>	Append standard output
>>&	Append standard output and standard error

The form of a command with standard input and output redirection is:

% *command* −[*options*] [*arguments*] < *input file* > *output file*

If you are using **csh** and do not have the *noclobber* variable set (see section 8.2), using **>** and **>&** to redirect output will overwrite any existing file of that name. Setting *noclobber* prevents this. Using **>!** and **>&!** always forces the file to be overwritten. Use **>>** and **>>&** to append output to existing files.

Redirection may fail under some circumstances: 1) if you have the variable *noclobber* set and you attempt to redirect output to an existing file without forcing an overwrite, 2) if you redirect output to a file you don't have write access to, and 3) if you redirect output to a directory.

Examples:

% who > names	Redirect standard output to a file named `names`
% (pwd; ls −l) > out	Redirect output of both commands to a file named `out`
% pwd; ls −l > out	Redirect output of **`ls`** command only to a file named `out`

Input redirection can be useful, for example, if you have written a FORTRAN program which expects input from the terminal but you want it to read from a file. In the following example, `myprog`, which was written to read standard input and write standard output, is redirected to read `myin` and write `myout`:

```
% myprog < myin > myout
```

You can suppress redirected output and/or errors by sending it to the *null device*, **`/dev/null`**. The example shows redirection of both output and errors:

```
% who >& /dev/null
```

To redirect standard error and output to different files, you can use grouping:

```
% (cat myfile > myout) >& myerror
```

Bourne Shell Family

The Bourne shell uses a different format for redirection which includes numbers. The numbers refer to the file descriptor numbers (0 standard input, 1 standard output, 2 standard error). For example, **2>** redirects file descriptor 2, or standard error. **&*n*** is the syntax for redirecting to a specific open file. For example **2>&1** redirects 2 (standard error) to 1 (standard output); if 1 has been redirected to a file, 2 goes there too. Other file descriptor numbers are assigned sequentially to other open files, or can be explicitly referenced in the shell scripts. Some of the forms of redirection for the Bourne shell family are:

Character	Action
>	Redirect standard output
2>	Redirect standard error
2>&1	Redirect standard error to standard output
<	Redirect standard input
|	Pipe standard output to another command
>>	Append to standard output
2>&1|	Pipe standard output and standard error to another command

Note that **<** and **>** assume standard input and output, respectively, as the default, so the numbers 0 and 1 can be left off. The form of a command with standard input and output redirection is:

$ ***command*** **-[***options***] [***arguments***] <** ***input file*** **>** ***output file***

Redirection may fail under some circumstances: 1) if you have the variable *noclobber* set and you attempt to redirect output to an existing file without forcing an overwrite, 2) if you redirect output to a file you don't have write access to, and 3) if you redirect output to a directory.

Examples:

$ who > names	Direct standard output to a file named `names`
$ (pwd; ls -l) > out	Direct output of both commands to a file named `out`
$ pwd; ls -l > out	Direct output of **ls** command only to a file named `out`

Input redirection can be useful if you have written a FORTRAN program which expects input from the terminal and you want to provide it from a file. In the following example, `myprog`, which was written to read standard input and write standard output, is redirected to read `myin` and write `myout`.

```
$ myprog < myin > myout
```

You can suppress redirected output and/or error by sending it to the *null device*, **/dev/null**. The example shows redirection of standard error only:

```
$ who 2> /dev/null
```

To redirect standard error and output to different files (note that grouping is not necessary in Bourne shell):

```
$ cat myfile > myout 2> myerror
```

5.3.3 Pipes

UNIX uses the concept of a *pipe* to connect the standard output of one program directly into the standard input of another program. This is specified by separating the two commands with the pipe operator, the vertical bar (**|**). The general format is:

% ***command1*** **|** ***command2*** **| . . .**

where, of course, each command can have options and arguments. To implement pipes of commands, the shell forks off multiple processes. For example if you run the command:

```
history | more
```

the shell forks twice; the grandchild runs **history**, the child runs **more** (after hooking up the right file descriptors to the right pipe ends), and the parent shell waits for the process to finish. The **history** command, a built-in, is implemented in the grandchild shell process directly, while the **more** command requires an *exec* system call.

The **tee** command can be used to send output to a file as well as to another command.

```
% who | tee whoout | sort
```

This creates a file named `whoout` which contains the original **who** output. It also sorts the **who** output and sends it to standard output, the terminal screen. The following example sends the (unsorted) **who** output to the file and the screen:

```
% who | tee whoout
```

5.3.4 Filters

A *filter* is a command or program which gets its input from standard input, sends its output to standard output, and may be used anywhere in a pipeline. Examples of filters are the UNIX utilities:

- **more** (and **less**)
- **grep**
- **awk**
- **sort**

The combination of UNIX filters **grep**, **awk**, and **sort** and the use of pipes is very powerful.

more and less

The **more** filter allows you to display output on a terminal one screen at a time. You press **SPACEBAR** to move to the following screen, and **q** to quit.

less is a much more flexible variant of the standard UNIX utility **more** and is provided under FUE[1]. The command **less** lists the output (e.g., specified files) on the terminal screen by screen like the command **more**, but in addition allows backward movement in the file (press **b** to go back one full screen) as well as forward movement. You can also move a set number of lines instead of a whole page. To view a file with the **less** filter, enter:

% less [*options*] [*filename*]...

The options and usage are described in the man page for **more** and **less**.

1. FUE sets your environment variable *PAGER* to the **less** filter.

After displaying a page of information, **more** and **less** display a colon prompt (**:**) at the bottom of the screen and wait for instructions.

```
     LESS(1)                  UNIX System V                  LESS(1)



     NAME
          less - opposite of more

     SYNOPSIS
          less [-[+]aABcCdeEimMnqQuUsw] [-bN] [-hN] [-xN] [-[z]N]
               [-P[mM=]string] [-[lL]logfile] [+cmd]
               [-ttag] [filename]...

     DESCRIPTION
          Less is a program similar to more (1), but which allows
          backwards movement in the file as well as forward movement.
          Also, less does not have to read the entire input file
          before starting, so with large input files it starts up
          faster than text editors like vi (1).  Less uses termcap (or
          terminfo on some systems), so it can run on a variety of
          terminals.  There is even limited support for hardcopy
:
```

You can search for patterns in the file by entering ***/pattern*** at the **less** prompt. Continue to search for the same pattern using a slash (**/**). A further advantage is that **less** does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like **vi**.

grep

The **grep** filter searches the contents of one or more files for a pattern and displays only those lines matching that pattern. **grep** is described in Section 6.4.2.

awk

awk is much more than a filter; it is a powerful pattern scanning and processing language. Although you will need to spend a little time learning how to use **awk**, it is very well suited to data-manipulation tasks. It handles internally what you would have to handle laboriously in a language like C or FORTRAN. You can do in a few lines what would take many, many lines of FORTRAN.

awk works best when the data it operates on has some structure, for example a document with heading levels, or a table. In the case of a table, you can tell it the field separator (spaces, colons, commas, tabs) and it can align and interpret the contents of the field according to the way you use it. Or you can reorder the columns, or change rows into columns and vice-versa.

We present here some very basic information to get you acquainted with the concepts of **awk**, but you will need a more in-depth reference in order to use this utility. A widely-available book on **awk** is *The awk Programming Language* by Aho, Kernighan, and Weinberger, Addison-Wesley. Another good reference, from which much of the information in the present section is extracted, is *sed & awk* published by O'Reilly & Associates.

There are several versions of **awk**, and they differ from platform to platform. "Old" **awk** may be **awk** or **oawk**, "new" **awk** may be **nawk**. FUE provides a GNU version of **awk** called **gawk** as part of the **shells** product.

Some of the features of **awk** are:

- Ability to view a text file as made up of records and fields in a textual database
- Use of variables to manipulate the database
- Use of arithmetic and string operators
- Use of common programming constructs such as loops and conditionals
- Ability to generate formatted reports

With **nawk**, additional features make it easier to write larger scripts. Using **nawk** you can:

- Define functions
- Execute UNIX commands from a script
- Process the result of UNIX commands
- Process command-line arguments more gracefully
- Work more easily with multiple input streams

awk executes a set of instructions for each line of input. You can specify instructions on the command line or create a script file. Input is read a line at a time from one or more files or from standard input. The ***instructions*** must be enclosed in single quotes to protect them from the shell. (Instructions always contain curly braces which are interpreted as special characters by the shell.) We refer you to one of the books on **awk** for the available instructions. We'll use the instruction **`print`** in our examples.

For command lines, the syntax is:

`% awk '`*instructions*`'` ***files***

As an example, say that file `test` contains only the line `Hello, world`. The command:

```
% awk '{ print }' test
```

produces the output:

```
Hello, world
```

Multiple command lines can be entered by separating commands with semicolons or using the multi-line input capability of the Bourne shell. **awk** programs are usually placed in a file where they can be tested and modified. The syntax for invoking **awk** with a script file is:

`% awk -f '`*script*`'` ***files***

where **`-f`** indicates that the filename of a script follows.

awk interprets each line of the input data file(s) as a record, and each word on that line, delimited by blank spaces or tabs, as a field. You can reference these fields, either in patterns or procedures. **`$0`** represents the entire input line; **`$1`**, **`$2`**, ... refer to the position of individual fields on the input line. As an example, say that the file `personnel` contains a list of employees' first names, last names, and addresses. The command:

```
% awk '{ print $1}' personnel
```

would produce output of the type:

```
John
Alice
Mary
Eric
```

To use the pattern-matching features of **awk**, you need to be familiar with the metacharacters used in regular expressions (see section 5.3.5). A pattern is enclosed between forward slashes (**/**) on the command line or in a script. When **awk** reads an input line, it attempts to match each pattern-matching rule in a script. Only the lines matching the particular pattern are the object of an action. If no action is specified, the line that matches the pattern is printed (executing the **print** statement is the default action).

In our `personnel` example, let's assume that Alice is from Illinois (IL) and Eric is from Iowa (IA). To bring up the complete records with the pattern IL or IA, we could issue the command:

```
% awk '/I./{print}' personnel
```

where the metacharacter **.** matches any single character. We could more simply type:

```
% awk '/I./' personnel
```

and get the same result in either case:

```
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Eric Smith 24 Birch St., Albert City, IA 50510
```

In Appendix C we present **awk**'s programming model, which is beyond the scope of the present section. This model will help you understand the potential that **awk** offers the programmer.

sort

sort sorts the lines of the specified files, typically in alphabetical order. Using the **-m** option it can merge sorted input files. Its syntax is:

% sort [*options*] [*field-specifier*] [*filename(s)*]

For example, start with the `personnel` file contents:

```
John Smith 75 South Ave., Denver, CO 80145
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Mary Fahey 901 California St., San Francisco, CA 94121
Eric Smith 24 Birch St., Albert City, IA 50510
```

Run the command:

```
% sort personnel
```

to reorder the file contents as follows:

```
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Eric Smith 24 Birch St., Albert City, IA 50510
John Smith 75 South Ave., Denver, CO 80145
Mary Fahey 901 California St., San Francisco, CA 94121
```

sort is very easy to use. Read the man page for **sort** to see what the available options are and how to specify the sort fields. If a field is not specified, the sort key is the entire line. The sorted output goes to standard output by default.

5.3.5 Regular Expressions

A *regular expression* is a string composed of letters, numbers, and special symbols that defines one or more strings. They are used to specify text patterns for searching. This is similar to wildcards on VMS.

A regular expression is said to *match* any string it defines. The major capabilities include:

1) match single characters or strings of characters

2) match any arbitrary character

3) match classes of characters

4) match specified patterns only at the start or end of a line

5) match alternative patterns

Regular expressions are used by **vi**, **grep**, and **awk** (and at least a couple of utilities not covered in this manual, for instance **ed** and **sed**). **grep** in fact stands for **g**lobal **r**egular **e**xpression **p**rinter. For a complete discussion of regular expressions, refer to a UNIX text. To get you started, we include a table of special characters that can be used in expressions.

Note that regular expression special characters are different from those used in filename expansion.

`.`	Matches any single character Example: *.ing* matches all strings with any character preceding *ing*; sin*g*ing, *p*ing
`*`	Represents 0 or more occurrences of the preceding character Example: *ab*c* matches *a* followed by 0 or more *b*'s followed by *c*; *ac*, *abc*, *abbbbbc*
`.*`	Matches any string of characters (`.` matches any character, `*` matches any number of occurrences of the preceding regular expression)
`$`	Placed at the end of a regular expression, matches the end of a line Example: *ay$* matches *ay* at the end of a line; ... tod*ay*
`^`	Placed at the beginning of a regular expression, matches the beginning of a line Example: ^*T* matches a *T* at the beginning of a line; *T*oday ...
`"`	Delimits operator characters to prevent interpretation
`\`	Turns off special meaning of the following single character (`\` is often called a *quote* character)
`[]`	Specifies character classes
`[...]`	Matches any one of the characters enclosed in square brackets Example: *[bB]ill* matches *bill* or *Bill*

There is an extended set of special characters available for full regular expressions, including for example `?` and `+`. These can be used in **egrep** and **awk**. Refer to a UNIX book for information.

5.4 Job Control

Any command you give to the shell (true for all shells except **sh**) is a *job* and is given a *job number.* A single command is the simplest job. A series of commands separated by semicolons, or commands piped together, create a single job. A script also creates a single job. A job may consist of many processes, because each command is a process.

The job stays with its environment, for example, the current directory. If you subsequently change directories after putting a job in the background and then resume the background job, you will be in the original directory again.

Job control allows you to work on several jobs at once, switching back and forth between them at will, and it allows you to stop, start, and kill them. When you start up a job interactively, it is by default in the *foreground* and attached to your terminal. You can move that job into the *background* so you can start up another job or observe another job that is already running. You can move any background job into the foreground so it is once again attached to your terminal. You can run any number of background jobs at any one time, but there can be only one foreground job. The use of multiple windows on an X terminal makes much of this transparent.

5.4.1 Priority

You can control the priority of a command or shell with the shell command **nice**:

% nice [+*n* | –*n*] [*command*]

n is the value by which you want to increase or decrease priority. Values range from 1 to 19, with the default at 10. The higher the **nice** value, the lower the priority of a process, and the slower it runs. (You are being *nicer* to other users!) If no number is specified, **nice** sets the priority to 4. If ***command*** is omitted, the priority is set for the current shell. If ***command*** is specified, it is run at the specified (or default) priority in a sub-shell. You can use **nice** to lower the priority of a command or shell that makes large demands on the system but isn't needed right away.

Note that another **nice** command exists, `/bin/nice`. It is not a built-in shell command. If you do **man nice**, you will get information on this one. In order to get information on the shell command **nice**, do **man csh** (using **csh** as an example).

5.4.2 Background, Foreground, and Suspended Jobs

You run jobs in the *background* so that you can perform other tasks in the foreground (i.e., interactively). Jobs are always in one of three states: running in the foreground, running in the background, or suspended. Any job intended to run in the background should have its output and error redirected to a file.

There are two ways to put jobs into the background:

Using the & Metacharacter

One way to start a job in the background is to append the ampersand metacharacter (**&**) to the end of the command line. In the first example, the standard output is redirected to a file (in this case, the syntax is valid for both shell families):

% *command* > *output-file* &

Note that the parentheses are necessary in the next example in order to send both commands to the background:

% (*command1*; *command2*) &

The shell prints a line indicating the job number and process ID of its top-level commands, and the job is started as a background job.

Using the Suspend Control Character

The other way is to use the suspend control character, called *susp* or *swtch*, (see section 2.4) which is usually assigned to **<CTRL-Z>**. It stops or *suspends* the foreground (the currently running interactive) job, moving it to the background; it does not kill it.

After stopping a job, you can either resume it with the **fg** command or make it run in the background with the **bg** command (see below). You may want to stop a job temporarily to do another task and then return to it interactively, or you may want to stop it in order to let it finish as a background job.

When a background job terminates, this is reported just before the next prompt (so the message doesn't interrupt the current foreground job).

A background job will stop if it tries to read from the terminal. If output is not redirected, a background job can either (continue to) send output to the terminal or be stopped if it attempts to write to the terminal. The following command can be used to toggle this behavior:

```
% stty [-]tostop
```

The minus indicates negation, meaning that background jobs will continue to run even if they attempt to write output to the terminal and that the output will appear on the terminal screen. However, programs which attempt to interrogate or change the mode of the terminal will be blocked when they are not in the foreground whether or not *tostop* is set.

Listing Jobs

The **jobs** command lists your jobs:

```
% jobs [-l]
```

This command lists the background jobs, their job number, status, and the command being executed. A plus sign in the output means that job is *current* (in control of your terminal), a minus sign means that job is *next*. *Current* and *next* refer to its relation to the foreground (see **fg**). The **-l** option lists the process ID as well.

Commands Used for Controlling Jobs

There are a number of commands to control jobs: **fg**, **bg**, **stop**, **kill**. All of them can take an argument which specifies the particular job, or they can have no argument. The argument can take two basic forms: a simple process ID number (as displayed by **ps**) or a form prefixed with a percent sign (**%**). If no argument is given, the current job is acted upon.

The **%** form of the argument can be **%-** where **-** indicates the previous job, **%*n*** where ***n*** is the job number as displayed by the **jobs** command, **%*pref*** where ***pref*** is some unique prefix of the command name and arguments of one of the jobs, or **%?*str*** where ***str*** is some unique string in one of the jobs.

You can use the **fg** command to move a suspended or background job into the foreground:

```
% [fg] %[job]
```

The **fg** is not mandatory. If the job specification is omitted, the current job will be brought into the foreground, and the next job becomes current.

Examples (note that the first **%** on each line represents the default **csh** prompt):

% fg %5	Bring job number 5 into the foreground
% %1	Bring job number 1 into the foreground
% %	Bring the current job into the foreground

After stopping a foreground job, you can start it running in the background with the **bg** command. **bg** puts the current or specified jobs into the background, continuing them if they were stopped. In the following commands, ***job*** stands for job number.

% bg %[*job*]

We described above how to stop (suspend) a foreground job with the suspend control character (**<CTRL-Z>**). Similarly, you can suspend a background job with the **stop** command:

% stop %*job*

You can abort a suspended or background job with the **kill** command:

% kill %*job*

If you attempt to exit a shell (logout) when there are stopped jobs, you will get a warning message. A second **logout** will log you out if you choose not to see what jobs are stopped before you exit. In the C shell family, background jobs will continue running after you log out.

5.4.3 Scheduling Jobs: at and cron

UNIX provides two methods for running jobs at some specified time.

If AFS is installed on your system, there are Kerberos authentication problems with running programs which spawn a job external to your login process group. **at** and **cron** fall into this category. A work-around is available for executing an authenticated **cron** job (send mail to *uss-group@fnal* requesting the full details of this procedure), however a more elegant solution is expected to be available sometime in 1996. See the *FNALU User's Guide*, document number GU0008, for more information.

at

The first is the **at** utililty. This allows the user to queue a job for later execution.

The format of the **at** command is:

% at *time* [*date*] [+*increment*]

at reads the commands from standard input. Standard output and standard error output will be mailed to you unless they are redirected.

The shell saves the environment variables and the working directory that are in effect at the time you submit the job and makes them available when the job is executed.

- The ***time*** can include 1, 2, or 4 numbers. One or two digits is assumed to be hours, four digits to be hours and minutes. It can be specified as two numbers separated by a colon (hours:minutes), either in 24-hour format or with *am* or *pm* appended. The names *noon*, *midnight*, *now*, and *next* are recognized.
- The ***date*** is either a month name followed by a day number (and optionally a year number followed by an optional comma) or a day of the week (fully spelled out or abbreviated to three characters). The words *today* and *tomorrow* are known. If no date is given, *today* is assumed if the hour is greater than the current hour and *tomorrow* if it is less.
- The optional ***increment*** is a number suffixed by *minutes*, *hours*, *days*, *weeks*, or *years* in singular or plural form.

Examples:

% at 8

% at 0800

% at 8:00am Jan 24

% at now + 1 minute

at reads from standard input, meaning you type in the commands (there may or may not be a prompt). When you are finished, terminate input with **<CTRL-D>** followed by a carriage return.

You can also redirect the input to a file of commands, for example:

% at now + 1 hour < myscript

at runs in the Bourne shell (**sh**) by default. If you need to force it to run in C shell, you can use the trick illustrated in the following interactive example:

% /bin/csh << xxxxx

? at now + 2 minutes

? source .cshrc

? alias > aout

? <CTRL-D> (followed by carriage return)

The first line causes **csh** (C shell) to read the following lines up to **xxxxx** or to the end-of-file. There is no **xxxxx**, of course, so it reads until you give it the **<CTRL-D>**. The third line runs your `.cshrc`. It is an illegal Bourne shell command, therefore you can tell **at** ran in the C shell and that your `.cshrc` file was executed. You will receive a message similar to the following, and the results will be mailed to you (alas, **at** will say it's using `/bin/sh` even if you've "tricked" it):

```
warning: commands will be executed using /bin/sh
job 826157640.a at Wed Mar  6 18:14:00 1996
```

After 2 minutes, `aout` is mailed to you. It contains a list of all the aliases defined in your `.cshrc` file. If you are running a script using **at** then the script will be run under whatever shell you specify in the script.

For example, say you run:

% at now + 2 minutes

? *script*

? <CTRL-D> (followed by carriage return)

where ***script*** is a file that contains the line `#!/bin/csh` at the beginning. The commands in the script will execute under **csh**.

cron

The second method for running jobs at some specified time is the **crontab** command. It is designed for jobs that need to be run on a regular basis, e.g., once a night, or once per week. Note that **cron**, like **at**, uses the Bourne shell so that output redirection must be specified using Bourne shell syntax. Scripts will be run under what ever shell is specified in the script. If no shell is specified then Bourne shell is used.

The format of the command is:

% crontab [*filename*]

% crontab [*options*]

where ***filename*** is the name of a file containing the commands that you want to have executed. If you do not specify a file, then **`crontab`** will read commands from standard input as you type them, ending with **<CTRL-D>**, and the commands will be run in Bourne shell. The system utility **cron** reads the crontab file and runs the commands. Standard output and standard error will be mailed to you unless they are redirected.

The command can also take the following options:

`-r`	remove crontab file
`-l`	list contents of crontab file

A **cron** file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

- minute (0-59)
- hour (0-23)
- day of the month (1-31)
- month of the year (1-12)
- day of the week (0-6 with 0=Sunday)

If an asterisk appears in a field instead of a number, **cron** interprets that as a wildcard for all possible values. The sixth field of a line in a **cron** file is a string that is executed by the shell at the specified times.

Examples:

The user creates a **cron** file `myfile`, and runs **`crontab`**:

```
#Myfile
# Run script that archives to 8mm tape for backup.
# Monday-Thursday at 2200 backup everything that has been
# changed.  Every Friday at 2200 backup everything.
0 22 * * 1-4 /usr/buckley/daily  1>>/usr/buckley/cron/backup.log 2>&1
0 22 * * 5   /usr/buckley/weekly 1>>/usr/buckley/cron/backup.log 2>&1
```

```
% crontab myfile
```

This command will perform an incremental backup at 10pm Monday to Thursday and a full backup at 10pm on Friday.

Chapter 6: The UNIX File System

The UNIX file system has a hierarchical or tree-like structure with the directory called *root* (**/**) as its source. The system is essentially composed of *files* and *directories*. In this chapter we describe techniques for manipulating files and directories, and commands designed to provide information about them.

In Appendix D you can find a table of UNIX equivalents for commonly used VMS commands.

6.1 Directory Structure

The UNIX system automatically puts you at a specific location in the file system when you log in. This is called your *login directory.* Typically, this is the same as your *home directory.* The name of your home directory is usually the same as your login name. Within this directory you can create files and additional directories (sometimes called *subdirectories*) in which to group the files. You can move and delete your own files and directories and control access to them.

The root of the file system is called *root* and is written as a slash (**/**). In other words, to change to the root directory, type:

```
% cd /
```

There is only one directory tree on a system even if several devices are mounted in that tree. (All devices are viewed as files.) The *current directory* or *working directory* is the directory that you are currently working in, which is also the directory that commands refer to by default. Files in your current directory can, therefore, be specified by their filenames only.

6.1.1 Pathnames

Wherever you can use a filename, you can also use a *pathname*, which is how you point to files that are not in your current directory. You can refer to files in other directories using either a *relative path name*, that is a path specified relative to your current directory, or with an *absolute path name*, that is a path specified relative to the root of the file system.

Absolute path names are preceded with **/**, the root directory. If a pathname does not begin with **/** it is assumed to be a relative path name. Relative path names begin with a directory or filename, a **.** (pronounced “dot”) which refers to the current directory, or **..** (pronounced “dot dot”) which refers to the directory immediately above the current directory. The character **/** also separates components of the pathname, which are directory names, except for the last one, which can be either a simple filename or a directory name.

In summary, every file has a pathname, and its absolute pathname is of the form:

```
/rootdir/dir2/... /filename
```

The following is the form of a relative pathname of a file:

```
dir_n/dir_n+1/... /filename
```

An example of an absolute path name is:

```
/usr/smith/project1/afile
```

If my current directory is `/usr/smith`, then I can refer to the file `afile` in subdirectory `project1` with a relative pathname like this:

```
project1/afile
```

Or, if my current directory is `/usr/smith/project1`, I can refer to a file named `fileb` in `/usr/smith/project2` as:

```
../project2/fileb
```

Note that you cannot necessarily tell if `fileb` is an ordinary file or a directory name. Many commands will accept a directory name, and if it *is* a directory name, the command in which it is used may perform the action on all files in the directory. This behavior can be dangerous!

6.1.2 The Home Directory

Your home directory is the top of your personal branch in the file system, and is usually designated by your username, for example */path/username*.

Tilde (~)

In most UNIX shells other than **sh**, the *tilde* (**~**) stands for the home directory. Used alone, it specifies your home directory. Followed by a different user's login name, it expands into the pathname of the home directory of that user. This is a convenient way to refer to a user's directory, because it is independent of where the system manager may place the directory on the disk.

The use of tilde (**~**) to refer to a home directory is limited. It isn't available in the Bourne shell, and isn't available in FORTRAN.

Of the following three examples, the first refers to the file `def` from your own home directory, the second to the home directory of user *jones*, and the third refers to file `data1` in the subdirectory `project1` of jones' home directory.

```
~/def
~jones
~jones/project1/data1
```

To change to jones' home directory you'd enter:

```
% cd ~jones
```

logdir

FUE provides the command **`logdir`** which returns the full path of the specified user. **`logdir`** by itself returns the path of the invoking user. For example:

`% logdir` ***username***

is equivalent to:

`% echo ~`*username*

To change directories you'd enter (note the use of backquotes to use the output of the enclosed string):

`% cd` `` `logdir` *username*`

In contrast to the tilde, **logdir** can be used within commands, scripts, FORTRAN and C programs, and other programs in all shells.

$HOME

The environment variable *HOME* is automatically set to the absolute pathname of your home directory . Environment variables are discussed in section 8.1 and *HOME* is described in section 8.2. To see the value of *HOME*, enter:

```
% echo $HOME
```

From some other directory, you can change to your home directory or one of its subdirectories using a command like the following:

```
% cd $HOME[1]
```
or

```
% cd $HOME/mysubdir
```

6.1.3 Command Line Directory Shortcuts

.	Current directory
..	Parent directory of the current directory (“up” one directory)
~	Your home directory (all shells but **sh**)
$HOME	Environment variable whose value is your home directory
~username	Home directory of another user (all shells but **sh**)
/	Root directory

6.1.4 Directories and Executables

It is appropriate at this point to mention the relationship between directories and commands. A command is simply the name of an executable file, located in some directory. To execute a command, the shell first needs to find the executable file. The shell therefore needs to be given a set of directories to search. This information is provided via the environment variable *PATH* which is a list of search directories. You can display it with the command:

```
% echo $PATH
```

PATH is explained more thoroughly in section 8.2. Standard UNIX commands are generally grouped in a few standard directories (e.g., `/usr/bin`), and your default *PATH* contains these. Please see section 8.2 to learn how to run executables that you create and store in your own directories.

The utility **which** is useful in cases where a command may be ambiguous, for example due to aliasing (see section 8.3), and you want to know exactly which executable file or files the command runs. **which** lists the files that would be executed if the specified command(s) had been run. The syntax for **which** is:

```
% which command [command2 ...]
```

Each argument is expanded if it is aliased, and your path is searched for the executable files associated with the commands. See the man page for more information.

---

1. **cd** by itself is equivalent to **cd $HOME**.

6.2 Files

An ordinary file contains ASCII characters or binary data and is considered by the UNIX system to be merely a sequence of bytes. No structure is imposed on the file and no meaning attached to its contents by the system; the meaning depends on the program that reads the file.

A directory file contains an entry for each file in that directory. The directory entry for a particular file contains the file name and *inode number*. The inode number is a volume data structure used by the file system. It has an associated entry in the *inode table* which contains other information about the file such as the owner, file protection, modification date.

A *hidden file* is an ordinary file whose name begins with a period (pronounced "dot"). The `.login`, `.cshrc`, and `.logout` files described in Chapter 8 are hidden files. The reason they are called hidden is that the **`ls`** (list files) command does not list them by default. Use the **`-a`** option with **`ls`** to see them. Hidden files do not appear in filename expansion of *, either. Filename expansion is discussed in section 6.2.2, below.

UNIX does not support file versions. If you edit a file and save it with the same name, your earlier version is overwritten. Similarly, if you copy or rename (move) a file to a filename that already exists, the original file is overwritten.

6.2.1 Filenames

A full file specification has only two parts, the directory specification and the file name. A filename is composed of from 1 to 14 characters in old UNIX implementations and a much larger number in more recent versions (up to 255, typically). Although you can use any character in a filename except **`/`**, UNIX assigns special meaning to many characters (metacharacters), so they should be avoided (see section 2.5). It is safe to use the upper- and lowercase letters, numbers, dash (−), underscore (_), period (`.`), and comma (`,`). As mentioned in the previous section, files beginning with a dot (`.`) are hidden files. The "filenames" **`.`** and **`..`** (single and double dot) are reserved. The **`.`** refers to the current directory, and the **`..`** refers to the current directory's parent directory. No two files in the same directory can have the same name, but files in different directories can have the same name.

VMS users have not had the luxury of using dashes in filenames. Dashes are much more common in UNIX filenames simply because it's easier to type `my-file` than `my_file`.

Filenames are case sensitive. This means `MYFILE` is different from `Myfile` is different from `myfile` is different from `myFile`, etc.

You cannot distinguish a directory file from an ordinary file by its name, although some people make their own convention by beginning directory filenames with a capital letter, or ending them in `.d`.

Filename extensions are not required in UNIX. You can include a period and an extension in a filename to help describe the contents of the file, but it will not have special meaning to UNIX itself. However, programs can make use of extensions, for example the FORTRAN compiler expects certain extensions. Note, you can have more than one period in a filename, for example, `lex.yy.c`.

6.2.2 Filename Expansion and Wildcard Characters

The UNIX shells have a number of special characters which can be used on the command line when specifying filenames and directory names. They allow the shell to expand the argument into a set of filenames. These characters are called *wildcards*. Filename references that contain these characters are called *ambiguous file references*. Filename expansion is also called *globbing*.

The question mark (**`?`**) causes the shell to generate filenames which match any single character in that position. For example, `out?` matches `out1` but not `out12`.

The asterisk (**`*`**) causes the shell to generate filenames which match any number of characters (including zero characters) in that position. For example, `myfile` matches `myf*`. The * alone means all files (except those that begin with dot (**`.`**), which is a special case).

A pair of brackets (**`[ ]`**) surrounding a list of characters causes the shell to match filenames containing the individual characters in that position. The brackets define a *character class* and each definition can only replace a single character in a filename. In other words, it is like a question mark that will only allow certain characters. For example, `memo1` and `memoa` match `memo[14a]`), but `memo3` and `memo1a` do not. A hyphen can be used to define a range of characters, for example `[a-z]` represents all lowercase characters. Thus `memo[a-z]` matches `memoa` but not `memo2` or `memoB`.

Character	Action
`?`	matches any single character in a filename
`*`	matches any string of characters (including the empty string) in a filename
`[ ]`	matches any single character from the set enclosed in the brackets

Examples:

- **`% ls out*`** lists all files beginning with `out`
- **`% ls out?`** lists all files with 4-character names beginning with `out`
- **`% ls out[ab]*`** lists all files beginning with `out` followed by `a` or `b` (e.g., `outa4`)
- **`% ls *out*`** lists all files containing `out`

Filename expansion may surprise you with the results. For example, **`ls b*`** would list all files starting with b in the current directory, but it would also list the contents of all **directories** whose names start with `b` because of the way **`ls`** behaves for a directory argument. If you want to be sure of what filename expansion will result in, you can use the **`echo`** command to check it before executing a command.[1] For example, say you have a few matching files in your directory for the command:

```
% echo *out*
```

You would obtain output something like this:

```
fout fout275 inandout out1 out2 out
```

Filename expansion in **csh** can be turned off by setting the *noglob* variable:

```
% set noglob
```

To turn it back on, type **`unset noglob`**.

6.3 Manipulating Files

This section describes the basic file manipulation commands:

1. **`echo`** is otherwise useful for sending messages to the terminal from a script and sending known data into a pipe.

- listing the contents of a directory
- displaying the contents of a file
- copying and renaming a file
- deleting a file
- changing a file's access permissions

Section 6.5 describes the commands you can use to change and manipulate directories.

6.3.1 List Directory Contents: ls

The **ls** command, which stands for **list**, is used to list the contents of a directory. **ls** has many options, some of which are system-dependent, so only a few of them are described here. For a complete description of the command, refer to the man pages for **ls.**

ls by itself lists the names of the files and subdirectories in the current directory (in multicolumn format on some platforms), sorted alphabetically.

The format is:

% ls [*options*] [*filenames*]

where some of the options are:

-a	List all entries, including those that begin with **.** (dot).
-l	List in the long format, giving mode, number of links, owner, group, size in bytes, and (by default) time of last modification, by default sorted by filename.
-C	List in columns (default on some platforms)
-F	Put a / after the name of each file that is a directory, an * after the name of each file that is executable, and an @ after the name of each file that is a symbolic link.
-R	Recursively list subdirectories encountered.
-t	Sort by time stamp (latest first) instead of by name. The default time stamp is the last modification time (see **-u**).
-u	Use the time of last access for sorting if used with the **-t** option or printed in the date column if used with the **-l** option. **-ult** both sorts by and displays last access date.
-d	If the argument is a directory, list the directory itself, not its contents. Use with **-l** to get the status (e.g., permissions) of a directory.

If the argument is a directory, **ls** displays the contents of the directory. Note that this can happen unintentionally as a result of filename expansion. This behavior can be prevented with the **-d** option. The **-t** option is useful when looking at recent files:

% ls -lt

will result in the long output sorted by reverse modification date rather than by filename.

The following is a sample output of **`ls -l`**.

```
total 251
drwxr-xr-x 3 nicholls g020c 512   May 2  08:53 Tools
drwxr-xr-x 2 nicholls g020c 512   May 2  09:01 bin
-rw-r--r-- 1 nicholls g020c 446   May 4  14:09 defaults
-rw-r--r-- 1 nicholls g020c 95418 May 1  17:42 intro.lpr
-rw-r--r-- 1 nicholls g020c 0     May 10 17:51 lsout
-rw-r--r-- 1 nicholls g020c 6683  May 1  16:46 man.lpr
-rw-r--r-- 1 nicholls g020c 12258 May 9  16:16 out
```

The first line indicates the number of blocks used. The rest of the lines report on (sub)directories or files in the directory being reported on. The first column of the output is called the *mode*. The character in this first column indicates the type of file, and for our purposes here, they are:

`d`	directory
`-`	ordinary file

The next 9 characters are interpreted as three fields of three characters each, indicating the read (**`r`**), write (**`w`**), and execute (**`x`**) permissions for *owner*, *group*, and *all*, in that order (see section 6.6.1 for a discussion of permissions).

Next is the number of links to the file or directory. This refers to the number of different names established for it. Normally files have 1, and directories have 1 each for the directory itself, its parent directory, and each of its subdirectories. In the sample output above, notice that the directory `Tools` has 3 and `bin` has 2. Evidently, `Tools` has one subdirectory and `bin` has none.

The next fields are the login name of the owner, the group to which the owner belongs, the size of the file in bytes, the date and time the file was last modified, and, finally, the filename (which can be a directory name).

6.3.2 List File Contents: cat, less, more, head, and tail

UNIX has a number of commands that can be used for displaying the contents of a file at the terminal.

cat

`cat`, which stands for "con**`cat`**enate and print," is the standard UNIX file display; it simply prints the file to the screen.[1] When piped to **`less`** (see section 5.3.4 which describes **less** as a filter), **cat** displays the file contents a screen at a time, and some simple commands may be executed at the supplied prompt.

`% cat` ***filename...*** **`[| less]`**

As its name suggests, **cat** is in fact quite useful for copying and concatenating files. Output is often piped to a file rather than to the screen, using standard output redirection (see section 5.3.2). The following example concatenates the three specified files and copies them sequentially to a single file called `allthreefiles`:

```
% cat fileone filetwo filethree > allthreefiles
```

1. There are better ways to display a file (see **less** and **more**, which follow **cat** in this section).

less, more

A shortcut for **cat** *filename* **|less** is to use **less** as a file browser:

% less *filename*

And wherever you use **less**, you can alternatively use **more**, although it is not as functional as **less**. You cannot move backwards through the file with **more**.

head, tail

head displays the first ***n*** lines of the specified file or files. If more than one file is specified, the filename is displayed before each set of file contents of ***n*** lines. ***n*** defaults to 10 lines.

% head [–*n*] [*filename...*]

tail displays the last lines of a file. Its syntax is a bit different:

% tail [+|–*n* lbc] [*filename...*]

The option **+*n*** displays the file starting ***n*** lines down from the beginning of the file, **–*n*** displays the last ***n*** lines. **l**, **b**, or **c** requests display of ***n*** lines, blocks, or characters (default is **l** lines). If more than one file is specified, the filename is displayed before each set of file contents. ***n*** defaults to 10 lines.

tail is useful when you want to see how far a process got. To display the last line of a log file, enter:

% tail –1 *logfile*

6.3.3 Copy a File: cp

The command **cp** (stands for **co**p**y**) can be used to make a copy of a file, leaving the original version intact. You can copy a single file to another one (in the same or a different directory), or you can copy one or more files to a different directory, retaining the same filenames. The syntax for these two situations varies slightly:

% cp [*options***]** ***file1 targetfile***	the file ***file1*** is copied to ***targetfile***, where ***targetfile*** may include a path
% cp [*options***]** ***file1*** **[***file2 ...***]** ***targetdirectory***	one or more files (***file1 file2*** ...) are copied to ***targetdirectory***

If the target is a file, its contents are overwritten unless **–i** is specified, in which case you are prompted for confirmation.

Some options are:

–i	If the target filename exists, you are prompted for confirmation before overwriting.
–r	Used only with the ***targetdirectory*** form. Recursively copy a directory, its files, and its subdirectories to ***targetdirectory***.

The first example below copies `myfile` to `anotherfile`, both in my current directory, prompting for verification in case `anotherfile` already exists.:

% cp –i myfile anotherfile

New users may find it useful to define **cpi** as the alias for **cp –i** to use in place of **cp** so that prompting always occurs. Section 8.3 discusses aliases.

The second example copies files `proj1` and `proj2` to another directory named `newproj` which is parallel to the current directory (has same parent directory as current):

```
% cp proj1 proj2 ../newproj
```

The third example copies the file `oldproj/proj1` to my current directory (`.`), which is a parallel directory to `oldproj` (has same parent directory). The file `proj1` keeps its name.

```
% cp ../oldproj/proj1 .
```

6.3.4 Move (Rename) a File: mv

The **mv** command (stands for **m**o**v**e) allows you to rename a file in the same directory or move a file from one directory to another. If you move a file to a different directory, the file can be renamed or it can retain its original name. **mv** can also be used to move and rename directories.

% mv [*options*] *source1* [*source2 ...*] *target*

Depending on whether the ***source(s)*** and ***target*** are files or directories, different actions are taken. These are described in the table below. If ***target*** is a filename, only one ***source*** file may be specified.

Source	Target	Result
file	***name***	Rename file as ***name***
file	existing file	Overwrite existing file with source file
directory	***name***	Rename directory as ***name***
directory	existing directory	Move directory to be a subdirectory of existing directory
one or more files	existing directory	Move files to directory

An important option is:

-i If ***target*** exists, the user is prompted for confirmation before overwriting.

6.3.5 Reference a file: ln

The **ln** command establishes an alternate name for a file or a directory. It provides a short hand for referencing files or directories that would ordinarily require a long path name. The syntax for **ln** is similar to that for **cp** and **mv**, and in fact they are all run by the same executable.

The syntax differs slightly for files and directories:

% ln [*options*] *file1 file2* Link ***file1*** to ***file2***. If ***file2*** already exists, it gets overwritten.

% ln -s [*other options*] *file1* [*file2 ...*] *directory*

Create symbolic link(s) named ***file1*** (***file2***..) in ***directory*** (the names do not change). Always use the **-s** option in this form; it creates a symbolic link that lets you see the name of the link when you run **ls -l**; it's the only way to know the name that a file is linked to.

For example, executing the command:

```
% ln -s mydata /e741/run1/e_mu2/r5742
```

where `/r5742` is a directory, not a file, allows you to reference the data file as `mydata` (i.e. same filename) as if it were in the directory `/e741/run1/e_mu2/r5742`.

On the other hand, if `r5742` is a filename in directory `/e741/run1/e_mu2`, you're establishing a link such that the filename `/e741/run1/e_mu2/r5742` is an alternate name for `mydata`.

The most commonly used options are:

-i	you are prompted before overwriting an existing filename.
-s	makes a *symbolic*, as opposed to an ordinary or *hard*, link. A symbolic link can point to a file that is on a different file system (on a different disk for example), whereas a hard link cannot.

Note that when using the AFS file system, hard links can only be made between files that are in the same directory, so use the **-s** option even if you're in the same directory tree.

Please see a good UNIX book for more in-depth discussions of linking.

6.3.6 Remove a File: rm

The **rm** command (stands for **r**e**m**ove) is used to remove the entries of one of more files.

% rm [*options*] *file...*

Some commonly used options are:

-i	Confirmation of removal of write-protected file occurs interactively, whether the standard input is a terminal or not. If used with the **-r** option, you are prompted about each directory before it is examined.
-r	Causes **rm** to delete the contents of the specified directory, including all its subdirectories, and the directory itself (**r**ecursive). This option should be used cautiously.

The file list can include ambiguous file references, so **rm** should be used cautiously. You can use the **echo** utility with the same ambiguous file reference to see the list generated.

Removal of a file requires write permission to its directory, but neither read nor write permission to the file itself. If the file has no write permission and the standard input is a terminal, the set of permissions is printed and you are prompted for confirmation. If the answer begins with a **y**, the file is deleted. If the standard input is not the terminal, the files are deleted without confirmation.

New users may find it useful to define **rmi** as the alias for **rm -i** to use in place of **rm** so that prompting always occurs. Section 8.3 discusses aliases.

6.3.7 Compress or Expand a File: gzip, gunzip

Several utilities are available on UNIX systems for file compression. **compress** and **pack** are native UNIX utilities, and **gzip** is provided by FUE. We recommend you use **gzip** for file compression, and its associated utility **gunzip** for file expansion. **gunzip** recognizes and can expand files compressed with **compress** and **pack** as well as **gzip**.

The file extensions **gunzip** recognizes include `.gz`, `-gz`, `.z`, `-z`, `_z`, and `.Z`. **gunzip** also recognizes the special extensions `.tgz` and `.taz` as shorthands for `.tar.gz` and `.tar.Z`, respectively. When compressing, **gzip** uses the `.tgz` extension if necessary instead of truncating a file with a `.tar` extension.

You will need to reference the man pages for details on syntax, options and usage. In their simplest forms **gzip** and **gunzip** can be used as follows, starting with the original uncompressed file ***filename***:

% gzip ***filename***

The result is *filename*`.gz`, whose size is reduced with respect to ***filename*** according to Lempel-Ziv coding (LZ77), the same compression scheme used by **compress**. Whenever possible, **gzip** replaces each file by one with the extension `.gz`, while keeping the same ownership modes and access and modification times. **gzip** will only attempt to compress regular files. In particular, it will ignore symbolic links. If the compressed file name is too long for its file system, **gzip** truncates it.

Compressed files can be restored to their original form using **gunzip**, or equivalently by using the **-d** option with **gzip**. If the original name saved in the compressed file is not suitable for its file system, a new name is constructed from the original one to make it "legal".

To restore *filename*`.gz` to its original name and size, enter:

% gunzip ***filename*****.gz**

To extract a tar file, after using **gunzip**, enter (the option may be entered as **xf** or as **-xf**):

% tar xf ***filename***

6.4 Information About Files

This section gives a cursory overview of simple uses for two very powerful commands for dealing with files: **find** for searching for files and **grep** for searching for strings within files. We also describe **wc** which displays the size of a file, **od** which creates a dump of a file, and **file** which can determine file type.

6.4.1 Find a File: find

The **find** utility tests each file in the given pathname list to see if it meets the criteria specified by the expression supplied. It does this by recursively descending the directory hierarchy for each path name. The format is:

% find ***path-name-list expression***

path-name-list can contain file expansion metacharacters. Each element in ***expression*** is a separate boolean criterion. A space separating two criteria is a logical AND operator, a **-o** separating the criteria is a logical OR operator. A criterion can be negated by preceding it with an exclamation point (**!**). Criteria are evaluated from left to right unless parentheses are used to override this. Special characters must be quoted (use ****) and there must be spaces on each side of the special character pair.

Some of the criteria that can be used within ***expression*** are:

-name ***filename***	True if ***filename*** matches the name of the file being evaluated. Ambiguous file references can be used if enclosed in quotes.
-type ***filetype***	True if the type of the file is ***filetype***, where ***filetype*** is either `d` (directory) and `f` (ordinary file).
-atime ***n***	True if the file has been accessed in ***n*** days.
-mtime ***n***	True if the file has been modified in ***n*** days.

-newer *filename*	True if the file has been modified more recently than ***filename*** has.
-print	Causes the matching path names to be displayed on the screen.
-exec *command* **\;**	True if ***command*** returns a zero exit status. ***command*** must be terminated with a quoted semicolon (note the ****). An empty pair of braces ({}) within the command represents the filename of the file being evaluated.
-ok *command* **\;**	Same as **-exec** except the generated command line is displayed and executed only if the user responds by typing `y`.

In the previous list, **+*n*** means more than ***n***, **-*n*** means less than ***n***, ***n*** means exactly ***n***.

Note that **find** doesn't do anything with the found files, it doesn't even display the names, unless instructed to.

Examples:

- Search the current directory and all subdirectories for the file `lostfile`:

```
% find . -name lostfile -print
```

- List all files ending in `.html` in your `/wwwork` subdirectory:

```
% find wwwork -name '*.html' -print
```

- This command will prompt you if you want to execute **more** on each file that begins with the letter `d` in the current directory and all subdirectories (Enter `y` if you want the file displayed.):

```
% find . -name 'd*' -ok more {} \;
```

- List all files in the current directory that *don't* begin with `m`:

```
% find . ! -name 'm*' -print
```

- Find all files in the current directory and all subdirectories that contain the string *hello*:

```
% find . -exec grep -l "hello" {} \;
```

- Remove all files in your directory tree that are named `a.out` or have the extension of `.o` and haven't been accessed in a week:

```
% find ~\(-name a.out -o -name '*.o') -atime +7 -exec rm {} \;
```

Note that using the find command uses many system resources.

In particular on AFS systems, you may accidentally end up searching servers all over the world if the top of the search is at the root directory (**/**). Generally you should be careful to only search the part of the UNIX tree that interests you. Here is an example:

- Look for ***filename*** starting at the root directory (**/**), and exclude searches in the `/afs` and `/nfs` branches:

```
% find / \( -name /afs -prune \) -o \( -name /nfs -prune \) -o -name filename -print
```

6.4.2 Search for a Pattern: grep

The **grep** utility searches the contents of one or more files for a pattern.

The format is:

% grep [*options*] *pattern* [*file* ...]

Some of the options are:

-c	Display only a count of lines that contain the pattern.
-i	Ignore upper/lower case distinctions during comparisons.
-l	Display only the name of each file that contains one or more matches.

The pattern can be a simple string or a regular expression (see section 5.3.5). You must quote regular expressions that contain special characters, spaces, or tabs (this can be done by enclosing the entire expression within single quotation marks).

Examples:

- Find all non-hidden files in the current directory containing the string *smith*:

```
% grep -i smith *
```

- Search the file `abc` for a string beginning with *f*, followed by 0 or more *r*'s, and ending in *og* (e.g., *frog, fog, frrog*):

```
% grep 'fr*og' abc
```

- Search the file `myfile` for a line beginning with a **T**:

```
% grep '^T' myfile
```

- Search `/usr/jones/junk` for the characters *file* followed by a number (e.g., *file1, file3*):

```
% grep 'file[0-9]' /usr/jones/junk
```

- Display a line if Smith is logged in:

```
% who | grep smith
```

- Show all processes being run by Smith:

```
% ps -ef | grep smith
```

- Show all environmental variables containing ***string*** in their name or their translation:

```
% env | grep string
```

- Show all aliases containing ***string*** in their name or their translation:

```
% alias | grep string
```

6.4.3 Count a File: wc

The **wc** command, which stands for **w**ord **c**ount, counts the number of lines, words, and characters there are in the named files, or in the standard input if the argument is absent. If there is more than one file, **wc** totals the count as well.

```
% wc [-lwc] [names]
```

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of **l**ines, **w**ords, and **c**haracters be reported. The default is **-lwc.**

UNIX users frequently count things by piping them into **wc**. For example, to display the number of users logged into the system, you can execute:

```
% who | wc -l
```

6.4.4 Dump a File: od

The **od** (**o**ctal **d**ump) command can be used to examine the contents of a file in various formats: octal, decimal, hexadecimal, and ASCII. The default is octal.

The format is:

```
% od [options] [file] [offset] [|less]
```

If ***file*** is not included, standard input is assumed. The options are:

-c	Produces a character dump.
-d	Produces a decimal dump.
-o	Produces an octal dump.
-x	Produces a hexadecimal dump.

The **-c** option prints non-printable characters as a printable character preceded by a backslash: **\0** is null, **\b** is backspace, **\f** is form-feed, **\n** is new-line, **\r** is return, and **\t** is tab.

The ***offset*** specifies where in the file the dump is to begin, if different than the beginning of the file. It is of the form **[+]*n*[.][b]**. The **+** is only necessary if you have no file specified so that the command interpreter knows this is the offset not the file. Without **.** or **b**, ***n*** indicates the dump starts at (octal) byte ***n*** of the file. A **.** displays ***n*** in decimal, a **b** in 512-byte blocks.

We recommend that you always pipe the output of **od** to **less** (or **more**) so that you can manipulate it. Large files can be unwieldy, and you may not be able to stop the output once it's going!

6.4.5 Determine File Type: file

The **file** utility can be used to determine the file type of a file according to its contents. It bases its guesses on a list of "magic numbers" recorded in a "magic file", `/etc/magic`. Some of the file types are:

- ASCII text
- C program text
- directory
- executable

file determines the filetype by looking at the beginning of the file and comparing it to entries in the magic file. The command format is:

% file ***filename...***

6.5 Manipulating Directories

This section describes the commands you can use to organize and use the UNIX directory structure. It describes how to make and remove directories, and move from one directory to another. Listing the contents of a directory (files and subdirectories) was described in a previous section. Section 6.6.3 explains the meaning of access permissions as applied to directories.

6.5.1 Print Working Directory: pwd

The **pwd** command (for **p**rint **w**orking **d**irectory) displays the path name of your working (current) directory. The command format is:

% pwd

6.5.2 List Directory Contents: ls

The **ls** command, which stands for **list**, is used to list the contents of a directory. **ls** has many options, some of which are system-dependent. A few of them are described in section 6.3.1. For a complete description of the command, refer to the man pages for **ls.**

6.5.3 Change Directory: cd

When you first log in to the system, you are placed in your home directory, which is then also your current working directory. You can use the **cd** command (for **c**hange **d**irectory) to change your current working directory. The command format is:

% cd [*directory*]

You can specify a complete path or a relative path. You can use **..** (for the parent directory) in your pathname. You must have execute permission (which provides search permission in this case) on a directory to **cd** to it.

If ***directory*** is not specified, you are returned to your home directory.

The following examples illustrate moving to different directories:

- your home directory

  % cd

- a subdirectory called `Tools`

  % cd Tools

- a colleague's home directory (using absolute pathname)

  % cd /usr/jones

- a colleague's subdirectory (using tilde)

  % cd ~jones/ourfiles

- a parallel directory (has same parent directory as current directory)

  % cd ../Tools

6.5.4 Make a Directory: mkdir

The **mkdir** command (for **m**a**k**e **dir**ectory) is used to create a directory. The command format is:

% mkdir *dirname ...*

If a pathname is not specified, the directory is created as a subdirectory of the current working directory. Directory creation requires write access to the parent directory. The owner ID and group ID of the new directory are set to those of the creating process.

Examples:

- create a subdirectory in the current working directory

  % mkdir progs

- create one with a full pathname (the directory `Tools` must already exist)

  % mkdir /usr/nicholls/Tools/Less

6.5.5 Copy a Directory

The most straightforward way of copying a directory and its contents is to pipe the output of the **ls** command (see section 6.3.1) into the file copy facility **cpio** (see the man pages).

First, create the destination directory using **mkdir** (see section 6.5.4), if it doesn't already exist. Secondly, from the source directory, run the command (shown with recommended options; see man pages for option information):

ls | cpio -pdmV *destination_directory*

The ***destination_directory*** must be specified relative to the source directory.

6.5.6 Move (Rename) a Directory: mv or mvdir

See section 6.3.4 for information on **`mv`**. To move a directory (***olddirname***) and its contents to a different position in the directory tree, use the command format:

`% mvdir` ***olddirname newdirpath***

If ***newdirpath*** exists already, then the directory gets moved to ***newdirpath/olddirname***. Note that the two arguments cannot be in the same path. For example:

`% mvdir x/y x/z`	is ok, but
`% mvdir x/y x/y/z`	is not ok.

6.5.7 Remove a Directory: rmdir

You can remove a directory with the **`rmdir`** command. The directory must contain no files or subdirectories, and you must have write permission to the parent directory.

`% rmdir` ***dirname*** **`...`**

You can use an absolute or relative pathname.

You can also use **`rm -r`** as described in Section 6.3.6. **`rm -r`** will delete a directory, all subdirectories, and all files. **This command should be used with extreme caution.**

For example, the following command deletes the directory `temp`, all subdirectories of `temp` and all files contained in those directories, prompting before each removal, and confirming removal of write-protected files (**`-i`**):

`% rm -ir /usr/jones/temp`

6.6 File and Directory Permissions

6.6.1 File Access Permissions

The UNIX file system allows you to control read, write, and execute access to your files on the basis of user (owner), group, and other (everyone else).[1] In this section we will consider only the standard UNIX file permissions. AFS file permissions are treated in the next section.

Note that in the AFS file system, file permissions are mediated by Access Control Lists (ACLs) that are set on a directory level. The standard UNIX file permissions don't apply in this case except for the owner permissions, which apply to all users. See section 6.6.2.

Also see the *FNALU User's Guide* (GU0008) for more information on AFS permissions.

To determine the current permissions, use the long form of the **`ls`** command, **`ls -l`**. Referring to the example below, the nine characters immediately following the first field represent the one-bit flags known as the *mode bits* that control file access. A dash indicates a bit is not set, **`r`** stands for

1. Note **o** is for **other** and not for **owner** as on VMS.

read access, **w** for **w**rite access, and **x** for e**x**ecution access. The first set of three characters refer to owner permission, the middle three for group permission, and the last three for all other user classes.

```
total 251
drwxr-xr-x 3 nicholls g020c 512   May 2  08:53 Tools
drwxr-xr-x 2 nicholls g020c 512   May 2  09:01 bin
-rw-r--r-- 1 nicholls g020c 446   May 4  14:09 defaults
-rw-r--r-- 1 nicholls g020c 95418 May 1  17:42 intro.lpr
-rw-r--r-- 1 nicholls g020c 0     May 10 17:51 lsout
-rw-r--r-- 1 nicholls g020c 6683  May 1  16:46 man.lpr
-rw-r--r-- 1 nicholls g020c 12258 May 9  16:16 out
```

In the example, ignoring the directory files (which have a **d** in position 1), the owner has **rw** access to the files, whereas group and others have read (**r**) access only.

chmod

The **chmod** command, which stands for **ch**ange **mod**e, is used to change access permissions of a file or directory:

% chmod ***mode filename ...*** or

% chmod ***mode directory ...***

In the *absolute form* of the mode where the level of protection is specified in octal format, ***mode*** looks like 741 or 554, for example, where each of the three octal numbers represents the sum of the permissions granted to its class: user, group, and other, in that order. The three types of permission have the values:

read	4 (100 octal)
write	2 (010 octal)
execute	1 (001 octal)

For example, a mode of 741 means owner can read, write, and execute (4+2+1=7); group can read (4+0+0=4); and others can execute the file (0+0+1=1). To give this permission to a file `test`, you would enter:

```
% chmod 741 test
```

You can use an alternate form of ***mode*** in the **chmod** command in which ***mode*** is a three-character field specifying an action to be taken. The action is to add or subtract one or more permissions from one or more user classes. It takes the form:

who operator permission(s)

These three positions within the field take the following characters:

who	represents the user class or classes; it takes any combination of **u**, **g**, **o**, and **a** for **u**ser, **g**roup, **o**ther and **a**ll (*all* includes the three individual classes)
operator	**+** or **–** for adding or subtracting permissions, or **=** for setting a specific permission and resetting all other permissions for the specified user class(es)
permission(s)	any combination of **r**, **w**, and **x** for **r**ead, **w**rite, and e**x**ecute, indicating the permissions to be permitted, denied, or reset.

Examples of the **chmod** command:

• Remove group execute permission to the file `progs`:

```
% chmod g-x progs
```

• For the files `out` and `out1`, add group read and write, and deny write to other:

```
% chmod g+rw,o-w out out1
```

• Set group read permission and reset all other group permissions to `myfile`:

```
% chmod g=r myfile
```

Note that classes of users or levels of protection not specified in a command are not modified in this form of the command (with the exception that **`=`** resets other permissions).

umask

With the **`umask`** command you can specify a mask that the system uses to set access permissions when a file is created. In order to understand **`umask`** you need to know that access permission at file creation is application-dependent. Each command or application sets a file permission in its *open* command.[1] The system then "subtracts" any user-defined mask, resulting in the final access permission for the file. You can set a umask by this command:

```
% umask [ooo]
```

where ***ooo*** stands for three octal digits. The user-specified "mask", ***ooo***, has the same positional structure as described above for **`chmod`**, but specifies permissions that should be **removed** (disallowed).

For example, a mask of 022 removes no permissions from owner, and removes write permission from group and others. Thus a file normally created with 777 would become 755 (this would appear as `rwxr-xr-x` in the format put out by the command **`ls -l`**). The following command could be put in your `.cshrc` or `.profile`.

```
% umask 022
```

The meaning of permissions applied to directories is described in Section 6.6.3.

6.6.2 AFS File Permissions

File permissions work quite differently from those in standard UNIX, which are described in section 6.6.1. In AFS, you can still use the standard **`chmod`** command just as you would in a standard UNIX file system. However, the behavior of the file will be different. Although in the AFS file system all of the bits on a file may be examined or changed, **only the owner bits are actually used in AFS, and they apply to all users of the file (as permitted by users' ACL settings; see below)**. Thus to turn off write access to a particular file, you need to turn off the owner write bit of the file. This inhibits writing to that file for all users, including the owner.

Other AFS permissions are done with ACLs (Access Control Lists) which take effect at the directory level only. ACLs can be set separately on a directory for different users and groups. In general users are granted all permissions on their home directories. When a child directory is created, it inherits the parent directory's ACLs. These ACL rights include:

`l`	lookup rights
`i`	insert rights
`d`	delete rights
`a`	administrator rights
`r`	read rights

1. Normally only the loader creates files with execute permission.

w	write rights
k	lock rights

The specific actions permitted for each category are enumerated in the *FNALU User's Guide*.

You can examine a directory's ACL rights with the command:

% fs listacl ***dir_path***

You can change them using the command:

% fs setacl ***dir_path AFS_principal permission(s)***

where ***AFS_principal*** typically is of the form ***group_owner:group_name***. For example, to set the ACL for the current directory to include read and lookup rights only for any user, enter:

% fs setacl . system:anyuser rl

See **man fs_listacl** and **man fs_setacl** for further information. (Note the underscores here.)

6.6.3 Directory Permissions

You can grant or deny permission for directories as well as files, and protection assigned to a directory file takes precedence over the permissions of individual files in the directory.

- Read permission for a directory allows you to read the names of the files contained in that directory with the **ls** command, but not to use them.
- Write permission for a directory allows you to create files in that directory *or* to delete any file in the directory, regardless of the file protection on the files themselves. It does not allow you to see the files or use them without **r** and **x** directory permission. In other words, write permission to a directory allows you to alter the contents of the directory itself, but not to alter, except to remove, files *in* the directory (which is controlled by the file's permissions).
- Execute permission allows you to list the contents of the directory.

File access permissions of directory files are changed with the **chmod** command (see section 6.6.1).

6.7 Temporary Directories

By convention, there are directories named `/tmp` (and sometimes `/usr/tmp`) where programs and users can store temporary files. Many programs (e.g., compilers) write temporary files there or in the area specified by the environment variable *TMPDIR*. Since these are public areas, it is necessary to manage this space, which means that you cannot count on files being retained in these directories.

Many systems on site have fairly small `/tmp` areas and therefore you must be careful not to fill up this space. In general, you should only use `/tmp` for very temporary, small files. On many systems files in `/tmp` will disappear after a reboot or after existing for a week. You can set *TMPDIR* to a different location if there is not enough space in these areas.

Contact the administrators of the particular system to find out what the current policy is on the machine.

Chapter 7: Printing

This chapter covers the standard FUE print utility **flpr**, as well as filter programs and techniques available for formatting the output prior to printing. **flpr** is the Fermi implementation of the standard UNIX **lpr** utility. Most software applications supplied by the Computing Division use **flpr** as a default.

Note that many X-windows applications allow you to specify in a pop-up window the print command you want to use.

7.1 The FUE Print Command: flpr

The Fermilab print utility **flpr** (usually pronounced "flipper") implements the **lpd** printing protocol, and can be used to print UNIX files to any print server using this protocol, including VMS print queues and most UNIX systems. Related commands are **`flpq`** (to check the print queue) and **`flpk`**[1] (to kill a submitted print job). The format of each command is:

`% flpr [`*options*`] [`*file*`] ...`

`% flpq [`*options*`] [`*users*`|`*jobs*`] ...`

`% flpk [`*options*`] [`*users*`|`*jobs*`] ...`

The options are described in the man pages. All three programs will display a list of their options when given an argument of **`-\?`**. Which options actually function depends on the functions supported by the **lpd** host.

The following command produces the version of **flpr** in use, when and where it was created, the **flpr** defaults in effect, and the command format:

`% flpr -\? -v`

To print a file, generally you will just need to enter the following:

`% flpr -q` ***queue file***

This assumes that your implementation of **flpr** uses FNPRT.FNAL.GOV as the default host computer (most do), and that your printer is known to FNPRT (most are). If the **`-q`** option is not specified, the queue is taken from (in order of precedence):

1) environment variable *FLPQ*

2) user control file (`.flprrc`)

3) system control file(s)

4) data compiled into **flpr**

---

1. Many hosts do not support **`flpk`**.

A full list of queues supported by FNPRT is available by running the command **`obtain printer`** to create the file `flp.printers` in your directory[1].

If your printer host is not known to **flpr**, you can use the option **`-h`** to specify it on the command line, set your own default printer host with the environment variable *FLPHOST*, or put an appropriate entry in your `.flprrc` user control file.

You can set environment variables such as *FLPQUE* and *FLPHOST* with the **`setenv`** command for the C shell family (see section 8.1):

`% setenv FLPQUE` ***queue***

`% setenv FLPHOST` ***host***

and for the Bourne shell family:

`$ FLPQUE=`*`queue`*`; export FLPQUE`

`$ FLPHOST=`*`host`*`; export FLPHOST`

A useful feature of **flpr** is the capacity to use *nicknames.* A nickname specifies the printer host computer and the queue as a pair. If you often use a printer unknown to FNPRT, this feature may be handy for you. You would then enter a print command in this format:

`% flpr -P` ***nickname file***

The `flp.printers` file mentioned above defines many nicknames, most or all of which are associated with the host FNPRT, and is therefore more useful as a queue reference for the **`-q`** option. An example nickname is `wh10w_lw`, which is defined as:

```
printer wh10w_lw fnprt.fnal.gov wh10w_lw
```

Why are printer nicknames useful? If you're using a printer not recognized by the **flpr** defaults, you'll need to specify both host and queue unless you use a nickname. Also, we recommend that you use nicknames rather than queue names in shell scripts. If a printer fails or is removed, then its associated nickname can be redefined, and the scripts don't need to be changed. If a queue name is specified in a shell script, then this capacity does not exist. You can define a nickname yourself, or if a widely-used printer is down, it may be appropriate for your system administrator to establish a substitute printer by changing the nickname in `flp.printers`.

A printer nickname does not have to be the same as the printer queue name. It may be a short, easily remembered name, such as *ps* for a postscript printer or *lp* for a text printer. Such personal printer nicknames can be established in your personal **flpr** control file, `$HOME/.flprrc`.

As an example `.flprrc`, assume you use the FNPRT queues `b0trwqms_hp` for text printing, and `b0trwqms_ps` for Adobe PostPscript printing. You want to use the nicknames *lp* for text printing and *ps* for PostScript. Further, you would like the print jobname to contain identifiers for your login name and the file being printed. You could build a `.flprrc` as follows[2]:

```
host fnprt.fnal.gov
queue b0trwqms_ps
identifier %l$%b
printer ps fnprt.fnal.gov b0trwqms_ps
printer lp fnprt.fnal.gov b0trwqms_hp
```

1. This file actually specifies printer-queue combinations. These names are the same as the queues when FNPRT is the host, the case for most if not all of the entries.
2. The *identifier* in the file is equivalent to the **`-I`** option described in the **flpr** man page.

If this file exists, it will be used automatically. **flpr** also accesses any system-wide defaults files, such as `flp.defaults` and `flp.printers` stored in a system directory. The `flp.defaults` file contains default values for the host, username, printer queue, protocol temporary filename format and one or more associations between a nickname and a host and printer queue pair.

Another option for the **`flpr`** command that may occasionally be useful is **`-l`** ***login-name***. This specifies the remote login name to be used in case you want another user name associated with the print job. The default name is the same as that on the current system.

From the *UNIX Resources* page on WWW, see *Printing from UNIX at Fermilab* under **Our Local Environment** for information on the Fermilab print server, and listing and checking print queues.

7.2 Pre-Printing Options

a2ps and **psnup** are described in detail in the man pages and in the Fermilab DCD Release Note 41.0 *Users Guide to UNIX Printing Utilities*. These utilities can be used from the UNIX command line as well as from many applications. **pr** is also available to format the pages before printing with **flpr**. **pr** is explained in the man pages.

7.2.1 a2ps

The **a2ps** utility converts ASCII to two-column PostScript, and encloses the text in boxes with headers indicating date, time, filename and page number. You also get a line saying "Printed by ***username*** from ***node***". **a2ps** is useful if the printer can only accept PostScript. You can use it to produce different output formats (e.g., "2-up", where two pages of your file are shrunk to fit on a single sheet). It provides options to add items like line numbers, user, file, and system information to the output. The command syntax is:

`% a2ps [`*`global options`*`] `*`files`*` [`*`positional options`*`] `*`files`

where ***global options*** apply to all files being printed, and ***positional options*** are applied only to the files found in the remainder of the command line (with the exception of **`-H`** as noted in the man page which only applies to the *next* file).

Here is an example where we pipe the output of **`a2ps`** to **`flpr`** (**`-p`** indicates portrait mode):

`% a2ps -p `*`file1`*` | flpr -q wh10w_lw`

7.2.2 psnup

The **psnup** utility takes a PostScript file and prints it "***n***-up". This refers to how many pages get printed on a single sheet. Portrait and landscape mode alternate as ***n*** changes. Occasionally this procedure doesn't work due to problems with PostScript variance.[1] The syntax of the command is:

`% psnup [-p`*`n`*`] [-r] [-R] [-s`*`n`*`] `*`files`

where:

`-p`*`n`*	the number of pages of PostScript (or *spots*) that should be printed per sheet of paper. ***n*** is constrained to be 2, 4, 8, 16, 32, or 64.

1. **psnup** inserts PostScript code in front of the PostScript it is given. Because PostScript is a programming language, this doesn't always work. The file may be printed in its original format, it may never be printed, or it may print fine!

-r	sets the first spot in lower right and progresses horizontally to the upper left. This is handy for when the pages have already been reversed by another program, and you are printing on a printer that reverses pages.
-R	sets the first spot in the upper left hand corner and progresses horizontally to the lower right. This is for non-reversing printers.
-s*n*	***n*** is the number of the spot you want the first page of output to be placed. *The first spot on the page is 0, not 1.*

7.2.3 Setting Duplex Mode

Duplex mode refers to printing on both sides of the paper. This is often desirable for larger documents, although not all printers support this feature.

PostScript Files

To print a PostScript file in duplex mode on a printer that supports this feature, you must prepend some specific text to the file. An easy way to do this is to maintain this text in a separate file, and concatenate it to your PostScript file when you are ready to print it in duplex mode. The two-line text file, which we'll call `duplexps` here, must have the following contents:

```
%!
statusdict begin true setduplexmode end
```

To print your PostScript file in duplex mode, enter the command:

% cat duplexps ***postscript_file*** **| flpr -q*****duplex_printer***

Text Files

For text files, you need to prepend a different sequence to the file. The technique we present here works with any PCL-based printer with a duplex device in it (assuming no filters are in the way) . Since text files are easy to edit, just add the following line to the top of your file:

```
<ESC>&l1S<ESC>&a0G
```

The trick is to get the *escape* character in your file! The sequence **<CTRL-[>** will work for escape, but it must be preceded by a "quoting" character, which differs from editor to editor.

For this (these) editor(s):	... enter:
vi	**<CTRL-V> <CTRL-[>**
emacs	**<CTRL-Q> <CTRL-[>**
nu/TPU, fermitpu	**<CTRL-V> <CTRL-V> <CTRL-[> <CTRL-[>**
EDT+	**<CTRL-[> <CTRL-[>**
NEdit	Use **INSERT CONTROL CHARACTER** from the **EDIT** menu; escape is 27 decimal. The escape character may not echo on your screen depending on your font.

... to insert an escape character. Then print the file in the usual manner to an appropriate printer.

7.3 Other Print Utilities

There are two other traditional printing systems under UNIX. One of these is **lpr**, upon which **flpr** is based. **lpr** requires that each printer be defined in the local `/etc/printcap` file, which must be done by the system administrator. One of **flpr**'s advantages is that no such definition is required since the host name and print queue can be specified on the command line. **lpr** is the default print command used by some applications, however. You can obtain in your current directory the file `printcap` which contains all printcap entries for print queues defined on FNPRT by executing the command:

```
% obtain printcap
```

The other traditional printing system is called **lp**. This has its own set of files which must be set up by system administrators and is recommended for use only in situations where the other options are not available to you.

Chapter 8: Working Environment

This chapter describes the methods used to set up your working environment in UNIX. Some of these are standard UNIX (e.g., shell and environment variables), and some are provided and/or customized by FUE (e.g., the login scripts).

8.1 Shell Variables and Environment Variables

Every UNIX process runs in a specific *environment*. An environment consists of a table of *environment variables*, each with an assigned value. When you log in certain *login files* are executed. They initialize the table holding the environment variables for the process. (Exactly which files run will be made clear later in this chapter.) When this file passes the process to the shell, the table will be accessible to the shell. When a (parent) process starts up a child process, the child process will be given a copy of the parent process' table. Environment variable names are generally given in upper case.

The shell maintains a set of internal variables known as *shell variables*. These variables cause the shell to work in a particular way. Shell variables are local to the shell in which they are defined; they are not available to the parent or child shells. Shell variable names are generally given in lower case in the C shell family and upper case in the Bourne shell family.

8.1.1 C Shell Family

The C shell family explicitly distinguishes between *shell variables* and *environment variables*. A shell variable is defined by the **`set`** command and deleted by the **`unset`** command. The purpose of your `.cshrc` file (discussed later in this chapter) is to define such variables for each process. To define a new variable or change the value of one that is already defined, enter:

`% set` ***name=value***

where ***name*** is the variable name, and ***value*** is a character string that is the value of the variable. If ***value*** is a list of text strings, use parentheses around the list when defining the variable. The **`set`** command issued without arguments will display all your shell variables. You cannot check the value of a particular variable by using **`set`** ***name***, omitting ***=value*** in the command; this will effectively unset the variable. To see the value of a single variable, use the **`echo`** command:

`% echo $`*`name`

If the value is a list, to see the value of the ***n***th string in the list enter:

`% echo $`*`name``[`*`n`*`]`

To delete, or unset, a shell variable, enter:

`% unset` ***name***

Environment variables are set by the **setenv** command, and displayed by the **printenv** or **env** commands, or by the **echo** command as individual shell variables are. Some environment variables are set by default (e.g., *HOME*, *PATH*).

The formats of the commands are (note the difference between **set**, above, and **setenv**):

% setenv [*NAME value*]

% unsetenv *NAME*

where ***value*** is interpreted as a character string. If the string includes blanks, enclose the string in quotes ("). The current environment variable settings can be displayed using the **setenv** command with no arguments.

To use a shell or environment variable in a command, preface it with a dollar sign (**$**), for example ***$name***, or ***$NAME***. This tells the command interpreter that you want the variable's value, not its name, to be used. You can also use **${*name*}** or **${*NAME*}**, which avoids confusion when concatenated with text.

Note that when a shell is started up, four important shell variables are automatically initialized to contain the same values as the corresponding environment variables. These are *user*, *term*, *home* and *path*. If any of these are changed, the corresponding environment variables will also be changed.

8.1.2 Bourne Shell Family

The Bourne shell family does not really distinguish between shell and environment variables. When a shell starts up, it reads the information in the table of environment variables, defines itself a shell variable for each one, using the same name (also uppercase), and copies the values. From that point on, the shell only refers to its shell variables. If a change is made to a shell variable, it must be explicitly "exported" to the corresponding environment variable in order for any forked subprocesses to see the change. Recall that shell variables are local to the shell in which they were defined.

Shell variables are defined by assignment statements and are unset by the **unset** command. The formats of the commands are as follows:

$ *NAME=value*[; export *NAME*][1]

$ unset *NAME*

where ***NAME*** is the variable name, and ***value*** is a character string that is the value of the variable. If the string includes blanks, enclose the string in double quotes. The values of all the current variables may be displayed with the **set** command.

To use a variable in a command, preface it with a dollar sign (**$**). This tells the command interpreter that you want the variable's value, not its name, to be used. For example, to see the value of a single variable, enter:

$ echo $*NAME*

You can also use **${*NAME*}**, which avoids confusion when concatenated with text.

---

1. In most cases you will want to include the optional part of the command, so that it reads: ***NAME=value*; export *NAME***

8.2 Some Important Variables

These variables are important for all shells, unless noted otherwise.

DISPLAY

In order to use an X windows application, the environment variable *$DISPLAY* must be set correctly. Normally the Fermi login files set it correctly for you. Its value is of the form ***node:screen.server***. At Fermilab, this will generally look like ***node*`.fnal.gov:0.0`** where *node* is your machine name.

Finding the value of *name* is different on different types of terminals and workstations:

UNIX workstation	On a local window, run **`funame -n`** to get the name.
NCDX terminal	("X-terminal") Use the TCP/IP Name found under **SHOW VERSION** from the startup menu.
Tektronix X terminal	Enter setup by pressing **F3** (the **SETUP** key). Select **SETUP**. Under **CONFIGURATION SUMMARIES**, select **TCP/IP**. A screen appears with the nodename. This is the name you want.
VMS workstation	If the workstation is connected directly to the monitor, then use the name of the workstation.

HOME

Your home directory is the top of your personal branch in the file system, and is usually designated by your username, for example `/path/username`. The value of the variable *HOME* is the pathname of your home directory. The command **`cd`** without arguments always returns you to *$HOME*. In all shells except **sh**, the tilde (**~**) symbol used in filename expansion, expands to the value of this variable. For example `~/myfile` is equivalent to `$HOME/myfile`. `~username` is equivalent to the $HOME directory of user *username*.

PATH

The *PATH* variable lists the set of directories that the shell looks in to find the commands that you enter on the command line. (For the C shell family, the shell variable *path* takes its value from *PATH*.) If the path is set incorrectly, some commands may not be found. If you enter a command with a relative or absolute pathname, the shell will only search that pathname for it, and not refer to *PATH*.

If you include the current working directory, "dot" (`.`), in your *PATH*, the shell will always find your current working directory. This allows you to run executable files from your current working directory by typing in only the filename. The Fermi login files include the dot at the end of the path for you.

For the C shell family, see the following line in the `setpath.csh` file (see section E.1.3):

```
set path = ($path .)
```

For the Bourne shell family, see the following line in the `setpath.sh` file (see section E.2.3):

```
PATH = ${PATH}.:
```

See section 8.4 for information on these files. If "dot" is not in your *PATH*, then in order to execute a file, you need to precede the executable filename by **`./`** on the command line. This provides the current directory pathname explicitly.

MANPATH

The *MANPATH* variable lists the set of directories that the shell looks in to find man pages.

SHELL

This variable is set to your default shell. Your default shell is determined by the last field in your password entry (see section 4.1.2).

ignoreeof

This shell variable is in **csh**, **tcsh**, **ksh** and **bash**. **sh** doesn't have it. When the *ignoreeof* variable is set, you cannot exit from the shell using **<CTRL-D>**, so you cannot accidentally log out. You must use **`exit`** or **`logout`** to leave a shell (see section 2.2).

noclobber

This shell variable is in **csh**, **tcsh**, **ksh** and **bash**. **sh** doesn't have it. With the *noclobber* variable set, you are prevented from accidentally overwriting a file when you redirect output. It also prevents you from creating a file when you attempt to append output to a nonexistent file.

noclobber has no effect on utilities such as **cp** and **mv**. It is only useful for redirection. See sections 5.3.2 and 6.3.

8.3 The Alias Command

The **`alias`** command allows you to create your own names or abbreviations for commands by performing string substitution on the command line according to your specifications. Aliases are recognized only by the shell that invokes them, they do not get "inherited".

Never use the actual command syntax as an alias for itself. If for some reason an error occurs and the login file which defines your aliases doesn't run, UNIX executes the standard version of the command. Normally you'd see an error message in this case, but what if you miss it? This can be disastrous. For example, if you are accustomed to using **`rm`** (*remove file(s)*, see section 6.3.6) as an alias for **`rm -i`** (*remove file(s), but prompt for confirmation*), when you run **`rm`** you will expect a confirmation prompt. If the alias didn't get defined you won't get a prompt, and you may end up removing files you need. That is why we suggest **`rmi`** as an alias for this command.

8.3.1 C Shell Family

The format of the **`alias`** command is:

`% alias [`*`new`* **`[`*`old`***`]]`**

When you enter ***new*** the shell substitutes ***old***.

The first example causes **`ls -l`** to be executed when the command **`ll`** is entered:

```
% alias ll ls -l
```

The next example creates the command **`dir`** to list directory files only:

```
% alias dir 'ls -l | grep ^d'
```

`grep` in this case searches for a **`d`** in the first column of each line.

8.3.2 Bourne Shell Family

Alias

The **alias** command is supported by **ksh** and **bash**, but not **sh**. For the entire Bourne shell family you can use *shell functions* instead of aliases; we discuss these below. The format of the **alias** command is:

% alias ***name*** **= '*alias_contents*'**

The first example causes **ls -l** to be executed when the command **ll** is entered:

```
% alias ll='ls -l'
```

The next example creates the command **dir** to list directory files only:

```
% alias dir='ls -l | grep ^d'
```

grep in this case searches for a **d** in the first column of each line.

Shell Functions

The Bourne and Korn shells support shell functions, which are similar to shell scripts in that they store a series of commands for execution at a later time. Shell functions are more quickly accessed than scripts because they are stored in memory instead of a file, and the shell preprocesses them. They can be used in place of aliases in order to be completely portable between **sh**, **ksh**, and **bash**. For information on the format of shell functions and how to use them, refer to the **sh** man pages or a UNIX text.

8.4 Tailoring Your Environment

This section discusses the FUE-customized login files (also called the *Fermi files*) used to set up your UNIX environment. All the files discussed in this section are printed in Appendix E. You will automatically have your own copy of these files in your home directory[1]. The default files exist in `/usr/local/etc` and you can recopy them to your home directory if you ever need to. Once you understand the functions of the various files, you can tailor them to suit your tastes.

Many of these files include sample code that you may want to activate. A `#` in the first column indicates a comment line. To activate a command line that's been "commented out", remove the `#`.

8.4.1 C Shell Family Fermi Files

The C shell executes *hidden* FUE-customized files at various times in your session. They include the files `.cshrc` and `.login`, which you may choose to further modify.

When you log out, the shell looks for a logout script in your home directory called `.logout`. FUE does not provide this file, but you can create it yourself, and it will get run automatically. This file is not required.

As an example, the following `.logout` file simply clears the screen:

```
clear
```

---

1. Exceptions are the `fermi.*` and `setup.*` files which are called directly from `/usr/local/etc`.

.cshrc and fermi.cshrc

Upon logging in, the first file to execute is the `.cshrc` located in your home directory. The shell also executes this file each time you invoke a new C shell, for example when you execute a C shell script or otherwise fork a new process.

Your `.cshrc` file first executes `/usr/local/etc/fermi.cshrc`, which:

- sets up the machine id, type, and operating system
- sets up **ups**
- establishes *path* (and therefore *PATH*) by running `/usr/local/etc/setpath.csh` (see section E.1.3), and *MANPATH*
- sets *fermimail* as the standard mail alias

`fermi.cshrc` also calls `/usr/local/etc/local.cshrc` which sets a couple of environment variables.[1] You do not have a copy of `fermi.cshrc` in your home directory; it is not designed to require individual customization. See section E.1.2 for the file listing.

The file `.cshrc` should contain all your aliases; many suggested aliases are provided for you to activate, and you can define your own. You can also set variables (*noclobber* and *ignoreeof* are already set for you) and parameters that are local to a shell.

Don't set any environment variables here. Any changes to their values will remain after you terminate a forked process, thus changing your standard environment for the duration of your login session. See section E.1.1 for the default file listing.

.login and fermi.login

The `.login` file is executed only at login time. After execution of `.cshrc`, the `.login` file located in your home directory is run. The default `.login` file first executes the file `/usr/local/etc/fermi.login`.

`fermi.login` performs several actions:

- sets **umask** (default file access permissions) so others can read and execute but not modify or delete your files
- determines the terminal type (and makes "best effort" at determining *DISPLAY* variable)
- sets common terminal characteristics
- sets a host of environment variables

You do not have a copy of `fermi.login` in your home directory; it is not designed to require individual customization. See section E.1.6 for the file listing.

Next, the `.login` file sets your prompt, and sets the variables *history* and *savehist*. You can edit your `.login` to modify your path and/or terminal settings, change the default values of environment variables or create your own, and/or include commands that you want to execute once, at the beginning of each session (for instance **`setup`** ***product*** commands). See section E.1.5 for the default file listing.

.logout

The C shell executes the `.logout` file in your home directory (if you have created one) when you log off the system.

---

1. This is a file for things that the local system manager wants to add to the login scripts. It may or may not have been created on your system.

Execute files to modify current session

If you modify your `.cshrc` or `.login` files and you want them to take effect in the current session, you must execute them with the **`source`** command:

```
% source .cshrc
```

```
% source .login
```

8.4.2 Bourne Shell Family Fermi Files

The Bourne shell executes *hidden* FUE-customized files at various times in your session. When you log on in the Fermi environment, the `.profile` and `.shrc` files in your home directory are executed for **sh**, **bash**, and **ksh**. Your `.shrc` file is also executed at any time a new **bash** or **ksh** is invoked.

The name of the file `.shrc` is determined by the *ENV* environment variable which is set to `~/.shrc` in the standard `.profile`, it is not a standard UNIX feature.

.profile and fermi.profile

The `.profile` file first executes `/usr/local/bin/fermi.profile`. This file performs several actions:

- sets **umask** (default file access permissions) so others can read and execute but not modify or delete your files
- sets up the machine id, type, and operating system
- establishes *PATH* (by running `/usr/local/etc/setpath.sh`; see section E.2.3) and *MANPATH*.
- determines the terminal type
- sets a host of environment variables
- sets common terminal characteristics

You do not have a copy of `fermi.profile` in your home directory; it is not designed to require individual customization. See section E.2.2 for the file listing.

The `.profile` file sets your prompt and the variables that govern your history list, your default editor, and your command line editor. You can edit your `.profile` to modify your path and/or terminal settings, change the default values of variables[1] or create your own, and/or include commands that you want to execute once, at the beginning of each session (for instance **`setup`** ***`product`*** commands). See section E.2.1 for the default file listing.

.shrc and fermi.shrc

The `.shrc` file first executes `/usr/local/etc/fermi.shrc` which sets up **ups** and performs some machine-dependent functions. See section E.2.5 for the default file listing.

The `.shrc` file should contain all your aliases;[2] many suggested aliases are provided for you to activate, and you can define your own. You can also set variables (*noclobber* and *ignoreeof* are already set for you except in **sh**) and parameters that are local to a shell, and you can activate and define functions. See section E.2.4 for the default file listing.

---

1. Remember for **sh**, there is not really a difference between shell and environment variables; see section 8.1.
2. Aliases are available for **bash** and **ksh**, but not for **sh**; see section 8.3.2.

Execute files to modify current session

If you modify your `.shrc` or `.profile` files and you want them to take effect in the current session, you must execute them with the `.` command:

```
$ . .shrc
$ . .profile
```

8.4.3 Storing Customized Code

If you wish to maintain versions of distributed code customized to your own needs, we recommend that you store them in the following directories:

`$HOME/bin`	for machine-neutral code
`$HOME/bin.$ARCH`	for architecture-specific code; `$ARCH` is the value returned by **`funame -s`**.

The path names for these directories will be added to your *PATH* when the Fermi files are invoked.

8.5 Multimedia Support

Multimedia support in mail and Web browsers is based on Multipurpose Internet Mail Extensions, or MIME. MIME defines a number of content types and subtypes that allow programs like **mosaic** to recognize different kinds of files and deal with them appropriately. The MIME type specifies what kind of file it is, such as image, audio, or video, and the subtype gives the precise format. On UNIX, MIME uses a `.mailcap` file to map MIME types to external viewer applications. The `.mime.types` file lists the associations between the MIME types and the file extensions.

8.5.1 The .mailcap File

When you first setup **www**, it creates a `.mailcap` file in your `$HOME` directory, if the file doesn't already exist. If an earlier version is found, the terminal displays a message saying that you can update it from the file in `$WWW_DIR/lib/TypeMap`. The `.mailcap` file's purpose is to provide a recipe for displaying multimedia files, and "playing" audio files. Its name refers to the fact that it was originally designed for multimedia mail, however its role has since expanded, and will probably continue to do so as more and more programs become multimedia. Any MIME-compliant mailer (e.g., **exmh**, **pine**) uses it, as do many Web browsers[1] to display a variety of filetypes that are downloaded over the net.

Each line of `.mailcap` consists of two fields separated by a semicolon (;), where the first field is the filetype (a generic or specific image or audio type), and the second is a **printf**-style format statement (see the man page for **printf**) for printing the command that invokes the viewer. Let's look at a couple of example lines:

```
image/jpeg; display %s
application/pdf; acroread %s
application/x-pdf; acroread %s
```

The first field in the examples, `jpeg` and `pdf`, are image filetypes. A generic image filetype would have an asterisk, e.g., `image/*`.

---

1. **mosaic** uses it, but **netscape** knows how to view lots of filetypes itself, and only relies on `.mailcap` if it can't find how to display a particular filetype.

The second field in the examples contains a command. **display** and **acroread** are the programs that are called when the corresponding filetypes are encountered.[1] The `%s` is a **printf**-style parameter for the string representing the filename.

For most situations, the `.mailcap` file should be sufficient as provided. Note that Adobe Acrobat filetypes (with extension `pdf`) are not included in your default file. You might want to modify `.mailcap` if:

- you want to add other MIME types such as Adobe Acrobat (`pdf` filetypes) or audio, or
- you have a different viewer that you prefer over **GhostView/GhostScript**

8.5.2 The .mime.types File

A default `.mime.types` file is not provided. To add support for a new MIME type, you'll need to create this file to provided the extension map. To illustrate the file contents, let's assume you want to add support for `pdf` files. You would include the following line:

```
application/pdf pdf
```

More information on the `.mailcap` and `mime.types` files can be found on WWW under the heading **WWW** from the *UNIX Resources* page.

8.6 Terminal Characteristics

You can specify your terminal type to UNIX if the default is not suitable. To do so, enter the command for the C shell family:

```
% set term=termtype
```

or for the Bourne shell family:

```
$ TERM=termtype; export TERM
```

where ***termtype*** is the name of a terminal type supported on the system. VT100 and VT220 are acceptable terminal types. If you always use the same kind of terminal, you may want to put this command in your `.login` or `.profile`. Note that the standard Fermi files attempt to set this variable correctly.

In Section 2.4 we listed some terminal control functions. Recall that you can display the settings with the **`stty`** command:

```
% stty -a
```

The format on each machine is different but should indicate approximately the same information. The following is the output from a Silicon Graphics workstation. The settings reflect the FUE defaults.

---

1. **display** is available as part of the **ups** product **ximagetools**, which can be setup either as a subproduct of **www**, or on its own. This allows you to setup **ximagetools** directly to view the image types. **acroread** is a subproduct of **acrobat;** run **`setup acrobat`**.

```
speed 9600 baud; line = 1;
intr = ^C; quit = ^; erase = DEL; kill = ^X; eof = ^D; eol = ^@; swtch = ^Z
lnext = ^V; werase = ^W; rprnt = ^R; flush = ^O; stop = ^S; start = ^Q
-parenb -parodd cs8 -cstopb hupcl cread clocal -loblk -tostop
-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr icrnl -iuclc
ixon ixany -ixoff
isig icanon -xcase echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3
%
```

In this display the second and third lines display the FUE default control characters. The character **^** indicates the control key (e.g., **^C** represents **<CTRL-C>**). Your reference books will most likely tell you to delete a character with the **#** key and delete a line with the @ key, **but this is not correct at Fermilab**. Use the character indicated as **ERASE** in the **stty** output for single character deletion, and **kill** for whole line deletion. The Fermi UNIX Environment defaults for these operations are the **DELETE** key and **<CTRL-X>**, respectively.

You can display a description of all of the options reported by **stty** with the command:

```
% man stty
```

If you don't like the FUE defaults, you can also set these functions with the **stty** command. The form for setting them is:

% stty ***control-char c***

where:

control-char	is one of the functions in the table in section 2.4.
c	is the representation of the key to be used for that function. A control character is specified preceded by a caret: **^x** represents **<CTRL-X>**.

Example:

```
% stty kill '^y'
```

There are two special representations: **^?** is interpreted as the **DELETE** key and **^-** is interpreted as undefined. You must include the quotes as shown in the example so that special characters are not interpreted incorrectly. You must be careful not to have two functions represented by the same key.

There are many other options that can be set with **stty**. Others that might be of interest are **echoe** which specifies that deleted characters are erased, and **-tabs** which specifies that the tab character be translated into the appropriate number of spaces. Refer to the man pages for more information.

Chapter 9: UPS Products and Applications

This chapter first discusses the Fermilab product support structure, and shows you how to get information about supported products (referred to as **ups** products) and applications. We then describe how to use available **ups** products, and how to obtain ones not yet installed on your system.

9.1 Product Support Overview

Computing Division products are supported under **ups** (**u**nix **p**roduct **s**upport). **ups** is part of the Core FUE environment (see Appendix B). A **ups** product is comprised of one or more applications, each of which is invoked by its associated command or commands[1]. **ups** maintains a central information repository for the supported products installed on a system. It also provides a uniform user interface allowing you to:

- view information about each **ups** product and its instances[2]
- access the available **ups** products and their associated applications

upd (**u**nix **p**roduct **d**istribution) is a companion utility to **ups**. It is used to distribute **ups** products stored as tar files on the **upd** host (UPD.FNAL.GOV) to a **ups** database on a remote node. **upd** is part of the Full FUE environment (see Appendix B). **ups** products are also available for distribution via anonymous **ftp**.

The complete list of **ups** products and supported applications managed by these two utilities is maintained in a separate set of databases accessible for reference via WWW, described in section 9.2. Documents related to each application are also available here for viewing. An on-line report is available that lists the current version of all the products; see **Products (KITS) Status Report** from the WWW *Computing Division home page*.

Off-site machines need to register for the product distribution services, **upd** and anonymous **ftp**. To do this, the responsible party for the machine needs to fill out the *UNIX Product Distribution Request Form* and mail it to *compdiv@fnal*. To find this form, go to the WWW *Computing Division home page*, and under the heading **Services** click on *Request and Registration Forms*. Or look in the top level anonymous **ftp** directory.

You can obtain documentation for **ups** and **upd** from the Computing Division librarian or on WWW. An overview of the usage of **ups** and **upd** is described in GU0006, *Overview of Product Installation and Distribution on UNIX Machines*.

---

1. Most applications have a single invoking command. However some applications have a set of commands, each of which invokes a different function.
2. A product *instance* is a unique combination of product name, version, flavor, and **ups** database. *Flavor* indicates the vendor platform on which the software runs, and often the software version as well.

9.2 The UNIX Software Database

The ever-changing list of software provided and/or supported by the Computing Division is maintained in an Oracle database accessible via WWW. On the Computing Division home page, look for *General Documentation and information about offline applications* under the heading **Information Resources**. Here you will find information on each **ups** product and supported application which includes product/application name, product number[1], and an abstract describing the product. In addition, text and/or PostScript versions of documents associated with the products are provided for viewing where they are available.

At the time of this writing, the database is undergoing enhancements. A keyword search facility is provided, which currently allows a one-word entry. Functionality to allow more flexibility in searching the database is planned.

We also plan to provide in the near future a facility to view the **ups** product-application relationships. As mentioned earlier, many of the **ups** products are comprised of several individual applications, each of which may require a unique invoking command. As a further step, we plan to extend the facility to allow cross-referencing at three levels: **ups** products, applications, and both setup and invoking commands.

As updates to the database are made, they'll be posted at the top of the Computing Division home page. Keep an eye out!

9.3 Accessing and Using Products

In the Fermi UNIX Environment, application products are supported under the **ups** system, which provides a uniform interface between products on the system and you, the user. You can find out what **ups** products are installed on your system with the command:

```
% ls $PRODUCTS
```

Note the capital letters.

To make a **ups** product available for your use, you have to run a **`setup`** command. To set up a product, enter:

`% setup` ***product***

where ***product*** is the **ups** product name as listed under the **`ls $PRODUCTS`** command. This makes available all the applications associated with that **ups** product. Some applications may also be set up individually. This command by default sets up the current version of the product; it is equivalent to:

`% setup -c` ***product***

The actions performed by a **ups** product setup depend on the requirements of the particular product; typically it sets the aliases and environment variables the product needs in order to run, and adds the product to the execution *PATH* variable. Each **ups** product creates a product directory and an associated environment variable ***PRODUCT*****`_DIR`**, where ***PRODUCT*** is the **ups** product name in upper case. You can see the contents of the product directory by running:

`% ls $`*`PRODUCT`*`_DIR`

1. This is an alphanumeric designation assigned by the Computing Division to identify each product and its associated documents.

You can also *unsetup* a **ups** product. **`unsetup`** unsets the aliases and environment variables that the **`setup`** command defined. You may need to unsetup a **ups** product if you are short on environment variable space and want to get rid of extra environment variables or shorten the *PATH* variable length. To do so, enter:

`% unsetup` ***product***

To find the current version of a **ups** product and its location, use the command:

`% ups list [-a] [`*product*`]`

More than one version, or instance, of a **ups** product may be installed on your machine. If **`-a`** is specified, all instances of the product(s) are listed. If ***product*** is omitted, all **ups** products are listed.

setup can tell if a **ups** product has already been set up. If the specified product has already been set up, it runs **unsetup** first to clean up before running **setup**.

We recommend that you *not* specify **ups** products by their version, but rather use primarily the *current* version for production, the *test* version (or new or development) for testing, or an *old* version *only if necessary.* We cannot maintain all versions forever, and unless there are overwhelming reasons, old versions will be removed in a timely manner. When it is necessary to run a specific version of a **ups** product, enter the **`setup`** command in the format:

`% setup` ***product version***

where ***version*** is the version number as declared in the **ups** database (shown from the **`ups list`** command). For example, to explicitly set up the non-current version 5.1 of the **futil** product, you would enter:

`% setup futil v5_1`

To set up a version which has been declared to the database as being in the test phase, use the format:

`% setup -t` ***product***

There are many other options available to **setup** to obtain specific flavors of software. For more information see the man page for **setup** or the *UPS User's Guide*. Most **ups** products will install appropriate man pages for their associated applications in a specified directory when the product is declared as current to **ups**. Once this is done, typing **`man`** ***application_name*** or **`man`** ***command*** will provide the available help.

Note that you must know the name of the application or its invoking command, **which may be different from the ups product name**. To get a listing of the applications within a **ups** product, use the command:

`% ls $`*`PRODUCT`***`_DIR/bin`**

9.4 Obtaining Products

As a user, you should be able to rely largely on your system administrators to install **ups** products for you. Occasionally you may need a product that has not been installed on your machine.

9.4.1 upd

Products are distributed with **upd** (**u**nix **p**roduct **d**istribution) from the platform UPD.FNAL.GOV. The UPD.FNAL.GOV platform maintains tar files of the currently available versions of all supported software products for the supported platforms. When **upd** is installed on your machine, you can use it to list the available **ups** products, list the contents of a tar file, copy a product or copy an individual file from a **ups** product tar file, and unwind a tar file.

Recall that off-site systems must register for this service.

If you want to make a **ups** product available for general use on your system, you should have it added to your local system's **ups** database. Contact your system administrator in this case.

To see what **ups** products are available for distribution, enter:

```
% setup upd
```

```
% upd
```

The following menu will appear. Enter **1** to get product information.

```
Welcome to the Menu interface for upd - choose one of the following options:

1 upd list: list available products
2 upd copy: copy a product from a remote node
3 upd unwind: unwind product into local directory
4 upd tarcont: list the table of contents of the tar file
5 upd extract: extract one or more files from tar file
6 upd get req: get the build and use requirements
7 ups declare: declare the product to UPS
8 ups tailor: execute products ups/tailor file
9 fork a shell: spawns a login shell

q quit
Enter choice : 1
```

This brings you to the menu printed below. The node name for the current general distribution platform is UPD.FNAL.GOV, so initially enter **1** at the `Enter choice :` prompt, and respond to `Enter node name [ ]` with **upd.fnal.gov**, as shown (you may drop the **.fnal.gov** portion of the node name and just enter **upd**):

```
UPD LIST: * implies must be set
 *1 [] node name
  2 [] product name
  3 [] instance
  4 [] flavor
  5 [] remote database

e execute upd list
 q quit this menu
Enter choice : 1
Enter node name []: upd.fnal.gov
```

Press return, and the menu will redisplay with your node name choice filled in. Continue with the other options until you have narrowed the selection to your satisfaction. Here we continue with option **3**, product instance, and specify **-c** to request the current version[1]:

```
UPD LIST: * implies must be set
 *1 [upd.fnal.gov] node name
  2 [] product name
  3 [] instance
  4 [] flavor
  5 [] remote database
e execute upd list
 q quit this menu
Enter choice : 3
Enter instance []: -c
```

For option **4**, the basic flavors currently supported are AIX, IRIX, SunOS, and OSF1. Your system's basic flavor can be found by the command **funame -s**. There are also products supported for *extended* flavors that take into account the OS type and release. The format of an extended flavor is the basic flavor followed by a **+** followed by the release number (e.g., IRIX+5)[2]. You can specify **IRIX+5** as the flavor in the **upd** menu or specify **"IRIX*"** (note that the quotes are required) to see both the basic and extended IRIX flavored products. This latter form is only useful when running `upd list`.

If you do not specify the **ups** product name (option **2**), **upd** will list all **ups** products which satisfy the other criteria (this may be a very long list).

When you're ready, enter **e** to execute the list.

From the list, you can decide what product, flavor, and instance you want, and execute further steps from the top level menu, e.g., **copy**, **unwind**. If you run **copy**, you must specify the flavor, otherwise you will get the flavor of UPD.FNAL.GOV's operating system by default.

Note, you do not have to bring over the complete tar file. You can use **upd tarcont** to list the contents of the tar file and **upd extract** to retrieve just what you want.

9.4.2 Anonymous ftp

ups products are also available via anonymous **ftp** from FTP.FNAL.GOV.

If you have a local **ups** database, we recommend distributing via **upd** because it provides a better interface for declaring products once the tarfile is copied. You can always invoke **upd** to do this after using **ftp**, of course.

To access anonymous **ftp**, enter the command:

```
% ftp ftp.fnal.gov
```

You will get the following response, to which you enter **anonymous** at the `Name` prompt:

```
Connected to fsui01.fnal.gov.
220 fsui01 FTP server (Version wu-2.4(2) Thu Jul 20 12:03:52 CDT 1995) ready.
Name (ftp.fnal.gov:your_username): anonymous
```

---

1. You may need to specify **current** instead of **-c**, depending on your system.
2. **funame -r** returns the full release number of the operating system. Normally you only have to specify the first digit. For example, if the command returns `5.2`, you can specify either **IRIX+5** or **IRIX+5.2**.

Once that is accepted, you will be prompted to enter your email address as a password, which is not echoed to the screen:

```
331 Guest login ok, send your complete e-mail address as password.
Password:
```

The screen will then display the following information:

```
230-This is the FTP service for FTP.FNAL.GOV. Use of this
230-service is for authorized usage only.
230-
230-The purpose of this service is to provide access to the
230-following categories of files:
230- - The FermiTools area: publicly available software under the
230-   /pub directory (all users). Files available listed in ls-lR.Z.
230- - The UNIX KITS area: This provides alternate access to files
230-   normally accessed via the upd program. (registered users)
230-   Files available listed in index.
230-
230-This server allows for on-the-fly compression  and uncompression
230-of files retrieved. It also allows you to tar up entire directory
230-structures.
230-
230-You are registered to retrieve files from the UNIX kits area.
230-
230-Please read the file readme
230-  it was last modified on Wed Sep 27 16:47:32 1995 - 111 days ago
230 Guest login ok, access restrictions apply.
```

Refer to section 12.1.1 for information on standard **ftp** commands.

A `readme` file is provided in the top level directory that discusses some of the available tools and features. **ups** products are under `/KITS/`. Access to this directory is restricted to registered users.

Recall that all machines in the `.fnal.gov` domain are automatically registered, but off-site users must register their machines using the UNIX Product Distribution Registration Form. You'll find this form in this top level directory under the filename `registration`.

Chapter 10: Editors

Several text editors are available at Fermilab. In this chapter we present our view of the advantages and disadvantages of the available editors, and we provide some basic information on the setup and use of each one. You will learn how to invoke each editor, and how to create, edit, and save a file in each one using a small subset of commands and features. We include only minimal usage information for the VMS-style editors.

10.1 The Available Editors

vi	A native UNIX screen editor. It is not generally considered to be one of the better available editors, however knowing the basics is useful for two reasons: 1) you may occasionally encounter an application that throws you into a **vi** session, from which you'll at least want to exit, and 2) **vi** is the one editor you are guaranteed to find on all UNIX machines[1].
emacs	A powerful modern public-domain screen editor available for both VMS and UNIX. You will find **emacs** installed on most UNIX machines.
NEdit	A popular and intuitive X-based editor written at Fermilab for UNIX.

The remaining available editors are UNIX implementations of Digital's products for VMS:

nu/TPU	An emulation of TPU
fermitpu	A locally ported UNIX version of the newest EVE (from VMS V6.1)
EDT+	A UNIX version of EDT

The VMS-style editors are legacy products, and as such are not guaranteed to remain available beyond the VMS-to-UNIX migration period. It is also unlikely that you will find these third party editors on UNIX systems outside of Fermilab. They are provided solely to ease the transition to UNIX for users already familiar with these products on VMS. **Users not already familiar with these VMS editors are strongly discouraged from choosing one of them for use in UNIX!** *All* users are encouraged to begin learning one of the permanent UNIX editors.

10.2 Comparison of Editors

The following table discusses the advantages and disadvantages of each of the above-mentioned editors.

---

1. In addition, UNIX provides **sed** for non-interactive editing and **awk** for more sophisticated and complex non-interactive editing. These two products are not covered in this manual.

Editor	Advantages	Disadvantages
vi	Available on all supported platforms and systems as a native UNIX tool. Well-documented. It is set as the default editor for many applications on many systems.	Considered to be very primitive in its screen capabilities. Non-intuitive interface. Lowest common denominator of screen editors.
emacs	Widely available public domain software package. Well documented. Many macros and enhancements available via the internet. Lots of expertise and consulting available via newsgroups, etc. Very configurable. **EDT**-style keypad emulation available. X and ASCII modes. Supports file versions.	Requires installation of **emacs**. Interface not very intuitive for users accustomed to VMS-style editors.
NEdit	Very intuitive and well-documented. Customizable. Gaining popularity world-wide. X-based.	Requires installation of **NEdit**. Requires X-based terminal. Not available on all systems/platforms.
nu/TPU	Nearly identical to Digital's **TPU** for people familiar with **TPU** programming. Sufficient for a wide variety of simple editing tasks. Close enough to **EVE/EDT**-style keypad to be familiar to VMS users. Does not exhibit screen-repainting problem of **fermitpu** (see below).	Duration of availability is uncertain. Implementation includes many faults. Limited documentation available on-line. Uses old version of **EVE** (from VMS V5.0). Does not match the interface that VMS users are used to. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.
fermitpu	Nearly identical to Digital's **EVE** of VMS V6.1 with the exception that the **EDT**-style keypad is used by default. Licensing is handled on the local node (i.e., no network license server problems). User-configurable and extendable. Does not require X.	Duration of availability is uncertain. Not available/installed on all systems/platforms. Not the same as **fermitpu** on the central facility VMS systems, although similar. Requires the FUE environment and the installation of **nu/TPU** and the **fermitpu** layered products. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings. On some platforms, the screen is repainted after every keystroke which is very slow.

Editor	Advantages	Disadvantages
EDT+	Very similar to an enhanced **EDT** for people familiar with **EDT** programming. Incorporates a powerful word processor, disaster recovery system, an extensive help facility, full feature programmable text processing and the GOLD-KEY style of editing. Does not require X, although X interface is available. Minimum learning curve for users familiar with Digital's **EDT**.	Duration of availability is uncertain. Requires the FUE environment and the installation of the **EDT** product. Not available/installed on all systems/platforms. Distributed licensing (implies that network problems and/or problems with the license server could prevent **EDT+** from being usable on occasion). Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.

Note: For all editors used with X display, the *DISPLAY* variable (see section 8.2) must be set properly.

10.3 Getting Started with the Editors

In this section we present a few important commands for each editor. The minimal information necessary for you to edit and save a simple file is provided. All these editors have many commands and sophisticated features that we do not cover here.

10.3.1 vi

As we mentioned in section 10.1 above, you may occasionally find yourself thrown into the **vi** editor unexpectedly, and you will certainly want to know how to exit, if nothing else. For that reason alone you should become familiar with a few **vi** commands. As is typical in UNIX, the **vi** commands are case sensitive.

vi requires no setup. Invoke it using the command:

`% vi [`*filename*`]`

Once the file is opened, you are in *command mode*. From this mode you can issue commands, move the cursor, and invoke insert mode.

To enter *insert mode*, type **`i`**. Text you type will be inserted *before* the cursor. From insert mode you can enter new text in the file. Press the **ESCAPE** key to exit insert mode and return to command mode. On many keyboards the **ESCAPE** key is labelled; if not, the sequence **<CTRL-[>** is mapped to the escape function.

Some useful vi commands available in command mode

`h,j,k,l`	move cursor left, down, up, right, respectively
`H`	move to top line of screen
`L`	move to bottom line of screen
<CTRL-F>, <CTRL-B>	scroll forward, backward one screen
/pattern	search for ***pattern***
`/`	repeat search in forward direction
`x`	delete current cursor position

x	delete back one character
dw	delete current word
dd	delete current line
***n*dd**	delete ***n*** lines starting with current
p	insert (**p**aste) last deleted text after cursor
:r ***filename***	read in contents of ***filename*** after cursor
:x	quit **vi**, writing file only if changes were made
:w	write file, do not quit
:w ***file***	save copy to ***file***, do not quit
:q!	quit file, discarding edits

Most UNIX guides contain a complete description of **vi**.

10.3.2 emacs

emacs is a popular editor on the net. Many UNIX books cover **emacs**, and a good reference for **GNU emacs** is *Learning GNU Emacs* (O'Reilly & Associates). We also refer you to an on-line GNU EMACS manual accessible from the UNIX Resources WWW page.

emacs can be set to have an **EDT**-style keypad.

To use the editor, the **emacs** product needs to be installed. To set it up, include in your login script or enter:

```
% setup emacs
```

The mode (X or ascii) in which **emacs** attempts to startup is determined according to your *DISPLAY* variable. To invoke, type:

```
% emacs [options] [filename]
```

To invoke it in ascii mode without a new window using an X terminal, type:

```
% emacs -nw [options] [filename]
```

emacs has an extensive interactive help facility, but the facility assumes that you know how to manipulate **emacs** windows and buffers. **<CTRL-H>** enters the Help facility. **<CTRL-H>-T** enters the help tutorial, which can teach beginners the fundamentals of **emacs** in a few minutes. **<CTRL-H>-A** enters Help Apropos, to help you find commands by function. **emacs** commands use the *Control key* and the *Meta key*[1]. In the following list **C-** indicates that the control key is pressed at the same time as the character that follows. Similarly, **M-** indicates the use of the Meta key, although it's not necessary to keep the Meta key pressed down while typing the next character. Note that some command sequences use multiple keystrokes, with and without the Control and Meta keys. A sequence like **C-x u** means hold down control while you press **x**, then just press **u**.

Important emacs commands

C-h	enter on-line help system

1. If you have a key labelled **META**, use it; if you don't, try the **ALT** key or the **ESCAPE** key (if you're running a native X window). As a last resort, the sequence **<CTRL-[>** should always work as a Meta key.

C–h i	enter the information browser which provides a menu of major topics (use **TAB** and **ENTER** keys or 2nd mouse button to select a topic; navigation information is provided)
C–p, **C–n**, **C–f**, **C–b**	move up (to **p**revious), down (to **n**ext), **f**orward, or **b**ackward by one line or character, respectively
C–v, M–v	move forward, backward, by one screen
C–s	search forward for characters (system will prompt you for string). To continue search, type **C–s** again.
C–r	search backward
C–d	delete a character
C–k	delete (kill) from cursor to end of line
C–y	restore what you've deleted
C–x u	undo last edit
C–g	get out of current command operation
C–x i	insert file at cursor position (system will prompt for filename)
M–q	fill paragraphs
C–@ or **C–SPACEBAR**	set the mark for the start or end of a region to select
C–w	delete all between mark (see **C–@**) and cursor's current position (paste back with **C–y**)
M–w	copy all between mark (see **C–@**) and cursor's current position (paste back with **C–y**)
C–x C–x	exchange mark and cursor's current position (since the mark is invisible, this allows you to find it)
C–x C–s	save the file
C–x C–w	write to file (system will prompt for filename)
C–x C–b	display buffer list
C–x o	move cursor to other window (when more than one displayed)
C–x C–c	exit emacs

Note, if the serial port or terminal device you are typing on is configured for **<CTRL-S>/<CTRL-Q>** flow control, you may find that **<CTRL-S>** (written above as **C–s**) within **emacs** causes the terminal to stop sending characters, the same as when used at the shell prompt. If you want to use the usual **emacs** key bindings and to have **C–s** work properly within **emacs**, you'll need to reconfigure your line to not do flow control. Where and how you do this will depend on how you're connected. If you're connected via a modem, you may need to reconfigure your modem, as well as the pseudo-terminal on your UNIX host. The latter can be done via the command **stty stop undef start undef** (which sets the stop and start characters to "undefined"). You could include this statement in your `.login` or `.profile`. The intermediate step, between the on-site modem and the Cisco router/terminal server, has flow control turned off by default.

Further documentation can be found from the man pages. Look under **Editors** on the *UNIX Resources* WWW page for documentation on the **EDT** emulation package and the **TPU-edt** editor for GNU Emacs.

10.3.3 NEdit

NEdit is very similar in appearance and operation to many PC text processing applications for Windows. It is menu/mouse driven, with keyboard shortcuts available. Some UNIX shell commands are available from within the editor. Make sure the **NEdit** product is installed on your system. To set up **NEdit**, include in your login script or enter:

```
% setup nedit
```

To invoke **NEdit**, type:

```
% nedit [filename]
```

Due to the user-friendly nature of the product, we present only a few basic commands to give you a flavor of this type of editor if you are not familiar with it:

Open a file	**`Open`** from the File menu; choose an existing file from popup window
Create a new file	**`New`** from the File menu
Include a file	**`Include`** from the File menu
Select text	Use the mouse, or the shift and arrow keys together
Cut/Paste text	**`Cut/Paste`** from the Edit menu
Search for text	**`Find`** from the Search menu
Fill paragraph	**`Fill Paragraph`** from the Edit menu
Spell check	**`spell`** from the Shell menu
Save the file	**`Save`** from the File menu
Close the file	**`Close`** from the File menu (you are prompted about saving)
Exit **NEdit**	**`Exit`** from the File menu

Further information is available from the man pages and a plain text document in the distribution kit, but the on-line help in the program is complete and more convenient. The documentation is also available from the Computing Division Library.

10.3.4 nu/TPU

nu/TPU is an excellent port of Digital's TPU (Text Processing Utility) programming language to UNIX. The major component in **nu/TPU** is software that can be used to build new text processors and batch-oriented text manipulation routines. **nu/TPU** is distributed with two editing interfaces:

- SI (Simple Interface), an **EDT**-style keypad map. This should be sufficient for users who do minimal editing and wish to use an **EDT**-style keypad.
- a clone of the old **EVE** interface (from VMS V5.0) complete with the EVE_functions (**but not the EVE$functions!**) from that version. This will probably not be satisfactory to users familiar with the later version of **EVE** (VMS V6.1), since it lacks many of the enhancements and fault corrections.

The product **tpu** needs to be installed prior to use. To set up **nu/TPU,** include in your login script or enter:

```
% setup tpu
```

To edit using the SI interface by default, type:

```
% tpu [filename]
```

To edit using the older **EVE** interface, enter:

```
% tpu -section=tpusec.ini [filename]
```

Once you are in the editor, on-line help is available via the **PF2** key or the Help command.

For complete documentation on **nu/TPU**, including documentation on all command-line options and all **nu/TPU** programming statements, you can order the *nu/TPU Reference Manual.* Look under **Editors** on the *UNIX Resources* WWW page for documentation on the **TPU-edt** editor for GNU Emacs.

10.3.5 fermitpu

The **fermitpu** product contains a locally developed port of Digital's **EVE** editing interface from VMS V6.1 (the version of VMS which is current during the VMS to UNIX migration). Note that this is not the same as the VMS product **fermitpu**, which contains many enhancements beyond the VMS V6.1 **EVE** editor. Users who are familiar with programming in **EVE** and **TPU** will probably be most comfortable with this editing interface.

The product **fermitpu** needs to be installed prior to use. To invoke, include in your login script or enter:

```
% setup fermitpu
```

```
% tpu [options] [filename]
```

Once you are in **fermitpu**, the on-line help (accessible via **PF2** or the Help command) contains documentation on all of the EVE callbacks and TPU built-ins.

Look under **Editors** on the *UNIX Resources* WWW page for documentation on the **TPU-edt** editor for GNU Emacs.

10.3.6 EDT+

This is a clone of Digital's **EDT** which includes the enhancement to remove the hard-coded 24-line display limit. The product **edt** needs to be installed. To setup **EDT+**, include in your login script or enter:

```
% setup edt
```

To invoke for editing within the same terminal window, type:

```
% edt [options] [filename]
```

To edit in a separate display window (requires X), type:

```
% xedt [options] [filename]
```

EDT+ allows both line mode and full screen editing. Full screen editing employs a keypad, which is located on the right-hand side of your keyboard. If you are in line mode when you first enter **EDT+**, simply type **`c`** to change to full screen mode. Once in full screen mode, you can start entering text, and your keypad is available for commands.

To return to line mode, enter **<CTRL-Z>**. At the prompt (**`*`**) you can type **`exit`** to exit and save, or **`quit`** which exits without saving the file updates.

Remember that when you're at the UNIX command prompt **<CTRL-Z>** is used as the suspend character.

Further documentation is available from the man pages, and from the vendor Boston Business Computing, Ltd. Look under **Editors** on the *UNIX Resources* WWW page for documentation on the **EDT** emulation package for GNU Emacs.

Chapter 11: UNIX Mail Systems

This chapter is designed to introduce you to the available mail handling systems at Fermilab. We concentrate on the UNIX **MH** (Message Handling) system which the Computing Division recommends and fully supports. Fermilab-supported **MH** products include both the standard line mode mail reader (referred to as **mh**), and a GUI version of **MH** called **exmh**. In this chapter, our goal is to get you up and running in **mh** or **exmh** as quickly and easily as possible, with a minimum of up-front configuration.

The Computing Division has recently begun supporting an additional non-graphical UNIX mail utility, **pine**. **pine** functionality is not documented in the present manual, but we provide some start-up information for you in section 11.2.3 and point out where you can get more information.

You'll also find a brief discussion of **Berkeley mail** at the end of the chapter, only because it is a mail reader available on all UNIX machines.

If you choose **mh** or **exmh** as your mail reader, you will undoubtedly want to customize some features soon after you're comfortable with the basics. Appendix F contains information on how to customize **exmh** and **mh**, and how to take advantage of some of the less essential but very useful features.

If you wish to do more in-depth customization of the **MH** products, we recommend the excellent reference book *MH and xmh - Email for Users & Programmers* by Jerry Peek, published by O'Reilly & Associates. This book is now on-line and can be accessed through the *UNIX Resources* web page.

Appendix G provides a quick reference for the **mh** commands described in this chapter.

Appendix H contains instructions on moving your VMS mail folders to UNIX and converting them to the **MH** system format and organization.

11.1 Mail Forwarding

11.1.1 The Fermilab Mail Server: FNAL

First, a general word on mail handling at Fermilab. FNAL formerly referred to the central Fermilab VAX computing cluster, where most people at the lab had an account. As computing became more distributed, a single mail serving node was set up to handle the mail flow. This mail server was designated as FNAL, and the central VAX cluster was renamed to FNALV. FNAL will continue to function as the Fermilab mail server.

The central mail server provides a simple, uniform mail address for all users at the lab:

***username*`@fnal.gov`**

We strongly recommend that you take advantage of this capability and have all your mail routed through FNAL. It is your responsibility to choose a node for reading mail and to make sure your forwarding address on FNAL is set to this node. We step you through the forwarding process in

section 11.1.3. However, we recommend that you work your way through the material in this chapter in the order it is presented, and change your forwarding address on FNAL only when you're ready to actually switch your mail activities to your chosen mail system on UNIX.

11.1.2 Forwarding on File-Sharing UNIX "Clusters"

Before getting started, you will need to choose a node for your mail activities. Once you do, forwarding mail to an **MH** system requires creating a simple `.forward` file in your `$HOME` directory. We show you how to do this in section 11.3 and again in F.1. If your chosen mail node is part of a UNIX "cluster" like FNALU on which a file-sharing system is installed (for example AFS or NFS), then creating a single `.forward` file in your common `$HOME` directory is sufficient for the entire cluster. The contents of this file indicates your chosen mail node within the cluster.

You might wonder why you need to choose a particular node in the case of a file-sharing cluster. The reason is that incoming mail on UNIX is sent to a non-shared area of *one* node; you can't change that, you can only specify *which* node you want used. From here the mail needs to be incorporated into the mail system of your choice; we cover this topic in sections 11.3.3, 11.4.3, and F.3. If you are working on a different node from the one receiving the incoming mail, you will be able to send mail, but you will only see previously incorporated mail; you will not receive (and cannot incorporate) new mail because it is inaccessible to this node.

11.1.3 Recommended Forwarding Procedure

Advantages

The procedure we describe here for setting your mail forwarding has three advantages:

- Every time you change your mail node, you only need to change the address on three nodes: FNAL, the old (or current) mail node, and the new mail node. If you have accounts on many systems, this is a timesaver.
- All Fermilab users have a standard, stable email address: ***username*`@fnal.gov`**.
- No mail is left sitting in the system mailbox (except as yet unincorporated mail on your chosen mail node).

Set Forwarding on your New Mail Node

As a first step, in your `$HOME` directory on your chosen UNIX mail node, create a `.forward` file using your preferred editor. Enter the following expression as the contents of this file:

`\`*username*`@`*node*`.fnal.gov`

where ***username*** is your login id on your chosen mail node, ***node***. For example, user **`fred`** who wants to receive mail on the FSUI01 node of the FNALU cluster would enter the line:

`\fred@fsui01.fnal.gov`

This expression is an instruction to put incoming messages in this designated node's spool area. The initial backslash (****) is required to prevent looping.

Later, you may decide to set up mail notification and/or unattended autoincorporation on your chosen mail node. These options are discussed in Appendix F. If so, you will need to replace the above expression with a different one in the `.forward` file on this node (or cluster) in order to prevent the redirection of incoming messages to two or more places.

Set Forwarding on FNAL

As a second step, you need to log on to the mailserver node FNAL and use the menu-driven program to change the forwarding to your new mail node. If you don't know or don't remember your FNAL password, send mail to *compdiv@fnal* and request a new one.

Make sure you set forwarding here on FNAL to your new mail node BEFORE you change the forwarding on your old mail node to FNAL! Otherwise you get an infinite mail loop.

Once you're logged on to FNAL, you'll see the following menu:

```
                              VMS Menu System V3.0
User: USERNAME
Date: 11/10/95
Forwarding Address:   username@FNALV.fnal.gov

Fermilab Mail Server Menu
    F  Set Forwarding
    P  Set Password
    U  Unsubscribe from a LISTSERV
    X  Exit the Menu


Time: 12:33 PM
Select Option:
```

Select option `F Set Forwarding`. This brings you to the following screen:

```
Please enter your new forwarding address below

 Supported addressing formats

 DECnet     Node::User-ID                        FNALV::JONES
 BITNET     User-ID@Node                         JONES@FNALV
 INTERNET   User-ID@Node.domain                  JONES@FNALV.FNAL.GOV
 Quickmail  User-ID.Mailcenter@QMGATE.FNAL.GOV   JONES.Mailcenter@QMGATE.FNAL.GOV



 Please do not enter quotes or standard prefixes (smtp%, JNET%, IN%...)


 Your current forwarding address is:     username@FNALV.FNAL.GOV


 Enter forwarding address [ ? for help]:
```

Set your new forwarding address here in the format:

***username*@*node*`.fnal.gov`**

where ***username*** is your login id on your chosen mail node, ***node***.

Set Forwarding on all Other Nodes (including old mail node)

On all other external systems where you might receive mail (excluding the cluster containing your new mail node), UNIX or otherwise, set your forwarding address to your FNAL address.

To do this on other UNIX systems, create a `.forward` file in your `$HOME` directory on each separate node or file-sharing cluster. The contents of this file should be a single line containing the expression:

***username*@fnal.gov**

where ***username*** is your login id on FNAL, the mail server. Note that there is no initial backslash here.

For information on how to set forwarding on non-UNIX machines, consult your system-specific documentation.

11.2 Overview of Mail Systems Available at Fermilab

As is typical for UNIX, there are many mail handling systems from which to choose. Here we provide a brief introduction to each of the available mail handlers at Fermilab. The Computing Division has chosen to provide and support the **MH** system (which encompasses **exmh** and **mh**) and **pine** as the recommended solutions.

The simpler, less functional **Berkeley mail** is a mail reader that you will find on all UNIX machines. Therefore we give a cursory treatment of it in this chapter, although we explicitly do not recommend it.

> First choose one UNIX node for your mail activities.
>
> Secondly, we advise you to become familiar with the different mail systems and select the one you want to use as your primary mail handler *before* you start your conversion process. Although **MH** and **pine** can be made to work together (see section 11.2.3), they were designed independently of each other, and by default organize and handle mail differently.

11.2.1 MH Graphical Interface: exmh

exmh is the GUI implementation of **MH**, requiring the X environment. It is built with **Tcl/Tk**. We recommend this as your mail handler if you most often work in the X environment. It is quite customizable, yet for the most part satisfactory with a minimum of configuration. **exmh** provides interfaces to many other powerful tools including a spell-checker, a fast-search mechanism, WWW, and more. **exmh** can read and send MIME (Multipurpose Internet Mail Extensions) multimedia mail.[1] Note that **exmh** displays nonprinting characters as their ASCII hex value. For example, embedded line-feeds (often included in mail messages originating from a PC) are shown as `\x0d`.

11.2.2 MH Line-Mode Interface: mh

mh is the command line implementation of the **MH** system, and as such is suitable for occasional use (when X-capability is unavailable) by those whose primary mail handler is **exmh**. **pine** may also be used in this capacity (see section 11.2.3).

mh is a very rich, flexible, and configurable mail system on its own. **MH** commands are not a part of a monolithic mail environment; each command is a separate program and the commands are issued directly from a UNIX shell prompt. Therefore, all the features of UNIX shells (pipes, re-direction of I/O, and so on) work with **mh**. You can use the man pages to get information on the **MH** commands.

---

1. MIME allows you to read and send image files, sound files, and other items that require encoding to be sent via the mail system. The O'Reilly book referenced at the beginning of the chapter includes an introduction to MIME.

11.2.3 pine

pine (Program for Internet NEws and mail) is a popular menu-driven, non-graphic UNIX mail system which the Computing Division has recently begun supporting. It can be configured to be compatible with the **MH** message storage structure, although it does not come this way "out-of-the-box".

pine is recommended for users who do not have access to X-capability. We have not documented **pine**'s functionality in this manual; however we provide some start-up information for you. You will need to select from the wide variety of available documentation on WWW and elsewhere to learn how to use **pine**.

As a starting point for documentation on **pine**, see the *UNIX Resources* page on WWW. You'll find information under the **Mail** heading. Or, also from the *UNIX Resources* page, select **Other UNIX Resources** followed by the *UNIX Reference Desk*, and click on *Applications* to find a good introductory document on **pine**.

Using pine and MH Interchangeably

If you have converted your mail to UNIX using the methods described in Appendices A and H, your mail folders are in **MH** format. See section 11.4.2 for more information on **MH**-style folders. If you plan to use **pine** and one of the **MH** interfaces interchangeably, you should leave your folders in **MH** format, and configure **pine** to recognize them. This involves creating appropriate mail folder collections. Follow these steps to make message storage in **pine** compatible with **MH**:

1) Run **`setup mh`** or **`setup exmh`** to create your **MH**-style `inbox` folder.

2) Run **`setup pine`**

3) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**. This creates the **pine** configuration file `$HOME/.pinerc`.

4) Exit **pine** (type **`q`** to quit).

5) Run the command **`fmh2pine`**[1]. This modifies the **pine** configuration, setting your **pine** inbox path to the default **MH** `inbox` folder, and adding an **MH** folder-collection containing all your **MH** folders. This command also makes your nested **MH** folders available to **pine**.

6) Run **`pine`** again.

Type **L** to select a folder to view. When you expand the folder collection lists and view the folders, you will find that the contents of your **MH** `inbox` folder is included in `INBOX` in **pine**'s `<mail/[]>` folder collection. Your other **MH** folders show up in the `<#mh/[]>` folder collection with their original names. Any nested **MH** folders you have defined will show up in separate folder collections according to their "parent" folders.

Anytime you add or delete an **MH** folder, run **`fmh2pine`** to update **pine**'s configuration accordingly.

Configuring pine (for use with or without MH)

You can further modify the configuration, if you like. Go to **pine**'s SETUP CONFIGURATION menu. There are many options available, and they come with on-line help. Some options are variables which take one or more values. To set a value for one of these variables, first type **`A`** to add, then enter the value. Or type **`C`** to change an existing value. Other options are in the form of a list of features that you can turn on or off individually. Enter **`X`** to enable a feature; leave

1. **`fmh2pine`** is part of **mailtools v2_2**.

unwanted features blank. We recommend that you configure your list to include at least the following (note that **fmh2pine** sets `folder-collections` for you to the two values shown here):

`user-domain=fnal.gov`	sets your "reply address" to *username*`@fnal.gov`
`folder-collections=mail/[]`	enables your basic set of **pine** mail folders
`folder-collections=#mh/[]`	enables your **mh** set of mail folders

The option list is rather lengthy, so we provide some suggestions for you. We have found the following features to be particularly useful. Explanations for them are in the on-line help.

```
auto-zoom-after-select
auto-unzoom-after-apply
enable-aggregate-command-set
enable-alternate-editor-cmd
enable-bounce-cmd
enable-flag-command
enable-full-headers-command
enable-jump-command
enable-suspend
enable-unix-pipe-cmd
enable-expanded-view-of-addressbooks
preserve-start-stop-characters
quell-user-lookup-in-passwd-file
```

Using pine Exclusively

If you choose to use **pine** exclusively, you will need your folders in the **pine** format. First, get ready to run **pine**:

1) Run **`setup pine`**

2) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**.

3) Complete the configuration as described in the above sections, however do not run **`fmh2pine`** and do not set `inbox-path` to `#mh/inbox`.

You can copy entire folders between the **pine** and **MH** collections as follows:

1) Go to the `FOLDER INDEX` screen for the desired folder.

2) Enter **`;`** (semi-colon) to issue the **pine** Select command.

3) Enter **`A`** to select All messages.

4) Enter **`A`** to Apply the command.

5) Enter **`S`** for Save.

6) Use **<CTRL-N>** or **<CTRL-P>** as necessary to choose the destination folder collection, then enter the new destination folder name. Or just type in a folder collection and name. Respond `Y` to create the new folder.

To use **pine** exclusively, you'll want to move any folders in the `<#mh/[]>` folder collection to the `<mail/[]>` folder collection.

Printing from pine

You will find it useful to set up printing from **pine**.

1) Go to **pine**'s SETUP PRINTER screen (enter **`S`** followed by **`P`**).

2) Move down to the `Personally selected print command` option. Enter **`S`** to

select this option.

3) Enter **A** to add a printer, then as prompted, enter the name of the printer (this name is for your reference only) and the full command for the printer. For example:

Printer: ***printer_name***

Command: **flpr -q*printer_name***

4) You may enter several printers. Enter **S** to select the printer you want to use.

5) Exit (**E**) and save changes.

11.2.4 Berkeley Mail

Berkeley Mail is an older mail system, and it is available on all UNIX systems. We do not recommend using it, but like the **vi** editor, knowing a little about it may come in handy sometimes! See section 11.5 for information on basic commands.

11.3 The exmh and mh Mail Handlers

In this section we provide a guide to the basic commands and operations in both **exmh** and **mh**. Be aware that many more features and functions are available in these products than you'll find documented here.We want to remind you that both **mh** and **exmh** are highly configurable interfaces to the powerful **MH** system. You can customize your personal setup to your heart's content, however we emphasize that neither **mh** nor **exmh** *requires* tweaking and tailoring beyond a very manageable minimal amount. We provide you with the information necessary to set up these interfaces for effective use in three places:

- We show you how to change fonts and editors in the basic command guide that follows in this section.
- In section 11.4 you'll find information on signature lines, mail aliases, and customizing the **exmh** folder display. We also provide information here on using **exmh** functions to incorporate mail.
- In Appendix F we discuss several nonessential but commonly tailored items, including customized mail notification (F.1.2) and message headers (F.2).

For the adventurous among you, get yourself a copy of the 700+ page O'Reilly text referenced at the beginning of the chapter, and customize away!

Recall that **exmh** and **mh** use the same underlying commands, although they have very different user interfaces. We've organized this section by function, showing you how to perform each selected function in both **exmh** and **mh**. This should make it easier to switch back and forth between the two interfaces if you sometimes find yourself at a graphics-capable terminal, and sometimes at an ASCII-only terminal.

We recommend that you perform couple of initial steps before starting to practice. These steps relate to mail forwarding, and we recommend them at this stage because of the way incoming mail is stored under UNIX (mail storage is discussed in section 11.4.3).

- Choose your mail node, and plan to do your practicing there.
- Create a file named `.forward` in your `$HOME` directory on this node containing the single line:

`\`*`username`*`@`*`node`*`.fnal.gov`

where *`username`* is your login id on your chosen mail node, *`node`*. This ensures that your "practice mail" will be available to you on the right node if you should need it later.

If you have an account on another machine, why don't you send a message or two to yourself on your chosen mail node before getting started so you have some messages to read? This isn't necessary, you can always send messages to yourself on your mail node.

11.3.1 Running Setup and Invoking the Application

exmh

`% setup exmh`

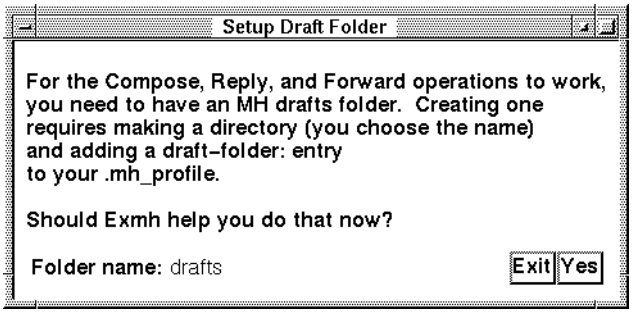
The use of **exmh** requires that you set your *DISPLAY* environment variable appropriately (see section 8.2).

When you setup **exmh**, you also setup several other products[1] which **exmh** either needs or with which it can interface. Among these is **mh**. If you've never run **`setup mh`**, then the first time you run **`setup exmh`** the file `.mh_profile` is created in your `$HOME` directory, and a message is sent to you (ignore the content of this message; it may be out of date). The `.mh_profile` file defines your basic **MH** configuration.

To invoke **exmh**, enter:

`% exmh [&]`

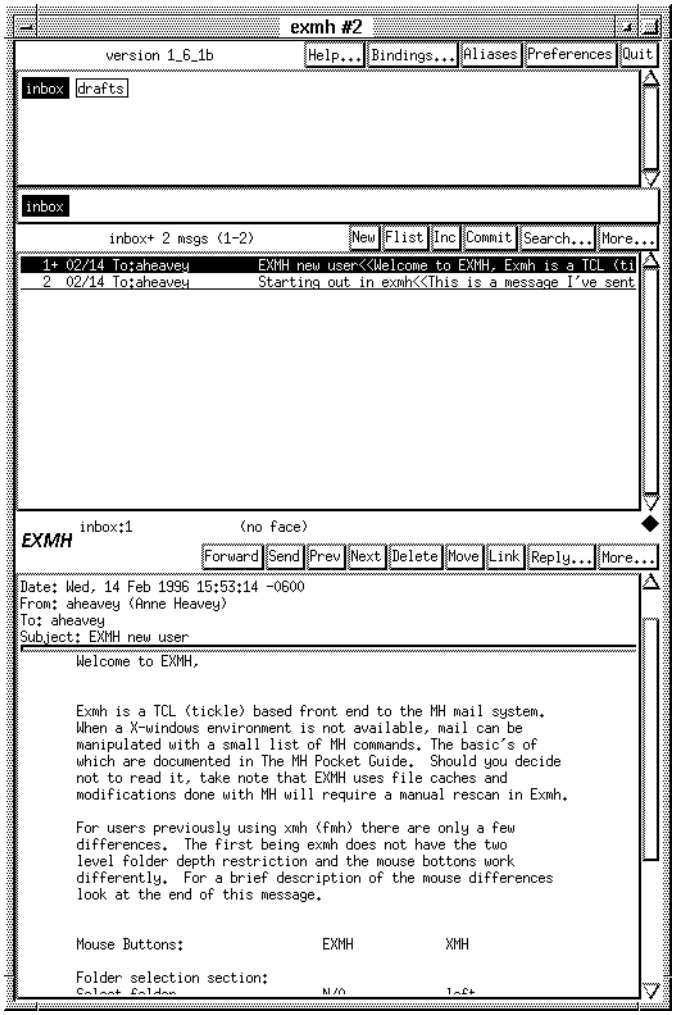
The first time you invoke **exmh**, you are asked if you want to create a message folder for message drafts in your `Mail` subdirectory. If you respond **YES**, it adds a line to your `.mh_profile` identifying the folder.[2] By default the folder is called `drafts`, but you are given the opportunity to name it as you wish.



1. Run the command **`ups list -l exmh`** to see the list of product dependencies.
2. It will also add the lines `unseen-sequence: unseen` and `repl:` to your `.mh_profile`.

Click on **YES** if you wish to continue.

Now an **exmh** window appears. The **exmh** window is split into three areas, which we also call windows. The title bars in each of the three windows have command or pull-down menu buttons. Actions are performed by clicking on the buttons and/or choosing from menu items.



The three windows have different functions:

Top	displays one "button" for every defined message folder. The current folder button is highlighted. Select a folder (make it current) by clicking on it with the left mouse button. **MH** message folders are discussed in section 11.4.2.
Middle	displays message headers in the current folder (the folder selected in the top box). The title bar for this area displays the current folder name and the number of messages. The header of the current message has a '+' next to its number and is highlighted.
Bottom	displays the current message (the message highlighted in the middle box). The title bar for this area displays the current folder name and the current message number.

A general usage note: You can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. To select multiple headers (useful for refiling, removing, etc. several messages at a time), hold down the shift key while you press the left mouse button to select each one.

You may find that on your terminal the default font size for **exmh** is rather small. If you really want to change it, here's how.

You'll need to include two lines at the top of your `$HOME/.exmh-defaults` file (above the three comment lines ending with: `!!! Do not edit below here`). You will need to choose which font specifications to use. The format of these two lines is as follows:

`*font:` *`new-font-specification`*

`*fl_font:` *`new-folder-label-font-specification`*

The `*font` line is for the text in the windows, and `*fl_font` is for the text in the folder label buttons. We recommend that you choose a fixed-width (as opposed to a proportionally spaced) font for text so that columns align properly in the middle (header) and bottom (message display) windows.

The font specifications from which you can choose may be platform-specific. Execute the command **`xlsfonts | less`** to see what fonts are available for you[a]. Note that some of the font specifications will look short (for example `vt33014`); they are font aliases, and will work here, too.

We suggest the following font specifications, if they are available on your system. Just add these two lines to your `.exmh-defaults` file as discussed above:

```
*font: -b&h-lucidatypewriter-medium-r-normal-sans-14-100-100-100-m-80-iso8859-1
*fl_font: -b&h-lucidabright-demibold-r-normal--14-100-100-100-p-84-iso8859-1
```

a. **xfontsel** is a point and click interface for selecting font names; you may want to use this in addition to **xlsfonts** to determine which fonts are available on your system.

mh

To setup **mh**, enter:

```
% setup mh
```

You don't need to do this if you've already setup **exmh**.

The first time you do this, it creates a file `.mh_profile` in your `$HOME` directory. It also automatically sends a message to you (ignore the content of this message; it may be out of date).

Now all the **mh** commands are available for use. As mentioned earlier, there is no single invoking command; you simply type **mh** commands at the shell prompt as you need them.

11.3.2 Composing and Sending Messages

Remember to send yourself a message or two so you'll have something to read later on.

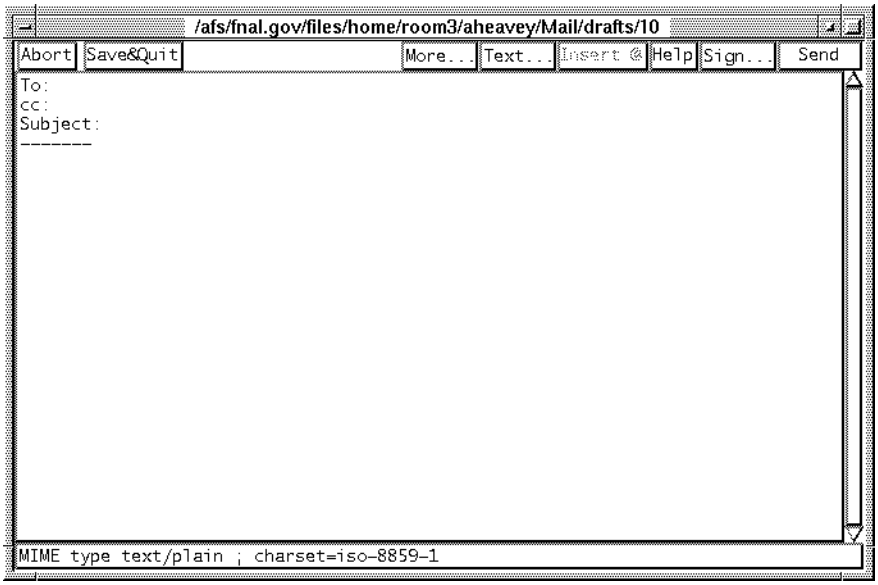
For both **exmh** and **mh**, a default message header with the following contents is provided for sending messages:

```
To:
cc:
Subject:
--------
```

If you wish to change the contents or appearance of the message header, you can create your own `components` file. See section F.2 for information.

exmh

To compose your message click on **SEND** in the bottom window. The word *comp* (for compose) will appear in the command menu bar. Then a new drafts window will pop up in which you will write your message. **exmh** uses **sedit** (Simple Editor) as the default editor for composing messages. As a new user, you may just want to use this editor for a while to simplify your initial setup.



Fill in the information, using a **TAB** between the header fields[1], and then type in the body of your message below the dashes. On many terminals **<CTRL-H>** is used for character deletion. Alternately, you can select the text to delete and press your **BACKSPACE** or **DELETE** key. Most mail programs expect a carriage return at the end of each line of the message, so it is always wise to include it.

Note that if you set your preferences (go to **PREFERENCES** button in top window) for **SIMPLE EDITOR** to Format Mail Default *On*, you don't need to explicitly include the carriage return at the end of each line, it will be done automatically for you after you exit the editor. This feature is only available for the **sedit** editor.[2] Once you finish in the "Simple Editor Preferences" window, click **DISMISS**. Then click **SAVE** on the "Exmh Preferences" window.

Once you have completed the message, you can send it to the recipient(s) by clicking on the **SEND** button. If you decide not to send it, you can either save it in the `Mail/drafts` folder by clicking on **SAVE&QUIT**, or discard the composition by choosing **ABORT**. There are many useful options available on the drafts window under **MORE...** such as **INSERT FILE** and **SPELL**. Under

---

1. Note, however, that if you tab twice in rapid succession, the cursor will move down to the message content area. You can use this feature to pass over all remaining fields if you don't want to enter information in them.
2. However, be aware that the line breaks inserted by **sedit** after you exit the editor may not be the places that *looked* like line breaks while you were in the editor. The line wrapping width is controlled by the value of **MAX LINE LENGTH** in the Simple Editor **PREFERENCES** window, not by the width of the editing window.

TEXT... you have some options for fonts, which only are useful if the destination machine/mail reader supports them. If not, the recipient will see things like `<bold>`*your message text*`</bold>` in the message.

If you don't want to use this editor, you can define a different default editor, or you can define a secondary editor and then select it from the **MORE** menu in the compose window (this secondary editor option is only available with **sedit** as primary editor).

To define a different default editor, select **EDITOR SUPPORT** under the **PREFERENCES** menu option in the top window. A pop-up window will appear. If your preferred editor "knows how to" create its own X window, e.g., **emacs**, **nedit**, enter in the Editor Command field:

exmh-bg ***editor***

where ***editor*** is the editor of your choice. If your preferred editor does not have this capability, e.g. **vi**, you'll need to enter:

exmh-bg xterm [-*options*]-e ***editor***

where **-*options*** are any options (i.e. font specification, preferred colors, geometry, etc.) to the **xterm** command. If you don't know about your editor, you'll find that one of these strings works and the other doesn't.[a]

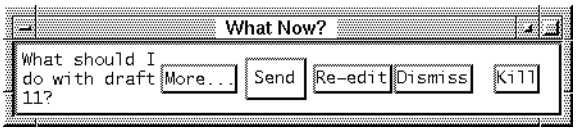
Your selected editor must be in your *PATH*; you cannot use an alias definition instead.

Click on **DISMISS** when you're done, and then be sure you click on **SAVE** in the "Exmh Preferences" menu.

a. If you have X-Authority, you can configure an editor using **exmh-async** ***editor***.

To send a message in which you want to include a file, compose the message, and use the "include file" command appropriate for the editor you're using.

When you save the composed message you are presented with a "What Now?" dialog box (a feature of **MH**) that gives you options to **SEND**, **RE-EDIT**, **DISMISS** (this "dismisses" the pop-up window and leaves the message in the drafts folder), or **KILL** the message.



mh

mh will throw you into **vi** by default for composing a message. This is also true for forwarding and replying to messages. You can specify a different editor on the command line if you like.

To compose a message in **vi**, enter:

% comp	This brings up a **vi** session. Compose your message. Then save the message file, and exit the editor.

Remember to have Chapter 10's handy list of **vi** "survival commands" with you! Just in case you get stuck, here's your emergency exit: hit the **ESCAPE** key (or **<CTRL-[>**) and then enter:

:x	to save and exit, or
:q!	to quit without saving

Once you exit the editor, **MH**'s **What Now?** program is automatically invoked to prompt you for an action. At the `What Now?` prompt, you can enter a carriage return to get a list of options. Normally you will enter one of these options:

send	to send the message
quit	to save the message in `Mail/drafts` but not send it
quit -delete	to quit and not save.

To compose a message in a different editor, enter:

% comp -e *editor* This brings up a session in your selected editor. This option will override any editor defaults. Compose your message. Then save the file, exit the editor, and respond to `What Now?`

If you *really* want to avoid **vi** at all costs, we'll show you how to set a different default editor.

> To change your default editor for **mh**, add the following line to your `.mh_profile` file:
>
> `Editor:` *`editorname`*
>
> where *`editorname`* is your preferred editor, for example `emacs`.

If you want to send a file rather than a message, use the command:

% send *file* This sends ***file*** to all the destinations defined, namely `To:` at a minimum, and `cc:`, if you've filled it in. (If you change your header to include `Bcc:` (blind copy), `Fcc:` (folder copy), it will send it to addresses in these categories as well.)

To send a message in which you want to include a file, compose the message, and use the "include file" command appropriate for the editor you're using. In **vi** enter:

:r *filename*

11.3.3 Incorporating and Reading Incoming Messages

Mail incorporation refers to the process of moving incoming mail messages from the default mail holding area on your UNIX node into your personal area. Section 11.4.3 discusses this in more detail. You can't read messages until you incorporate them. As messages get incorporated according to **MH** default standards, they go to the directory `$HOME/Mail/inbox` (`inbox` is called a folder), which was created for you during setup. Each message becomes a separate, numbered file in `inbox`. When you incorporate messages, you get all the messages in the spool; you cannot choose to incorporate some and not others.

exmh

There are a few ways to incorporate mail during an **exmh** session. You can incorporate your messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

To incorporate messages manually from within **exmh**, click **INC** from the middle window's menu. By default your spooled mail is placed in your `inbox` folder. You should now see the headers of the messages you sent yourself in the middle window. (If you didn't already send yourself some mail, go back and do it now.)

Once you incorporate your message(s), you can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. The message displays in the bottom window.

To set up immediate or periodic incorporation during your **exmh** session, see section 11.4.3.

mh

First, incorporate the new message(s) into the **MH** system using the command:

```
% inc
```

This will display the headers of the incorporated messages, with a plus sign (+) next to the current message. The current message is the first unread one, usually the first one you just incorporated unless you had unread messages already in the folder. (If you didn't already send yourself some mail, go back and do it now.)

After incorporating, you can use **scan** to display the message header(s):

% scan	Display message headers from the current folder (since we haven't created any other folders, the current one is `inbox`).
% scan last:*n*	Display the last ***n*** message headers from the current folder.
% scan *m*–*n*	Display message headers ***m*** through ***n***, inclusive, from the current folder.

Here's our first example of using **mh** commands together with standard UNIX commands. To select message headers in `inbox` according to a ***pattern***, you could pipe **scan** to **grep** in this way (the **+inbox** specifically requests the `inbox` folder, and the **–i** option of **grep** indicates a case-insensitive search for ***pattern***):

```
% scan +inbox | grep -i pattern
```

To read the current message, enter:

% show [*n*]	Display the current (or specified) message.

Note that the **show** command can be used with more than one message number. For example **show *n*** can be expanded to **show *m n* ...** or to **show *m*–*n***.

Other commands to read messages are:

% next	Display the next message.
% prev	Go back and display the previous message.

11.3.4 Replying to Messages

The default reply header looks like this:

```
To:
cc:
Subject:
In-reply-to:  Your message of "Day, date time"
              <message-id>
--------
```

If you don't like the standard reply header, you can reformat it by creating your own `replcomps` file. See section F.2 for information.

When you want to include the original message in your reply, it is often convenient to precede each line of the original message text by a `>` symbol, to distinguish it from your text. FUE sets this up automatically for **exmh** and **mh**.

This formatting can be inconvenient if the recipient needs to work with the original text, and thus needs to remove all the `>`'s. You can avoid this by *forwarding* the message back to the original sender, instead of replying. Forwarding is covered in section 11.3.5.

If you want to change the format of your replies, you can create a file called `.mh_filter` in your `$HOME` directory. The O'Reilly **MH** book describes the different possible ways of setting this up. The contents of the default `.mh_filter` is:

```
body:component="> ",compwidth=0
```

exmh

Select the message to which you want to reply so that it displays in the message (bottom) window. Then click on the **REPLY** button in the bottom window. You get a menu of options. **REPLY TO SENDER** sends it just to the sender, while **REPLY ALL** sends it to everyone listed in `To:` and `Cc:`. To include the original message in the reply, choose one of the **REPLY** menu options with **/INCLUDE**.

Once you have made a selection from the **REPLY** menu options, a composition window pops up. The header fields will be filled in. You can edit the header if needed. Move the cursor to the body area and type in your reply.

For **sedit**, click on the **SEND** button to send the reply. To skip the reply, click on **ABORT**.

For other editors, save the message and exit the editor. Then respond to the "What Now?" pop-up menu.

mh

By default, the **`repl`** command uses the **vi** editor. To avoid this, either use the **`-e`** ***editor*** option or set a different default editor in your `.mh_profile`.

To reply to a message, enter:

`% repl [`*`n`*`]` Reply to the current (or specified) message. By default, everyone who got the original message gets a copy (the sender, plus everyone in `To:` and `cc:`).

`% repl_inc [`*`n`*`]` Like **`repl`**, but include the original message. (This FUE command replaces the standard **mh** command **`repl -filter`** ***filter_file***.)

Either of these opens an editing session for the reply. As with **`comp`**, save your message and respond to the `What Now?` prompt.

The **`-query`** option used with either of these commands prompts about each recipient separately. To reply to selected addresses in the `To:` and `cc:` fields of the original message, enter:

`% repl -query` Reply to the current message. You will be prompted for each address in turn, to which you reply `y` or `n`, for example:

```
Reply to user1@FNAL.GOV? y
Reply to user2@xxx.yyy? n
```

Use the **`-nocc`** ***headerfield*** option with either command to prevent all addresses in a particular header field from receiving the reply; for example:

`% repl -nocc cc [`*`n`*`]`

Reply to the current (or specified) message, including all addresses except those in the original `cc` list.

11.3.5 Forwarding Messages

By default, forwarding uses a header file called `forwcomps` in the **MH** library directory. This file is similar to that used for the compose function (`components`). If you want your header for forwarding messages to be identical to that for sending messages, you can use the components file for both. Just include the following line in your `.mh_profile` file:

```
forw: -form components
```

If you want to customize the header for forwarding mail, you need to set up your own file called `forwcomps` in your `$HOME/Mail` directory. Edit it in the same manner as `components` (see section F.2).

exmh

To forward a copy of a message to someone, first select the message. You can select more than one message at a time. Then select **FORWARD** from the bottom window. A composition window comes up with a blank header and the message text. Fill in the header, and edit the message if you wish. Send or abort as explained earlier, according to your editor.

mh

By default, the **forw** command uses the **vi** editor. To avoid this, either use the **-e** ***editor*** option or set a different default editor in your `.mh_profile`.

To forward a message, enter:

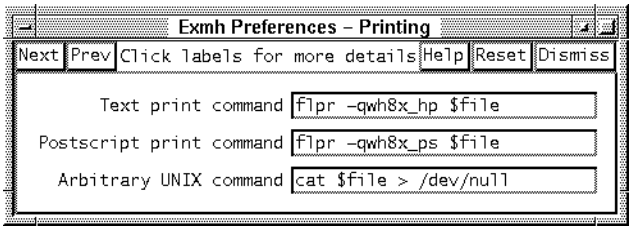
% forw [*n*] Forward the current message (or the specified message).

This opens an edit session. Fill in the header, and edit the message as you like. As with **comp** and **repl**, save your message and respond to the `What Now?` prompt.

11.3.6 Printing Messages

exmh

First set your print queue by selecting **PRINTING** under **PREFERENCES** in the top window. It should look like the following, with your own print queue substituted:



Click **DISMISS** when you've completed the change on the "Printing Preferences" window, then click **SAVE** on the "Exmh Preferences" window.

To print the current message, use the **PRINT** option from the **MORE...** menu in the bottom window.

To print several messages from a single folder, select them in the middle window, and use the **PRINT** option from the **MORE...** menu in the bottom window.

mh

To print a mail message, you use an **mh** display command piped to a UNIX print command (normally **flpr** under FUE). Any printing defaults you've set up for printing files will be used here (printing is covered in Chapter 7).

To print message number ***n*** on ***queue***, enter:

% show *n* | flpr -q *queue*

The option **-showproc** with the **mhl** argument formats the message a little differently so that it's easier to read (and nicer for printing), and pipes it to **more**. You would type:

% show -showproc mhl *n* | flpr -q *queue*

Especially good for printing multiple messages is the argument **pr** which separates mail messages onto successive pages:

% show -showproc pr *m n* ... | flpr -q *queue*

11.3.7 Extracting Messages

In **MH** format, each message is already a separate file, and can be manipulated as any other file. You may find however, that extracting messages is easier or quicker than standard UNIX file manipulation for some tasks.

exmh

To extract the current message to a file, use the **SAVE TO FILE** option from the **MORE...** menu in the bottom window. A window pops up to allow you to choose the file to create or to which you want to append the message.

mh

To extract a message into a file use standard UNIX output redirection:

% show ***n*** **>** ***filename***

This is essentially equivalent to the command:

% cp $HOME/Mail/*folder***/***n*** *filename*

The show command includes one extra line that the **cp** command doesn't create:

`(Message folder:n)`

11.3.8 Removing Messages

First, a note about message removal: When you use **rmm** in **mh**, or the **DELETE** button in **exmh**, the files are not actually deleted. They are renamed with a comma (**,**) or pound sign (#) in front so that **exmh/mh** doesn't recognize them as messages. The files are still there consuming disk space, and they don't go away unless YOU do something about them.

You can choose to periodically go through and clean out the old files that you no longer need in order to recover disk space. In **exmh**, use the **PURGE ALL FOLDERS** button (under the **MORE** menu in the middle window). This will delete the message files older than (default of) 7 days which you had previously "renamed". Or manually use the UNIX command **rm** on all files beginning with `a` , or `#` (user-configurable parameter).

A better solution is to add an entry to your `.mh_profile` changing the behavior of the **DELETE** button / **rmm** command:

```
rmmproc: /bin/rm
```

This will really delete the message files when you use **rmm** (**mh**) or when you hit the **COMMIT** button after deleting messages (**exmh**). For clarity in the following text, we simply refer to deleting messages.

exmh

To delete one or more messages, first mark them for deletion by selecting the headers in the middle window then click on **DELETE** from the bottom window. The message headers you selected will be highlighted; on some terminals they will be a different color or have a background pattern. If you are sure that you want to delete the marked messages, click on the **COMMIT** button in the middle window. If you change your mind before you commit, select the messages you do not want to delete and unmark them (**UNMARK** is an option under **MORE...** in the bottom window).

Note that after selecting a message, the cursor may move down to the next one, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

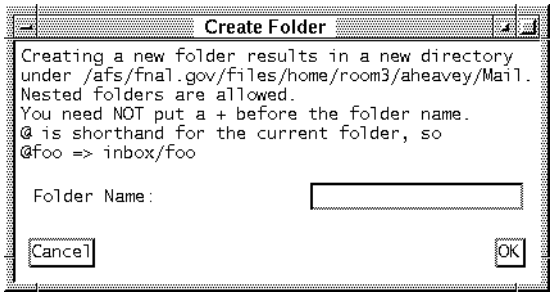
Use the **rmm** command to remove messages:

% rmm [*n m ...*]	Delete the current (or specified) message(s).

11.3.9 Creating, Changing and Removing Folders

exmh

To create a new folder from within **exmh**, select **NEW** from the middle window. The following window pops up for you to fill in. When you're done, click **OK** to create your new folder.



To switch current folders, click on the desired folder name in the top window with the left mouse button and the message headers from that folder will display in the middle window. It is now the current folder. You can read and work with the messages from this folder.

To view nested folders and their contents (see section 11.4.2), click the "parent" folder with the middle mouse button, then select a nested folder with the left mouse button.

To remove a folder, select it, then click on **DELETE FOLDER** under **MORE...** in the middle window.

mh

When you refer to a folder on the command line, precede it with a plus sign (**+**). To create a new folder, enter:

% folder +*new_folder_name*

There are a few ways to reset the current folder to a different one. The most explicit command for this is:

% folder +*new_current_folder*

Or, just running one of several commands with a different folder name changes the current folder until you again change it to something else. For example:

% scan +*folder_name*	Display message headers from ***folder_name*** and set this folder to current.
% show +*folder_name*	Display the current message in ***folder_name*** and set this folder to current.

Note that the **refile** command (see section 11.3.10) accepts folder name arguments but doesn't change the current folder.

To remove a folder, enter:

% rmf +*folder*	Delete the specified folder.

11.3.10 Refiling Messages

exmh

To refile one or more messages into a different folder, first select the header(s) in the middle window, then click the right mouse button on the destination folder in the top window. If they're moving to a nested folder, first click on its parent folder with the middle mouse button, and then on the nested folder with the right mouse button. The messages are now marked for refile. The message headers will be highlighted; on some terminals they will be a different color or have a background pattern. Click the **COMMIT** button (middle window) to complete the refile operation. If you change your mind before you commit, select the messages you do not want to refile and unmark them (**UNMARK** is an option under **MORE...** in the bottom window).

Note that after selecting a message, the cursor may move down to the next message header, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

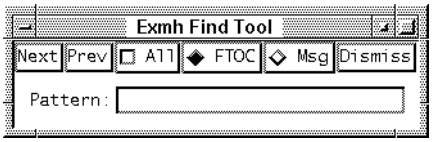
To move the current message (or specified messages) from the current folder to a different folder, enter:

```
% refile +folder [n m ...]
```

11.3.11 Searching for Messages

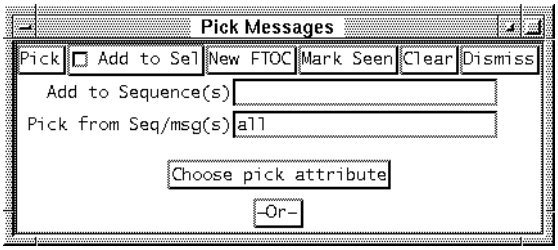
exmh

To search messages in a folder, first select the folder in the top window. Then select **SEARCH...** from the middle window's menu and select the relevant option. You can get help on various items by looking into the search help. If you choose **FIND IN TABLE OF CONTENTS**, for example, you get a Find Tool window where you can type in a search pattern.



The headers of the "found" messages are highlighted. You can then display only the headers of these messages by choosing **LIST ONLY SELECTED MESSAGES** from the **MORE...** menu.

The **PICK BY ATTRIBUTES** from the **SEARCH...** menu provides an interface to the **MH `pick`** command. This is a more sophisticated feature that allows you to set up complex logical search expressions. If you use the "before" or "after" search criteria, the date format accepted is `nn-mmm-yyyy`, for example `12-jan-1996`.



You have the option of saving the *sequence* of the picked messages. **MH** sequences are stored lists of message numbers. When you've finished setting up your search criteria, click **PICK**.

Another option you have is to use the **glimpse** utility, also available under the **SEARCH** menu for full text searches. This requires building an index (use the menu option **INDEX** under **GLIMPSE FULL TEXT**), which may take a long time. For more information see the man pages for **glimpse**.

mh

Pick searches messages within a folder for the specified contents, and then identifies those messages. Two types of search primitives are available: pattern matching and date constraint operations. See the man pages for more information on **pick**.

% pick -help Display a list of pick options.

% pick ***options*** Select messages by content, date, sender, recipient, etc.

You can use **pick** to find messages sent from someone, sent to someone, on a given date, before or after a date, and even search every line of every message in the current folder. To tell **pick** to scan through every line of every message in the folder `recent_work` and return the message numbers with the word "quark" in them, enter:

```
% pick -search quark +recent_work
```

Here is an example showing the selection of messages within a date range:

```
% pick -after 1-dec-1995 -and -before 1-jan 1996
```

You can use **pick** with sequences, which are described above in the **exmh** portion. Commands that accept message numbers accept sequences as well; just type the sequence name. For example, find all messages received from ***sendername***, and store them in the sequence **picked**:

```
% pick -from sendername -sequence picked
```

The result also tells you how many messages are picked. You can then scan these messages by typing:

```
% scan picked
```

You can search for a string within the sequence by typing, for example:

```
% pick -subject 'quark substructure' -sequence picked
```

You can use **pick** and **scan** together (note the use of backquotes here):

```
% scan `pick -from sendername`
```

11.4 Basic Configuration for MH

11.4.1 Configuration Files

In order to take advantage of the variety of features provided by the **MH** package, several configuration files can be used for both **exmh** and **mh**. In this section we list all the files discussed in this chapter and in Appendix F, and provide brief descriptions for reference.

You're already familiar with two of them, `.mh_profile` and `.forward`.[1] See sections F.2.1 and F.1, respectively, for more information on these two files.

`.mh_profile` defines your basic configuration (where mail is stored, the editor to use, various MH options, etc.). Running **setup mh** or **setup exmh** the first time automatically creates a default `.mh_profile` for you in your `$HOME` directory.

---

1. The `.forward` file is not specific to the **MH** system; it may be used with any mail handler.

`.forward` stores your forwarding address. It is used in mail forwarding (routing), notification, and unattended autoincorporation. You must create it in your `$HOME` directory.[1]

The following three files allow you to customize your mail headers. See sections F.2.2, F.2.3, and F.2.4, respectively, for more information on them:

`components` defines message header format for sending mail. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`replcomps` defines message header format for replies. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`forwcomps` defines message header format for forwarded messages. If you don't create your own copy in your `Mail` directory or elect to use the `components` file header format, the system default file in the **MH** library section will be used.

In sections 11.3.4 and 11.4.5 we show you how to create:

`.mh_filter` allows you to include the original message in a reply. It goes in your `$HOME` directory.

`alias file` stores distribution lists and local aliases for long mail addresses (its name is defined in `.mh_profile`). It is assumed by default to be in your `Mail` subdirectory, but you can define it otherwise.

In Appendix F we provide information on yet a few more configuration files!

`scan-form` provides the format of display when you scan mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`inc-form` provides the format of display when you incorporate mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`.maildelivery` controls how local delivery is performed with manual or background incorporation in **exmh** when the **PRESORT** option is selected. It also controls how delivery is performed when unattended automatic mail incorporation is defined in `.forward` using **slocal**. No default is provided; if you need this, you must create it in your `$HOME` directory.

11.4.2 MH Mail Folders

MH uses the concept of mail folders to store messages. Folders are simply subdirectories of your `Mail` directory which contain mail messages, one message to a file. To distinguish folders from files in **MH** commands, folders are prefixed with a plus sign (**+**) in a command. For example:

```
% refile +admin
```

is an **MH** command that moves (refiles) the current message in the current folder to the folder `admin`.

---

1. If you only have an account on one machine (i.e. one node that is not part of a file-sharing cluster) besides FNAL and if you don't want automatic mail incorporation or notification, you don't need the `.forward` file.

Folders can be multi-level, for example `outbox/feb96`. Within a folder, the message filenames are incremented numbers starting at one (1). Running **`ls -l`** on a folder, you will see a numbered file for each message (in "alphabetical" order), for example:

```
total 296
-rw-r--r--   1 username  g020         339 Nov  9 11:36 1
-rw-r--r--   1 username  g020         864 Nov  9 11:36 10
-rw-r--r--   1 username  g020        3297 Nov  9 11:36 11
...
-rw-r--r--   1 username  g020         753 Nov  9 11:36 2
-rw-r--r--   1 username  g020        2628 Nov  9 11:36 20
```

The command **`ls -lt`** is convenient for listing them in reverse chronological order.

After a lot of refiling and deleting of messages, the messages in your folders may not be in order by date anymore, and you will undoubtedly have numerical gaps in the file names. To reorder and renumber the message files:

exmh	use the **SORT FOLDER** and **PACK FOLDER** options under the **MORE...** button in the middle window
mh	use the commands **`sortm`** and **`folder -pack`**, respectively

11.4.3 Incorporation of Incoming Mail into Folders

The UNIX System Mailbox

Due to the wide variety of mail readers on UNIX and the configurability provided by many of them, incoming mail is not by default delivered to any particular file or area in your home directory. Instead it is collected in a *system mailbox*, a directory defined as the mail holding area, on each node. In FUE, the environment variable *MAIL* is defined to reflect the system-specific location of your mailbox file, set to `/usr/spool/mail/`*`username`* or `/usr/mail/`*`username`*, depending on the platform.[1] For example, user `fred` has his unread, unincorporated incoming mail stored in chronological order in the file `/usr/mail/fred`.

Your mail resides in this file until you incorporate it into a different file (mail folder). How and where it gets incorporated depends on the mail reader you decide to use. Here we discuss incorporation into the **MH** mail folder structure.

Options for Incorporating Mail

When new mail arrives, by default it resides the system mailbox until you incorporate it into your mail folder. There are a few ways to incorporate mail during an **exmh** session (i.e. "attended" incorporation), as we mentioned in section 11.3.3. You can incorporate messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

If you want to configure **exmh** so that it incorporates your mail at regular intervals and/or at particular times during your **exmh** session, first go to the **PREFERENCES** menu in the top window.

- Choose the **INCORPORATE MAIL** menu. Do you want to incorporate new mail when you invoke **exmh**? Check the box. Do you want to incorporate waiting mail when you click to open the **exmh** icon? Check the box. Dismiss this window and save your preferences.

---

1. See the file `fermi.login` (for C shell family) or `fermi.profile` (Bourne shell family) in Appendix E. On a SunOS 5 system, you may have to set the variable *MAIL* manually due to a bug.

- Choose the **BACKGROUND PROCESSING** menu. Do you want to incorporate new mail periodically? Check the **INC** box. The period is set to 10 minutes by default, but you can change this value. Dismiss this window and save your preferences.

Would you like **exmh** to sort your mail for you, automatically putting some messages in a particular folder? If so, you need to do two things:

1) Create a `.maildelivery` file describing how you want your mail sorted (see section F.2.6).

2) Tell **exmh** to use this file. Go to the **PREFERENCES/INCORPORATE MAIL** menu, and under **WAYS TO INC**, check the **PRESORT** box.

We discuss unattended automatic mail incorporation in section F.3. This is different from attended automatic incorporation in that mail is automatically incorporated whether or not you're running **exmh**, and whether or not you're even logged in. We do not recommend this method, although we include it here for completeness.

11.4.4 Signature Lines

To include a boilerplate signature to messages that you compose and send in **exmh** or **mh**, you need to create your own `$HOME/Mail/components` file, and simply add the signature to the bottom of this file, underneath the dashed line. For information on the `components` file, see section F.2. This file does not govern replying and may not govern forwarding (see section 11.3.5).

To add a boilerplate signature when replying to messages, you can add a signature to the end of your filter file, called `.mh_filter` in your `$HOME` directory (see section 11.3.4). This file gets used only when you're using the option to include the original text in the reply.

If you're using your own `$HOME/Mail/forwcomps` file for forwarding messages, include the signature lines at the end of this file. Note that your signature will appear *above* the text of the forwarded message.

11.4.5 Mail Aliases

You can send mail to many people at a time by defining an alias for the addresses of the recipients. You may also want to define aliases for long single addresses. Create a file (we call it `mh_alias` for example purposes; you can call it what you like) in your `$HOME/Mail` directory, and include an `Aliasfile` line like the following in your `.mh_profile`:

```
Aliasfile: mh_alias
```

This example alias file shows the required format:

```
d0spokes: mont@fnal.gov,\
          pgrannis@fnal.gov

cdfspokes: carithers@fnal.gov,\
           bellettini@fnal.gov
```

There cannot be any blanks after the backslash (\) in the aliases. To see your aliases you can use the command **`ali`**. For example if you type:

```
% ali d0spokes
```

you will get the response:

```
mont@fnal.gov, pgrannis@fnal.gov
```

11.4.6 Folder Order and Header Display (exmh)

Some further options available for **exmh** (but not for **mh**) to include in the `.mh_profile` file are:

```
Folder-Order: inbox folder2 folder3 * drafts
Header-Suppress: .*
Header-Display: Date To Cc From Subject
```

`Folder-Order` sets up the order of display of folders on your **exmh** window. Wildcards as used in filename expansion are allowed, so you only need to specify folders that you want displayed in particular places. In our example, we specify what folder to show first, second, third, and last. All the others will be sorted alphabetically between the third and the last.

`Header-Suppress` and `Header-Display` control which headers you see when you read a message. The example above indicates that everything is suppressed except the `Date:`, `To:`, `From:`, `Cc:`, and `Subject:` lines.

11.5 Berkeley Mail

In this section we discuss the simpler, less functional mail handler **Berkeley mail** because it is one that you will find on all UNIX machines. We describe how to read and send messages from the command line using the command **`mail`** or **`fermimail`**. In the Fermi files that run when you log in, the command **`mail`** is set to point to **Berkeley mail** by default (via the alias **`fermimail`**).

11.5.1 Sending Messages and Files

To send a mail message, the command format is:

`% mail [-s` ***subject*****`]`** ***recipient*** **`[<`** ***filename*****`]`**

If you don't include the **`-s`** option with a ***subject*** (subject must be enclosed in quotes if it's more than one word), the system will prompt you for one after you hit carriage return. The ***recipient*** is a user name in standard format. If you want to send a file instead of composing a message, use the input redirection (**<**) with the ***filename*** (and path, as necessary). If you don't include a file, the cursor will be placed at the start of the first line under the command (no prompt) for you to start composing your message. Conclude the message by going to a new line and typing a period (**.**) followed by carriage return. You will be prompted for a `Cc:` before exiting. To abort a message that you are composing, use the standard *kill* control sequence (see section 2.4).

A very simple example command line using this format to compose a message is:

```
% mail butler@fnal
```

An example using the subject option and sending a file is:

```
% mail -s "Meeting minutes" butler@fnalb.fnal.gov < minutes
```

11.5.2 Reading Messages

Assuming you have no autoincorporation set up, all of your unread incoming mail resides in the file designated by the variable *MAIL* (see section 11.4.3). When you log in, the system notifies you if this file contains any messages. It will not automatically notify you of incoming mail at other times during your session, unless you are in a mail reader.

To read your messages in the system mailbox, simply type:

```
% mail[1]
```

This puts you in command mode from which you can read messages. Enter **?** to get a list of commands. If there are messages, after you read them and when you are exiting mail, the system automatically sends the messages to your `$HOME/mbox` file, and deletes them from your system mailbox file. If there are no messages to read, the system will tell you so and exit.

For more information, see the man pages for **mail**.

1. Note that on some systems the command may be **mailx, Mail** or **fermimail**.

Chapter 12: Connecting to Remote Systems

Several utilities are available to enable you to transfer files between systems, to log into other systems on which you have an account, and to execute commands remotely. These features are described in this chapter.

Note that to use *any* of the communication methods with a VMS node, the VMS node must be running software to allow tcp/ip communication. All Fermilab VMS machines are licensed to run TGV's **MultiNet** software, which provides the necessary tcp/ip transport capability.

12.1 Transferring Files

12.1.1 ftp

ftp (File Transfer Protocol) can be used to transfer files to and from a remote system on which you have an account, and to manipulate the remote file system. The command format is:

`% ftp [-n] [-i] [`*`host`*`]`

If you specify ***host*** on the command line, you will be prompted for your (remote) username and password, and login will proceed. The option **`-n`** inhibits autologging on those systems on which it is enabled. The option **`-i`** eliminates prompting by the system, which can be tiresome when you're transferring many files.

Some commands available in **ftp** are described here; for a complete list, see the man pages for **ftp**.

`user` ***username password***	Identification for the remote system. If auto-log is enabled, you will be prompted automatically for the username and password. If you don't specify the password, it will prompt you.
`put` ***local_file*** [***remote_file***]	Transfer a local file to the remote system. If the remote filename is not specified, the local filename is used.
`mput` ***local_files*** [***remote_files***]	Similar to **`put`**, but you can use standard wildcard characters to transfer a series of files at a time.
`get` ***remote_file*** [***local_file***]	Transfer a remote file to the local system. If the local file name is not specified, it is given the name on the remote machine.
`mget` ***remote_files*** [***local_files***]	Similar to **`get`**, but you can use standard wildcard characters to transfer a series of files at a time.
`help` [***command***]	Display information about the meaning of ***command***. If no command is specified, **ftp** displays a list of the commands.
`quit`	Terminate the **ftp** session with the remote server and exit **ftp**.

open *host*	Establish connection with remote host. Needed if ***host*** was not specified on command line.
close	Close connection with remote host and return to **ftp** prompt. Doesn't exit **ftp**.
binary	Set file transfer type to binary image transfer.
dir [*remote_directory*] [*local_file*]	Print listing of the directory contents of the remote directory and optionally put the output in ***local_file***.
cd *remote_directory*	Change working directory on the remote machine to ***remote_directory***.
lcd [*directory*]	Change working directory on the local machine. If no directory is specified, your home directory is used.
delete *remote_file*	Delete the file ***remote_file*** on the remote system.

The following is an example **ftp** session where first one file, then a few files, are transferred to a UNIX machine from the FNALV cluster. The username is unnecessary if the assumed one is correct. The password, as usual, is not displayed.

```
<fsui01> ftp fnalv

Connected to fnalv.fnal.gov.
220 fnalv1.fnal.gov MultiNet FTP Server Process 3.4(14) at Wed 3-Jan-96 11:41AM-
CST
Name (fnalv:aheavey): 331 User name (aheavey) ok. Password, please.
Password:
230 User AHEAVEY logged into USR$ROOT17:[AHEAVEY] at Wed 3-Jan-96 11:42AM-CST, j
ob 25a26688.
ftp> get newfile
200 Port 159.217 at Host 131.225.68.16 accepted.
150 ASCII retrieve of USR$ROOT17:[AHEAVEY]NEWFILE.;1 started.
226 Transfer completed.  25 (8) bytes transferred.
local: newfile remote: newfile
25 bytes received in 0.14 seconds (0.18 Kbytes/s)
ftp> lcd psfiles
Local directory now /afs/fnal.gov/files/home/room3/aheavey/psfiles
ftp> mget c*
mget cmdconcp.frame.ps? y
200 Port 159.219 at Host 131.225.68.16 accepted.
150 ASCII retrieve of USR$ROOT17:[AHEAVEY]CMDCONCP.FRAME$5NPS;1 started.
226 Transfer completed.  182118 (8) bytes transferred.
local: cmdconcp.frame.ps remote: cmdconcp.frame.ps
182118 bytes received in 0.69 seconds (2.6e+02 Kbytes/s)
mget cmdconcp.ps? n
mget codemgmt.frame.ps? y
200 Port 159.224 at Host 131.225.68.16 accepted.
150 ASCII retrieve of USR$ROOT17:[AHEAVEY]CODEMGMT.FRAME$5NPS;1 started.
226 Transfer completed.  81133 (8) bytes transferred.
local: codemgmt.frame.ps remote: codemgmt.frame.ps
81133 bytes received in 0.32 seconds (2.5e+02 Kbytes/s)
ftp> quit
221 QUIT command received. Goodbye.
<fsui01>
```

Note that the CERN utility **zftp** (**z** for Zebra) is available for transferring FZ and RZ files (ntuple files) between systems. To use **zftp**, it must be installed on the server system, and you need to set up CERN library during the session (**setup cern**).

12.1.2 rcp

Another way to transfer files is with the **rcp** utility (stands for **r**emote **c**o**p**y). You can use this command if both local and remote hosts support **rcp** protocol. The basic format is:

% rcp [*options***]** ***file1 file2***

where ***file1*** and ***file2*** are the source and destination filenames, respectively; or

% rcp [*options*] *file1 file2 ... directory*

where the files listed are to be copied to the directory. On a UNIX system the format of a remote directory or filename is ***hostname*:*path***, and the **MultiNet** implementation is consistent with VMS usage, namely ***hostname*::*path*** (note the double colon). If only filenames are given, they are interpreted relative to your home directory. Both the source and the destination may be on nodes other than the current machine.

The available options are:

-p	Give each copy the same modification times, access times, and modes as the original file.
-r	Copy each subtree rooted at ***file1***, ***file2***, etc. (for those that are directories); in this case the destination (last argument) *must* be a directory.

rcp does not prompt for passwords. Your current local user name must exist on the remote host and allow remote command execution by **rsh**. **rsh** is a utility that connects to a remote host and either executes a specified command or logs you on via **rlogin** (see section 12.3).

More information is available in the **rsh** and **rcp** man pages.

When you are using **rcp** between VMS and UNIX, you must protect special characters from local interpretation by enclosing them in quotes. An example command to copy a file `$HOME/prog1.c` from UNIX to the VMS node FNALV is:

```
% rcp prog1.c fnalv:"[.c]prog1.c"
```

Here is an example showing how to copy a file from remote UNIX node FNSG01 to the home directory on the VMS node (**$** represents the VMS prompt):

```
$ rcp fnsg01::"/usr/products/bufio/v1_00/src" bufio.src
```

If you are using the C shell family, **rcp** will not work if your `.cshrc` or `.login` on UNIX, or your `LOGIN.COM` file on a remote VMS host, executes interactive or output-generating commands. You should check for non-interactive access, and exit before such commands are executed, as is done for a UNIX system in the FUE default `.cshrc` file. The lines in `.cshrc` that check this are the following:

```
# Place any items that you want executed even for non-interactive use here
# Skip if not interactive shell
if ( $?USER == 0 || $?prompt == 0 ) exit
```

On a VMS system, check for **OTHER** (TCP/IP is not **NETWORK**):

```
$ IF (F$MODE () .EQS. "OTHER") THEN $ EXIT
```

Although you can specify a username and password with **rcp** (see the local man pages or **HELP**), you can also have a kind of "proxy login" by creating a special file (named `.rhosts`) on the remote machine containing the names of the systems and users on those system that should be allowed access.

12.1.3 The .rhosts File

To use many of the file transfer utilities, you need to set up what is known as a `.rhosts` file on the remote machine. This file must include a list of the names of the systems to be allowed remote access without login, and the usernames, where each system-username combination is separated by

a space. If the username is left off, it is assumed to be the same as on the current system. If there are problems with name servers, you can be most assured that your address will be understood if you specify three lines for each connection requested:

1) one line with the complete Internet address,

2) one with the Internet number[1], and

3) one with the local name.

An example `.rhosts` file showing a single connection in the three specified formats:

```
fnsg01.fnal.gov
131.225.8.178
fnsg01 judy
```

When writing a `.rhosts` file on a UNIX host to allow access via remote commands (known as *r-commands*[2]) from a VMS cluster, you must include an entry for *each* cluster member and username from which you might access the UNIX host.

The following command line is an example of the command to copy a `FINGER.PLN` file from an account on FNALA to the file `.plan` in the same person's account on the UNIX node FNSG01. In order for this example to work, there must be a file named `.rhosts` on FNSG01 containing a line `fnala.fnal.gov`, and the username must be the same in both places.

```
$ RCP FINGER.PLN FNSG01::.PLAN
```

where **`$`** is the VMS prompt.

12.2 Logging in to Other Systems

12.2.1 telnet

telnet is a remote terminal protocol which allows you to establish a connection to a login server at a remote site. **telnet** in the UNIX world is analogous to **`SET HOST`** in the VMS world. You can connect to other UNIX machines, or you can connect to VMS machines if the appropriate software is installed on the system (e.g., with **MultiNet** for which Fermilab has a site license). The format is:

`% telnet [`*host*`]`

If you include the host, you are prompted for the username[3] and password. If you do not include the host, you go into command mode. You can enter **`help`** at this point for help on the commands.

---

1. Use **`nslookup`** ***node*** to get the number. On VMS, you must enter **`MULTINET NSLOOKUP`** ***hostname***.
2. r-commands execute on remote hosts, and use `.rhosts` for authentication. They are variants of UNIX commands without the "r", and include for example **`rsh`** (**`r`**emote **`sh`**) and **`rlogin`** (**`r`**emote **`login`**).
3. Some systems may not prompt you for a username; they will use your current login id. If you need to log in with a different username, enter a carriage return at the password prompt. Then the system will prompt you for a username.

12.2.2 rlogin

Another way of logging into a remote computer from another system on the network is by use of the **rlogin** utility. One difference between **telnet** and **rlogin** is that if the communications package is configured for it and you create an appropriate `.rhosts` file (see section 12.1.3), you can log in to the remote system using **rlogin** without typing the password. The format from a UNIX machine is:

% rlogin ***rhost*** **[-l** ***username*****]**

where:

rhost	is the remote host to which you want to log in
username	is the username if the name is different on the remote system than the originating system

From a VMS machine, the format is:

$ RLOGIN /USERNAME=*username rhost*

12.3 Executing Commands Remotely: rsh

You can execute commands on a remote system and have the output displayed on your terminal with **rsh**. **rsh** stands for **r**emote **sh**ell. In order for this to work, there must be an appropriate `.rhosts` file on the remote machine (see section 12.1.3).

The format is:

% rsh ***rhost*** **[-l** ***username*****]** ***command***

where:

rhost	is the remote host on which the command is to be executed
username	is the username on the remote system if different than the originating system
command	is the command to be executed on the remote system

The command may be a single command or a shell script to be executed on the remote system. Metacharacters in the command must be quoted for proper interpretation on the remote machine.

The **MultiNet** implementation on VMS is:

$ RSHELL[/INPUT=*infile***][/OUTPUT=***outfile***][/USERNAME=***username***]-** ***hostname commandline***

The **MultiNet** implementation does not handle metacharacters like the UNIX implementation does.

The **/INPUT** and **/OUTPUT** options in the **MultiNet** implementation allow you to execute a task on a UNIX machine which reads and writes files on VMS.

If the command is omitted, you will be logged in to the remote host using **rlogin**.

Chapter 13: Software Development

This chapter gives an introduction to UNIX software development tools in common use at Fermilab, providing information on:

- Supported languages
- Compiling and linking in C, C++ and FORTRAN
- Debugging

We do not include a discussion of general programming here, but rather, aspects of software development particular to UNIX. You will need to reference the man pages and the vendors' manuals for more system-specific details.

Useful documents relating to FORTRAN, C, and C++ programming can be found under **Software Development** on the *UNIX Resources* WWW page.

Many software development subjects of interest to Fermilab users are beyond the scope of this manual. Among them are:

- Object Oriented programming techniques including NextStep
- CASE tools
- Neural Network methods
- Lattice Gauge techniques used in ACP-MAPS, CANOPY, et.al.
- Data Mining methods available in the CAP facility.
- Many excellent commercial tools for building Graphics User Interfaces, debugging, migrating and analyzing performance of user code, building Database interfaces, etc.

13.1 Overview of Programming Languages and Tools

This is a short overview of some common programming languages. Our aim here is to give you a general idea of what tools are available, and how they can be used. The list is ordered from low to high level, and indicates common uses:

Assembler	not used at Fermilab
C	system services, user interface, general utilities
FORTRAN	FORmula TRANslation (physics calculations)
C++	object oriented general programming
Perl	interpreter, general purpose
Tcl/Tk	interpreter, user interfaces

Assembler

Traditionally, assembler has been needed for a couple of reasons:

- Access to system services and hardware
- Tuning program efficiency

Assembler programming is not generally necessary or desirable on the RISC based UNIX systems presently in use. RISC system performance is so heavily dependent on pipelining and various caches that it is extremely difficult, if not impossible, for an individual to write more efficient assembler code than is generated by the higher level language compilers. The C language provides all the hardware access and system service capabilities traditionally provided by Assembler.

C

As noted above, C has filled the programming niche traditionally occupied by Assembly language. In addition to its normal use as a high-level programming language, C can act as a universally portable Assembler.

Because the C language was created, has evolved, and has become standardized hand-in-hand with the UNIX operating system, it is the language of choice for applications involving system services and user interfaces. Using C is discussed in several of the following sections.

Now that an ANSI C standard exists and is widely implemented, portability of C code is much improved. ANSI C compilers are the default on most Fermilab systems. Likewise, adoption and availability of POSIX standards for operating system services has greatly improved program portability.

An excellent reference book for C programming is *The C Programming Language* by Kernighan and Ritchie published by Prentice-Hall.

FORTRAN

FORTRAN remains the most effective language for mathematical calculations. This is due partly to decades of research which has produced highly efficient optimizing compilers, and partly to the millions of lines of tested, portable code already in use.

C++

C++ adds object oriented programming constructs to the C language. At the risk of oversimplifying, it seems that C++ is substantially harder to learn and to use for writing new programs, but the resulting programs are much better structured, more maintainable, and more shareable than traditional C or FORTRAN programs.

An additional advantage of C++ is the availability of class[1] libraries. There is a Standard Template Library that will come with most compilers soon. This library contains many useful low level classes for such things as strings and streams I/O. Also, some vendors include other commercial class libraries as added value to their compilers. In addition to the vendor-supplied versions, C++ is available as a part of Gnu C. This has increased its popularity.

The C++ language standard is in the process of adoption by the ANSI and ISO standards committees. Both the Gnu and commercial C++ compilers have a good record of conforming to the draft standards.

Since the C++ standard is just now being finalized, there are at this time (July 1996) cross-platform porting issues. The safest bet for generating portable C++ code is to avoid using newer features such as exceptions and templates, which may be implemented differently (or not at all) by the various C++ compiler vendors, at least until the vendors catch up with the standard. As an

1. A *class* is similar to a *structure definition* in C.

alternative, use the **g++** command to run the Gnu C/C++ compiler (**gcc**) on all platforms. **g++** runs a script to call **gcc** with options that recognize C++. The Computing Division has not yet established its recommendation for a C++ compiler.

For documentation, you may refer to:

- the **CC** (upper case) command man page
- *The C++ Programming Language*, Addison-Wesley, by Bjarne Stroustrup

Perl

perl is installed at Fermilab as part of the FUE **shells** product. The man page for **perl** gives a good brief description:

> **perl** is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

There is an excellent text published by O'Reilly & Associates, Inc. on **perl**.

Tcl/Tk

Tcl is an *embeddable tool command language*, designed to provide a good interactive scripting environment both by itself and on a cooperative basis with other programs. **Tk** is an **X11** toolkit that provides the Motif look and feel and is implemented using the **Tcl** scripting language. The **Tk** toolkit is easy to use for building graphical interfaces largely because it is built on an interpreted language. **Tk** and **Tcl** are well integrated with each other, making it easy to switch between shell and graphical entry.

Tk and **Tcl** are available through **upd** (see Chapter 9).

The best reference for these languages is the book *Tcl and the Tk Toolkit*[1], by John K. Ousterhout, published by Addison-Wesley. The `README` file under the `tk` directory in `$TK_DIR` (created during setup) points to a draft of this book. Also see the man pages for information on these languages.

Other Languages and Language-Related Tools

Other tools exist that are commonly used as languages in the appropriate context. These include, for example, the various UNIX shells (as discussed in section 4.4), and **awk**, **sed**, **yacc**, and **lex**, for which O'Reilly & Associates, Inc. provides excellent texts.

There are many other languages which are not widely supported or are not in general use at Fermilab. These include Pascal, Modula-2, Lisp, Forth, Bliss, and others. We do not discuss them in this document.

---

1. Some publishers' catalogues use an ampersand (&) rather than the word "and"; check both in database searches.

13.2 Introduction to C and FORTRAN on UNIX

13.2.1 The C Compiler: cc

cc (lower case) is the C compiler command on all Fermilab-supported UNIX systems. ANSI C compilers are the default on most Fermilab systems. To compile one or more C source files (***filename.c***), you run the **cc** utility. **cc** automatically invokes the link editor **ld** unless the option **-c**, which explicitly suppresses linking, is used.

13.2.2 The FORTRAN Compiler: f77

All the Fermilab supported UNIX systems have good FORTRAN 77 compilers, which provide some minimal extensions described in *FORTRAN Compiler Evaluation of Fermilab Mandatory Requirements*, document number TN0057. These compilers also recognize most DEC-supported VAX-FORTRAN extensions.

f77 is the FORTRAN compiler on all Fermilab-supported UNIX systems. The **f77** command controls both compilation and linking functions automatically using appropriate FORTRAN runtime libraries. **f77** can produce object modules, partially linked objects, or executable programs, as appropriate. The option **-c** explicitly suppresses linking.

Note that on AIX, to get help on **f77** you need to type **man xlf** rather than **man f77**.

13.2.3 C and FORTRAN Compiling Basics

UNIX compilers, including **f77** and **cc**, generally use both filename extensions and the file content to determine how to handle the files listed on the command line. The commonly used extensions are:

For C

`c`	C source, e.g., `myfile.c`

For FORTRAN

`f`	FORTRAN source, e.g., `myfile.f`

For both

`o`	object file, e.g., `myfile.o`
`a`	archive library, e.g., `mylib.a`
(none)	executable image, e.g., `myfile`

For historical reasons the UNIX linkers produce by default an executable named `a.out`; the option **-o** ***filename*** is available to override this default.

We list here some basic compiling and linking examples. In these examples, we use a source file named `foo.c`, where `c` is the standard extension for C. In all instances shown here `c` can be directly replaced by `f`, and **cc** by **f77**, for FORTRAN.

To produce:	... enter the command:
`foo.o` object module	**cc -c foo.c**
`foo` executable from source	**cc -o foo foo.c**
`foo` executable from source + object file	**cc -o foo foo.c myobj.o**
`foo` executable from source + library	**cc -o foo foo.c $CERN_DIR/lib/libmathlib.a**

`foo` executable from source + X11/Motif (standard system libraries)

```
cc -o foo foo.c -lXm -lXt -lX11
```

The options used with the **`cc`** and **`f77`** commands are discussed in later sections.

13.2.4 Linking Order

Most UNIX linkers process source, object and library files in the order that they occur on the command line. Backward library references, from a file to an earlier library, will not be satisfied, except under AIX. It may be necessary to list a library more than once for successful linking.

AIX loads the entire content of all libraries into memory, eliminating ordering problems, but potentially creating memory usage problems. Listing a library more than once is unnecessary, and positively undesirable under AIX.

13.2.5 Displaying Active Options

You may wish to know which options are active for a given compilation, in order to verify that the defaults are what you expect. Each platform seems to provide this information somewhat differently:

Platform	Option	Result
AIX	**`-q listopt`**	Source listing with options, to file `*.lst`
IRIX	**`-v`**	Options and other details, to `stdout`
OSF1	**`-V`**	Source listing with options, to file `*.l` (FORTRAN only)
SunOS	not available	

13.2.6 Option Passing

At each stage of compilation, any unrecognized command line options are passed on to the next stage. Options that could be valid for more than one stage can be explicitly passed to a particular stage. To direct an option to a specific phase of compilation or linking, use the option **`-W`** (for AIX, IRIX, OSF1) or **`-Qoption`** (for SunOS). The phase is identified by the letter immediately following the **`-W`**. Thus, for example, **`-Wl,-m`** tells **f77** or **cc** to pass the option string **`-m`** to the **`l`** (link) phase. The **f77** command line might read:

```
% f77 -Wl,-m foo.f
```

for AIX, IRIX, or OSF1, and

```
% f77 -Qoption ld -m foo.f
```

for SunOS (where **`ld`** is the loader program)

13.3 Introduction to C++ on UNIX

CC (upper case) is the C++ compiler command on the Fermilab-supported UNIX systems that provide C++. Just as C++ is a superset of C, the C++ compilers are very similar to C compilers in that their options are usually a superset of C compiler options. The basic compiling information about C in section 13.2.3 is also applicable to C++, with the following exceptions[1]:

- Source filename extension conventions are compiler-dependent. Check the man pages for **CC** (upper case) to determine the extensions used on your system. Extensions include:

  `C` (upper case)

  `c` (lower case)

  `cxx`

  `cpp`

  `cc`

  `c++`

- The C++ compiler is invoked with an upper-case **CC**
- There are additional compiler options specific to C++; see the man pages.

The C++ compiler may not yet be installed on your Fermilab UNIX system.

13.4 C and FORTRAN Compiler Options

The default compiler options will produce a usable program on any of the supported platforms, however they may not be optimal for many situations. Several additional options are discussed in this section.

A caveat: One very annoying "feature" of the **f77** and **cc** commands is that some of the options must be specified with whitespace (at least one blank character) between the option identifier and the option value, others without whitespace and still others with or without, according to the user's choice. For instance, on IRIX systems, there must be whitespace between **-o** and the name of the executable file but there must not be any whitespace between **-l** and the library file name.

If compilers are upgraded it is possible that some of these options could change. For full details on all the options, see the man pages and the vendor compiler documentation.

13.4.1 Commonly-Used Options

-c	suppress linking, produce object file `*.o`. The linker is called as part of the compilation process by default.
-L ***directory***	add ***directory*** to the default linker search list; not needed for user libraries. The **-L** option adds directories to the linker's default search path. **-L** directories are searched ahead of system library areas. A user library could accidentally match the name of an obscure system library, with startling results.
-l*file*	search library `lib`***file***`.a`, from the default areas (the `lib` gets prefixed and the `.a` appended automatically)

1. This doesn't apply to Gnu C++. The compiler command for both Gnu C and Gnu C++ is **gcc**, and the available Gnu compiler options are different from the vendor compiler options.

With user-written libraries, specify the libraries on the command line with their true file names and a full path, without using the **`-L`** and **`-l`** options.

`-o` ***file***	(lower case **`o`**) produce executable program ***file*** rather than `a.out`
`-O`*n*	(upper case **`O`**) optimize at level ***n*** where ***n*** is 0, 1, 2, 3, or 4. The meanings of the different optimization levels vary from system to system. See the man pages for details.
`-w`	suppress informational and non-fatal compiler warnings
`-P`	(upper case **`P`**) run **cpp** (C preprocessor) only to produce `*.i` source listing
`-p`	(lower case **`p`**) enable profiling; see the man pages for details

Other commonly-used C compiler options are:

`-I` ***directory_name***	extend include path
`-D`	set value of preprocessor macro

Other commonly-used FORTRAN compiler options are:

`-u`	IMPLICIT NONE
`-C`	(upper case **`C`**) check runtime subscript range

13.4.2 Recommended Options for General Use

As mentioned earlier, the default options may not be optimal for a number of very common situations, namely for debugging, moving binary code between non-identical machines, or tuning for best performance. Even for general usage, some non-default options have proven helpful in avoiding internal compiler limits and providing better compatibility for migrated code.

This list shows options by platform which apply to all situations, and which we recommend for general use. They can be used in addition to other options you might choose based on your specific needs. As stated above, if compilers are upgraded it is possible that some of these options could change, so you should always consult the man pages.

AIX	**`-qextname -qrndsngl`**
	`-qflttrap=invalid:overflow:zerodivide:enable`
IRIX	**`-trapuv -lfpe`**
OSF1	(none)
SunOS	**`-xl -fnonstd`**

Discussion of General Options

`-qextname`	(AIX) append underscore (**_**) to external names (default at Fermilab). This is needed if you use any Fermilab or CERN libraries. Better to just make it a habit.
`-qrndsngl`	(AIX) standard IEEE math on intermediate `REAL*4` results. This produces bitwise identical results to IRIX/OSF1/SunOS for most programs, making port testing easier.

-qflttrap	(AIX) detect and report floating point errors. Without this, errors occur silently, and incorrect results may be produced. Unlike other platforms, these would not be NaN[1], but legitimate floating numbers. Do not use the **:imprecise** option here, as it appears to disable error detection entirely (AIX 3, XLF 2).
-trapuv	(IRIX) set uninitialized variables to NaN, to help catch nonportable code and latent bugs at runtime.
-lfpe	(IRIX) use the Floating Point Exception library, to report or core dump on errors. Without **-lfpe**, errors produce NaN values silently. Must be combined with the *TRAP_FPE* environment variable, or calls to `handle_fpes`, to be effective. *TRAP_FPE* must be set to the value: `OVERFL=ABORT;DIVZERO=ABORT;INVALID=ABORT` See section 8.1 for setting environment variables.
-xl	(SunOS) Extended Language, for Fermilab-required extensions (see section 13.2.2)
-fnonstd	(SunOS) trap floating point errors.

13.4.3 Debugging Option

The **-O**'s here are all upper case, for optimization.

-g		include full symbol table for debugger. This option interacts with the optimizer differently on each platform, so we provide some usage notes:
	AIX	you should not use **-O** with **-g**
	IRIX	**-g** forces optimization off; use **-g3** to permit optimization
	OSF1	**-g** changes default optimization to **-O0** (dash, letter-O, zero) An additional OSF1 option, **-synchronous_exceptions**, puts traps after each floating point operation. This is very slow, but will precisely locate floating point errors. Use this only for code which is known to produce floating point exceptions.
	SunOS	you should use **-O1** with **-g**

13.4.4 Portability Option for AIX

-qnomaf	Turns off Multiply/Add instructions in order to enhance portability.

13.4.5 Speed Optimization Options

Note that on most platforms a combination of options is required for best optimization. Recall that these options may be used in addition to general recommended and other options. The **-O**'s here are all upper case.

AIX	**-O2**
IRIX	**-O3 -mips2**
OSF1	**-O3 -feedback -tune host**
SunOS	**-O3 -cg92 -libmil**

1. NaN is the IEEE standard Not-a-Number bit pattern produced when a floating point operation generates invalid results.

Discussion of Speed Optimization Options

Platform	Options	Comment
IRIX	**-mips2**	Use full R4000 instruction set. This is important on R4000 and later systems.
OSF1	**-feedback** **-tune host**	Order routines for best cache performance, based on actual program execution statistics. Use best optimization for given generation of chip. The code will run anywhere, but is tuned for local speed.
SunOs	**-cg92** **-libmil**	SuperSPARC V8 instruction set Sparcstation 2 and later. See the man fpversion document. Hardware-specific inline math

13.4.6 Load Map Option

The load map option is actually a linker option. On AIX, IRIX and OSF1 this requires passing to the link phase the option that controls production of a load map.

Platform	Option	Output
AIX	**-Wl,-bmap:*file***	***file***
IRIX/OSF1	**-Wl,-m**	`stdout` (SysV style list of input/output)
IRIX/OSF1	**-Wl,-M**	`stdout` (BSD style primitive map)
SunOS	**-m**	`stdout`

Additionally, under AIX two related options are available:

-Wl,-bloadmap provides additional internal loader information

-Wl,-bxref provides additional cross reference information

13.4.7 Special FORTRAN Compiler Options

Source Code Listing Option

Each platform has an option that produces a source code listing. The file extensions for these listings are platform-dependent.

Platform	Option	Listing Extension
AIX	**-qsource**	`lst`
IRIX	**-listing**	`L`
OSF1	**-V**	`l`

Platform	Option	Listing Extension
SunOS	**-Xlist**	`lst`

Saving Local Variables

The FORTRAN 77 standard allows subprogram local variables to become undefined between calls, unless saved with a SAVE statement. Many UNIX **f77** compilers *require* the SAVE statements for retained local variables. SAVE'd variables are usually called *static*, and unSAVE'd variables are called *automatic*. For programs not yet properly equipped with SAVE statements, **f77** command line options are available as follows:

Platform	Static Option	Automatic Option
AIX	(default)	**-qnosave**
IRIX	**-static**	**-automatic** (default)
OSF1	**-static** (default)	**-automatic**
SunOS	(default)	**-stackvar**

C Preprocessor

The C preprocessor (**cpp**), a macro processor, is used in **f77** for conditional compilation, macro expansion and source inclusion. If you plan to run **cpp**, remember to double any backslash characters (****) in your code to prevent their misinterpretation as **cpp** escapes. The option shown in parentheses is the default option for the corresponding platform:

IRIX	(**-cpp**) **-nocpp**
OSF1	**-cpp** (**-nocpp**)
SunOS	any input file name `*.F` is automatically preprocessed by **cpp**
AIX	(see below)

The C preprocessor cannot be invoked through the **f77** command on AIX systems. It can, however, be invoked under AIX as a separate command:

% /lib/cpp -P -C ***infile outfile***

See the man pages for **cpp** syntax and usage information.

Extend Search Path for INCLUDE

Use the **f77** command line option **-I** ***path*** to extend the search path for the INCLUDE statement on AIX/IRIX/OSF1 (not supported on SunOS), where ***path*** is the path to the directory where the include files are found.

Internal Compiler Limits

You may need to set one of these if you are compiling a very large single source file.

AIX	**`-NQ20000`**
	Larger internal compiler tables. This value should be big enough for almost all programs.
IRIX/OSF1	**`-Olimit 1500`**
	Allows somewhat larger routines to be optimized.

13.5 FORTRAN Programming

There is some additional information about using FORTRAN in the UNIX environment that you will find useful to know.

13.5.1 External Reference and Entry Point Names

In order to avoid conflicts with the C runtime library when FORTRAN and C programs are included in a single program, most UNIX **f77** compilers internally append an underscore to FORTRAN external references and entry point names. At Fermilab we have set up all **f77** compilers to do this by default.

13.5.2 Separate Compilation of FORTRAN Subprograms: fsplit

By default, most **f77** compilers pre-link all the source code being compiled, even when you specify the **`-c`** option. If you compile a library with a single **`f77`** statement, it will usually contain a single module, and be linked as a whole.

The **fsplit** utility identifies and extracts subprograms from the original FORTRAN source file into individual files in the current directory. These files can then be compiled separately so that they retain their identity when assembled into a library.

The names of the extracted individual files are taken from their corresponding subprogram names. On some systems **fsplit** will overwrite any pre-existing file, including the original source file, whose name matches any of the subprogram names.

See the man pages for more information on **fsplit**.

13.5.3 AIX-Specific Issues

You must `CALL EXIT` to avoid having a `STOP` message printed on `stdout` at program termination.

The `OPEN` statement parameter `READONLY` is unavailable. All files are opened with `WRITE` access. This makes file sharing problematic under AIX.

13.5.4 Loading Block Data Modules

Many UNIX **f77** compilers enforce the standard restriction that variables in COMMON must be initialized only in BLOCKDATA subprograms.

To ensure the loading of BLOCKDATA subprograms from libraries, declare the BLOCKDATA program name as EXTERNAL in some important module which you know will be loaded.

13.5.5 Program Control

Command Line Arguments

A FORTRAN program can easily evaluate arguments included on the command line that runs the program. A couple of examples follow.

- The IARGC function returns the number of command line arguments:

  `N = IARGC ()` Sets `N` to the number of command line arguments

- The GETARG subprogram returns the value of a specified argument:

  `CALL GETARG(I , STR)` Puts the `I`'th argument into string STR

Environment Variables

The GETENV subprogram provides the values of environment variables. For example, to copy the value of variable `MY_OUT` into string `OUTFILE`, include in your source file:

```
CALL GETENV ( 'MY_OUT' , OUTFILE )
```

Printing

The usual FORTRAN carriage control characters placed in the first column of formatted output files are not interpreted by most UNIX text handling utilities. Use the UNIX **asa** utility to convert such FORTRAN output files to an equivalent standard ASCII text form. **asa** handles blanks, 0, 1 and + in column 1, removing any other characters. See the man pages for **asa** for details.

13.5.6 FORTRAN Source Code Analyzer: FLINT

FLINT is a FORTRAN source code analyzer, a proprietary product of IPT. At Fermilab, **FLINT** is available for SunOS 5 and IRIX 5. The **FLINT** User Manual for UNIX is available from the Computing Division library, document number PU0099. The manual is on-line in the file $FLINT_DIR/MANUAL wherever **FLINT** is installed.

FLINT provides several analysis features:

- local and global analysis
- cross reference tables
- program statistics
- calling trees

FLINT reports the following error conditions:

- inappropriate arguments passed to functions or subroutines
- inappropriate library calls
- inconsistencies in common-block declarations
- non-portable code
- type usage conflicts across different modules
- unused functions, subroutines and variables
- variables which are referenced but not set

FLINT Options and Files

Use the **-S** (upper case) option to specify an output file name. The default extensions listed below are used. A separate file will be created for each component of the listing according to the additional option shown:

`lnt`	Analysis output
`xrf`	Xref table (**-x**)
`tre`	Call tree (**-t**)
`stt`	Statistics (**-s**)

Use the **-L** option to create a `*.lbt` **FLINT** library, containing all subroutine interface information needed for subsequent call usage tests. This library may be specified as **FLINT** input in place of the original source.

Use the **-B** option to create a `*.fdb` database file, containing all the symbol information needed for subsequent cross reference and call tree analysis. This database file may be specified as **FLINT** input in place of the original source.

Lowercase options may be combined. Use a double dash to disable an option (e.g., **--w**); this is useful for disabling a default option.

Uppercase options take parameters (with or without a space between the option and the parameter, depending on the option) and do not combine.

When original source is not available, `*.lsh` ASCII files may be created to describe calling sequences. The file `$FLINT_DIR/unixlib.lsh` describes the standard UNIX system calls.

FLINT Usage Examples

To see the standard analysis warnings for Isajet, enter:

```
% flint $ISAJET_DIR/isajet.f
```

To see a calling tree of the Isajet program, without warnings, enter:

```
% flint -t -Tcondensed --w $ISAJET_DIR/isajet.f
```

To see an options summary, enter:

```
% flint "-?"
```

13.5.7 Future FORTRAN Enhancements

AIX XLF 3

This document describes the AIX XLF 2 compiler. IBM's standard **f77** compiler, XLF 3, is not yet available at Fermilab.

FORTRAN 90

The FORTRAN 90 standard includes FORTRAN 77 as a subset, and makes standard many of the extensions in common use. FORTRAN 90 is not yet commonly installed at Fermilab. This document is written for **f77** users.

13.6 Obsolete Programming Features

You may encounter these features in older code.

AIX XLF

Some years ago the AIX FORTRAN compiler command was **`xlf`**. The standard **`f77`** command is now available under AIX, and should always be used in place of **`xlf`**. However, as mentioned earlier, you still need to type **`man xlf`** to access the man pages.

Calling BLOCKDATA

Some systems (VS-FORTRAN) required an explicit call to each `BLOCKDATA` routine if you wished to force loading of that routine from a library. This is not necessary on any supported UNIX system.

BUFIO

The Fermilab **bufio** product for accessing raw variable length records on tape and disk is no longer supported. Improved capabilities are being supported in **RBIO** and **DAFT** (see sections 16.2 and 16.3).

RANLIB (SunOS/4)

The RANLIB utility added necessary library symbol tables under SunOS/4. This is done automatically under SunOS 5.

ar -s Option (ULTRIX)

The **`ar -s`** option added necessary library symbol tables under Digital's ULTRIX[1] and some earlier operating systems. This is done automatically under all Fermilab supported systems.

13.7 C and FORTRAN I/O

This section mainly applies to FORTRAN. For C, all you need to know is that the **RBIO** and **DAFT** libraries mentioned below are available.

Recall that file names in UNIX are case sensitive. It is customary to use lower case for normal files, reserving upper case names like README for documentation and control files.

Note that you cannot use the shell metacharacter tilde (**~**) to specify a home directory within a C or FORTRAN program; **~** is valid only on a UNIX shell command line (for all shells but **sh**). **`logdir`** can be used within programs for this purpose (see section 6.1.2).

13.7.1 Records

The UNIX operating system treats a disk file as a sequence of bytes. Interpretation of data as records is entirely up to individual applications. The FORTRAN I/O libraries provide the necessary record handling for FORTRAN programs. READ statements return only the content of the records, and not the control words mentioned below.

Formatted records are terminated by a new-line character **<CTRL-L>**, consistent with other UNIX text handling programs.

Unformatted records are both preceded and followed by a 32 bit integer containing an exclusive byte count.

---

1. ULTRIX has been superseded by OSF1.

13.7.2 Tapes

Tapes of course have real physical records, and must be handled differently than disk files. A tape file is sometimes called 'character special' to indicate that it is not accessed on a character-by-character basis. Tape handling is covered in Chapter 16.

There are no industry-wide standard tape I/O routines callable from FORTRAN. Fermilab makes use of several locally-written packages, including:

RBIO	labeled and directoried tape access (see document SU0033)
DAFT	raw binary tape access (see document PN0504)

You can find C and FORTRAN examples in Appendix B of the *DAFT User's Guide*, document PN0504.

13.7.3 Standard Input and Output

In conformance with the FORTRAN standard, `READ`s and `WRITE`s to unit `*` are directed to `stdin` and `stdout`. You can `READ` and `WRITE` to units 5, 6, and 0 without an `OPEN`. They are preconnected to `stdin`, `stdout`, and `stderr`, respectively. If you `OPEN` and write to any other unit number `#` without specifying a file name, a default name of `fort.#` will be used.

13.8 Performance Tuning for C and FORTRAN

13.8.1 Optimization

Using the compiler **`-O`** (upper case) options can improve program execution speed by factors of 3 or more, depending on the application, over unoptimized code. Note that your libraries must also be compiled at the same level in order for this to be effective.

Beware that there are some optimizer bugs. You should always do a limited run initially with and without optimizer options, and check your answers.

For production running, use the appropriate hardware-specific optimizations for the systems running the code. These options typically tune for cache sizes, instruction sets, and other internal hardware features, resulting in sizeable speed gains. On some systems this produces an executable that will run only on the targeted architecture.

It is common practice to retain debugger symbol tables in production programs, with only a small speed penalty. You may have to exercise care that the **`-g`** option does not also disable optimization of such production programs. Under IRIX, you must use **`-g3`** to get both optimization and symbol tables.

See the suggested speed optimization options, and vendor documents for details.

Floating Point Errors

You can obtain substantial speed increases on some systems, especially AIX, by disabling the detection and trapping of floating point errors such as overflow, division by zero, and invalid values.

On the systems with the biggest gains, this practice can produce apparently normal, but incorrect, results. For example, 1000./0. can produce the result 1000. It is hardly necessary to point out that this sort of thing can produce surprising physics results! For this reason our recommended options for general use are set to at least detect and report floating point errors.

Qualifiers which force precise trapping of floating point errors are generally only used when tracking down known problems, as they can impose a large performance penalty.

13.8.2 Word Length

It may be tempting to use arrays of short words to 'save memory'. On previous generations of computers this could also speed execution. On RISC systems there is a big performance penalty for this practice.

The current generation of RISC processors are optimized for 32 and 64 bit operations. Operations on 8 bit or 16 bit words are performed several times more slowly. The processor must extract the necessary data into a longer word, perform the operation, and mask the result back into the original location.

Alignment of variables is important for the same reason. A misaligned 32 bit word requires even more shifting and masking than a 16 bit word, with an even greater performance penalty. If you must combine different length variables in a data structure such as a COMMON, place longer words earlier in the data structure.

13.8.3 Feedback

The speed of a program can be as much limited by memory access as by processor speed. Effective use of memory cache is critical to getting good performance.

Cache usage can depend on the details of the linking process. Arbitrary changes in the ordering of modules in the executable can result in nearly 20% differences in execution speed, for typical physics code. Small changes like switching between static and shared libraries, or modifying a single subroutine call in your code, can result in substantial changes in linking order and hence in performance.

For this reason, some vendors provide mechanisms for setting optimized module ordering in the executable, based on data from a trial run of the application. Under OSF1, use the compiler **`-feedback`** option; see the **f77** or **cc** compiler man page for details and examples.

13.8.4 Inlining

Many compilers provide options for replacing calls to external modules with equivalent inline code, to permit better optimization and reduce subroutine call overheads.

Physics code does not generally benefit measurably from such inlining. Inlining within a library makes the inlined modules nonreplaceable at link time, leading to confusing results and difficult debugging. In our recommended speed optimization options we stop short of the levels that introduce inlining.

13.9 C and FORTRAN Mixed Programming

It is possible, with a little care, to combine C and FORTRAN modules in the same program. Some of the issues that need attention include:

- variable types
- array indexing
- external names
- arguments
- commons
- I/O
- linking

For newly written C programs, you may wish to use the `cfortran.h` header file available in the **cern** product.

The July/Sept 1994 *CERN Computer Newsletter* (217) contains an extensive tutorial by Alfred Nathaniel on C and FORTRAN interfacing, available locally under the heading **Software Development** from the *UNIX Resources* WWW page.

We give here a summary of the techniques used on the Fermilab UNIX systems. In Appendix I you can find an example that illustrates much of the information in this section.

13.9.1 Variable Types

Generally, these variable types are equivalent:

FORTRAN	C
INTEGER*1	char
INTEGER*2	short
INTEGER*4	int
REAL	float
REAL*8	double
LOGICAL	(unavailable)

C strings are zero-terminated, and have no intrinsic length. FORTRAN character variable lengths are given by an internal descriptor. FORTRAN character variables passed to C routines should be copied and zero-terminated before they are used.

The internal representation of FORTRAN LOGICAL variables is usually non-0/0 for .TRUE./.FALSE. respectively, but it is best not to count on this.

13.9.2 Array Indexing

By default C starts indexes at 0 and FORTRAN starts them at 1. C and FORTRAN multiple index ordering is reversed. FORTRAN substring selection appears as the first C string index. See the following equivalence table:

FORTRAN	C
`intv(j)`	`intv[j-1]`
`intv(j,k)`	`intv[k-1][j-1]`
`char(j)(k:k)`	`char[k-1][j-1]`

13.9.3 External Names

By default on Fermilab UNIX systems, the **f77** compiler modifies FORTRAN subprogram and other external names. It forces each name to lower case, and appends an underscore. Thus FORTRAN label `SUBPROG` would become C label `subprog_`.

13.9.4 Arguments

FORTRAN subprogram arguments are always passed as addresses (C pointers). C programs can specify arguments as either pointers or values. FORTRAN CHARACTER arguments are passed as pointers, followed by a set of additional values (not pointers) at the end of the argument list, giving the length of each CHARACTER argument.

C routines can always call FORTRAN routines, with due attention being given to arguments. FORTRAN routines cannot call arbitrary C routines.

13.9.5 Commons

FORTRAN COMMON's are accessible in C as *extern struct*'s, with the same name mapping as is used for entry points.

FORTRAN	C
`COMMON /FOO/ I`	`extern struct { int i ; } foo_ ;`
`K = I`	`k = foo.i ;`

It is best to keep your FORTRAN COMMON variables aligned on natural boundaries[1], in order to avoid potential padding words which may be inserted differently by various FORTRAN and C compilers. You get natural alignment easily by putting longer variables before shorter variables in the COMMON.

13.9.6 I/O

Mixed C/FORTRAN I/O to the same file is not advisable. Mixed C/FORTRAN I/O to `stdout`, where `stdout` is the terminal, will usually work reasonably well, making debugging easier.

13.9.7 Linking

The easiest and safest way to link C/FORTRAN programs is to use the **`f77`** command, which automatically includes both C and FORTRAN run time libraries. If you insist on linking with the **`cc`** or **`ld`** commands, remember to add the options:

```
-lF77 -lI77 -lm
```

13.10 Executing a Program

Once you create an executable, you run it the way you do a normal UNIX command, that is by typing its name followed by appropriate options or parameters.

You must be aware that if you have not included "dot" (`.`) in your path, whenever your executable is in a directory not explicitly included in your path, you will need to prefix the executable name with **`./`** to run it. This was also mentioned in section 8.2 under *PATH*.

---

1. Keep all ***n***-byte variables' addresses an exact multiple of ***n,*** for example 0, 4, 8,... for a 4-byte quantity.

13.11 Debugging

13.11.1 dbx

dbx is a utility for source-level debugging and execution of programs written in C, C++, and FORTRAN**. dbx** allows you to trace the execution of a specified object file. You can step through a program on a line-by-line basis while you examine the state of the execution environment.

Programs compiled with the **-g** option of **cc** (and other compilers) produce an object file. This object file contains symbol table information, including the names of all source files that the compiler translated to create the object file.

dbx also allows you to examine *core files* via its **where** command. A core file contains the core image produced when the object file was executed, providing information about the state of the program and the system when the failure occurred. A core file named `core` is produced by default.

dbx commands can be stored in a start-up `.dbxinit` file that resides in the current directory or in your home directory. **dbx** executes these commands just before reading the symbol table.

Running dbx

To invoke **dbx**, enter the following command:

% dbx [options] [*object_file* [*corefile*]]

where ***object_file*** is the name of the file you want to debug.

Once **dbx** is running, you should see the **(dbx)** prompt. At this point you can start issuing **dbx** commands.

Commands

There are many **dbx** commands, all described in the man pages. Some of the basic commands are **run**, **where**, **print**, **stop**, **list**, **cont**, and **quit**:

run [*arguments*]	Begin executing the object file, passing optional command-line ***arguments***. The arguments can include input or output redirection to a named file.
where [*n*]	List all active functions on the stack, or only the top ***n***.
print [*expressions*]	Print the values of one or more comma-separated ***expressions***. To print values of two-dimensional FORTRAN array elements use the format: **print *array_name*[1,2]**
stop *restriction* [if *cond*]	Stop execution if specified ***restriction*** is true. Restrictions include (this is a partial list): **at** (source line) ***n*** **if *cond*** **in** (procedure or function) ***func*** ***cond*** (condition) is a Boolean expression; if it evaluates to true, then execution is stopped.
list [*n1* [,*n2*]] or **list *func***	List the source text between lines ***n1*** and ***n2***, or on lines surrounding the first statement of ***func***. With no arguments, list the next ten lines.

cont [at *n*] [sig *signal*]	Continue execution from the point at which it was stopped if no arguments. Resume from source line ***n*** or, if a ***signal*** name or number is specified, resume process as if it had received the signal.
status [> *file*]	Show active **trace**, **stop**, and **when** commands
delete [*n*]	Remove traces, stops, and whens corresponding to each command number ***n***, given by **status**. If ***n*** is **all** remove all.
quit	Exit **dbx**.

Example

You may want to start by using **dbx** to set some break points within your code. To step through your code at the very beginning, you need to stop in the `MAIN` routine if you are debugging an object file created from FORTRAN source code (stop in `main` if your source is in C language). For example, you would type:

```
(dbx) stop in MAIN
```

Now you can issue the **run** command to start execution of your object file. You will get the process id and the name of the object file being executed.

```
(dbx) run
```

At this point, you may use the **list** command for the first 10-line listing of the source code:

```
(dbx) list
```

Use the **stop** command to set break-points at various lines or procedures within the object-file:

```
(dbx) stop at 10
```

```
(dbx) stop in sub123
```

```
(dbx) stop in sub456 if i == 24
```

The execution will stop in the example above at line 10, or in subroutine `sub123`, or in subroutine `sub456` when `i` is equal to `24`. To continue execution at any point in your debugging, issue the **cont** command:

```
(dbx) cont
```

To restart your debugging session, issue the **rerun** command:

```
(dbx) rerun
```

To exit **dbx**, type **quit**:

```
(dbx) quit
```

13.11.2 purify

purify is a commercial product that detects memory corruption and finds memory leaks in your executable programs. **purify** is currently available on the FNALU node FSUI01. The command to run it is **purify**.

See the man pages for information on its syntax, options and uses.

13.11.3 CASEVision

CASEVision is an advanced debugging environment for SGI. It is currently available on both of the SGI interactive nodes on FNALU. The command to run CASEVision is **`cvd`**.

See the man pages for more information. Additionally, documentation is available through the SGI on-line documentation tool **insight** (see section 3.1.3).

Chapter 14: The make Utility

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. Although **make** can be used in a wide variety of applications, in this chapter we concentrate on its use in the area of software development. We describe how to define relationships between source, object, library and executable files for use by **make**, and how to invoke **make** in its simplest and slightly more complex forms.

14.1 An Overview of the make Utility

make is a command execution utility. You can use it to essentially automate any task in which one or more "target" file(s) requires updating via a shell command when changes have been made to any of its "required" files (files from which the targets are built). There is some preparation to do, but once that is complete, all you do is enter the **`make`** command!

make compares the modification dates of target files to those of their required files. For each file that needs updating, **make** issues the predefined update command(s) for the file. For example, if file A is a required file of file B (the target), and if file A has a more recent "last modified date" than file B, then **make** *re-makes* file B by issuing the specified update command(s). Target files that are found to be more recent than all their required files are skipped over.

make is especially useful in long-term software development projects that involve large numbers of source files, libraries, and executables connected by a complex set of relationships. You can use it with any programming language whose compiler can be run with a shell command.

make obtains information about the files, their relationships, and the specific update commands from one or both of the following:

- a specially formatted, user-supplied control file called the *Makefile* (see section 14.2)
- **make**'s set of built-in default rules (see section 14.6)

In preparing to use **make**, you generally need to write a Makefile. The Makefile defines the relationships among the constituent and target files of your project by listing the required files for each target file, and stating the shell commands that must operate on the required files to create or update the target(s). **make** implicitly treats all required files as targets, in an iterative manner. A file listed as a required file in one definition statement may also explicitly appear as a target in another statement. For example, in a program, typically the executable file (the final target) is created from object files, which are in turn created by compiling source files. Once you have a working Makefile, you just run the **make** program to perform all your updating tasks.

The **make** man pages give a full description of the command, its options and features, as well as the format and usage of the Makefile.

We have included two examples in Appendix I that illustrate many of the points discussed in this chapter.

14.2 The Makefile and its Components

The Makefile is a blueprint that you design and **make** uses to create or update one or more target files based on the most recent modify dates of the required files. The **make** command line syntax remains quite simple in the case where a makefile is used:

% make [-f *makefile_name*] [*other options*] [*targets*]

If the **-f** option is not used, **make** checks for a Makefile of a particular name. It first looks in the current directory for a file named `makefile`, then for `Makefile`. If the Makefile is still not found, **make** looks in a couple of other places (see the man pages). In order to keep things simple and standard, we recommend that you always use the filename `Makefile` for your Makefile(s). Following this convention, you'll have only one Makefile in a directory. Your Makefile can contain instructions for building many different targets. When executing **make**, you can specify the desired target(s) on the **make** command line.

The Makefile can contain four types of definitions:

macro	A macro is a name that you define to represent a variable that may occur several times within the Makefile.
target	A target definition lists the target file, its required files, and the commands to execute on the required files to produce the target.
suffix rules	Suffix rules indicate the relationship between target and source file suffixes. For example in FORTRAN, object files (`*.o`) are created from source files with a suffix of `f` (i.e. `*.f`). If no source file is explicitly given on a target definition line, **make** uses suffix rules to determine what source file to use to produce the target.
suffix definitions	If your Makefile uses a suffix in a suffix rule, that suffix must be listed as a required file in a .SUFFIXES declaration in the same Makefile. A suffix used in **make**'s built-in rules is not automatically recognized in your own Makefile.

Each new line in the Makefile starts a new definition, except that in target definitions:

- shell commands with leading tabs are part of the previous definition (a standard Makefile format that you need to recognize, but that we suggest you avoid using for reasons explained in section 14.2.2.)
- shell commands can be continued in standard UNIX format, using a trailing backslash (\)

Blank lines are permitted between definitions.

14.2.1 Macros

A macro is a name that you define to represent a variable that may occur several times within the Makefile, or that needs to be updated frequently. Macros make maintenance of your Makefile much easier. They are commonly used to define settings, platform-specific commands, lists of required files for a target, lists of command options, and so on. **make** reads all the macro definitions before executing any commands. It is often convenient to put all the macro definitions at the head of the Makefile, but this is not necessary.

Format and Usage

Macro definitions are of the form:

macro_name* = *value

where ***macro_name*** is the name you want to use in place of the longer ***value.*** Everywhere in the Makefile that ***macro_name*** is found, **make** substitutes ***value***. For portability, if ***value*** contains any blank spaces, it must be enclosed in quotes in the definition statement.

Once a macro is defined, you refer to its value in the form `$(`***macro_name***`)`. If ***macro_name*** is a single character, you can omit the parentheses.

As an example, let's define a macro FFLAGS to set some FORTRAN default options to use with the **`f77`** command:

```
FFLAGS = "-O2  -w -Olimit 1500 -nocpp "
```

You can now refer to the value of FFLAGS within the Makefile in the form `$(FFLAGS)`. For example, you might include a target definition line like the following (the format is explained below in section 14.2.2):

```
foo : foo.f ; f77 -o foo $(FFLAGS) foo.f
```

Special Macros

To be sure that **make** uses the standard, portable Bourne shell, always include in your Makefile a macro of the form:

```
SHELL = /bin/sh
```

Some versions of **make** default to your current interactive shell if you don't include this explicit SHELL macro in your Makefile. The standard Makefile format that we describe in this chapter assumes **sh** as the command interpreter.

Macro Sources

Macro definitions are similar to and take default values from environment variables. **make** gets additional macro definitions from the following sources, and applies them in the order of precedence shown:

1) command line

2) Makefile

3) current environment

4) built-in rules

14.2.2 Targets

Targets are the files that you want to update or create. A complete target definition includes the name of the target, the files needed to build it (its required files), and the commands that must be executed to recreate the target.

Format

The standard Makefile format using Bourne shell conventions calls for:

- a space between listed targets
- a colon (`:`) between the last target and the first required file
- a space between listed required files
- a semi-colon (`;`) after the last required file if commands follow on the same line
- a semi-colon (`;`) between the successive listed commands

A simple target definition with a single target, required file, and command is of the form:

target : required_file ; shell_command

or, listing the command on the next line (note that the semicolon (**;**) is omitted here):

target : required_file

{tab}*shell_command*

In standard Makefile format, the commands beyond the first line of a definition must have leading tabs, and there must be no intervening blank lines. There are two reasons to avoid this format: first, when working with the Makefile it is hard to see the difference between a tab and blanks, and secondly, some editors change tabs to blanks (or vice versa), which causes problems (neither **emacs** in default mode nor **vi** changes tabs to blanks). You can check your file to see if it contains tabs or blanks by running the command:

cat -tev ***filename***

We propose as an alternative that you use the backslash character (****) to continue the single definition line down as many physical lines as you have to go. Also, note that under AIX the standard "tabbed" format can be unpredictable; it is therefore safer for *several* reasons to use our suggested format on this platform.

This "safer" format which we suggest if the entire definition doesn't fit on the first line is:

***target : required_file ; ***

shell_command

A target definition line can contain more than one of each element type. Or, it may contain only two of the three element types. A more complex definition in our suggested format looks like:

```
target_1 target_2 ... : required_file_1  required_file_2 ... required_file_n \
required_file_n+1 ...; \
shell_command_1 ; shell_command_2 ; shell_command_3 ; ... ; \
.
.
.
... ; shell_command_m
```

Usage

The relative modification times of the ***target*** and the ***required_file(s)*** determine whether the listed commands will be executed. If the target file is found to be missing or to be older than any of its required files, **make** executes the commands to rebuild it. **make** treats the required files iteratively as targets, whether or not they are explicitly listed as targets in subsequent target definitions, and rebuilds them as necessary before rebuilding the final target.

14.2.3 Suffix Rules

A suffix rule is much like a target definition except that it uses implied rather than explicit file names for target and required files. A suffix rule is of the form:

implicit_required_file_suffix implicit_target_file_suffix : ; command

As an example, assume you have a set of target files to rebuild whose filenames are all of the form `*.b`. The required files for these targets have filenames of the form `*.a`. Define the suffix rule:

`.a.b : ;` ***command(s)***

make expands this to the following target definition for all files ending in `.a` in the current directory:

filename*`.b :` *`filename`*`.a ;` *command(s)

Note that suffixes don't have to start with a dot (`.`).

14.2.4 Suffix Definitions

In order for the example suffix rule in 14.2.3 to be recognized, your Makefile must also contain a .SUFFIXES declaration of the form:

`.SUFFIXES : .a .b`

You may combine all the suffixes you use in the Makefile into one .SUFFIXES declaration, or group them into separate statements.

14.2.5 Control Files within a Makefile

You may encounter situations where it is useful to pipe input or output of one command to another within a Makefile. You can echo a small control or data file within the Makefile, rather than maintaining a separate external file. In the following target definition example from an Isajet Makefile, several control statements are echoed into the **patchy** utility in order to extract `isaint.f` from the `isajet.car` patchy library:

```
isaint.f : isajet.car ; \
(echo "+USE,*ISAJET,$(MACHINE).              "; \
 echo "+USE,INTERACT.     INTERACTIVE PATCH  "; \
 echo "+EXE.                                 "; \
 echo "+PAM,T=C.                             "; \
 echo "+QUIT.                                ";)\
| ypatchy isajet.car  isaint.f  \& genint.lis .GO
```

14.3 Running make

14.3.1 General Usage

The **make** utility is invoked with the **`make`** command. The command syntax is:

`% make [`*`options`***`] [`***`targets`***`]`**

Several ***options*** are available, and are described in the man pages for **make**. We provide a list of some of the commonly used options:

`-n`	Preview the commands, don't execute. Very useful for testing.
`-d`	Debug; list the operations used and why ("read **make**'s mind"). Available on all platforms except OSF1.

-f *makefile_name*	If your Makefile has a name other than `makefile` or `Makefile`, use this option followed by the file's name to identify it (leave a space between the option and the filename).
-p -f /dev/null [|less]	Print the built-in rules (see section 14.6).

The ***targets***, as mentioned earlier, are the files that you want to create or update. **make** searches the Makefile to find a target definition statement for each target listed on the command line.

14.3.2 Usage without Specifying Target

When **make** is invoked without a specified target, the first target in the Makefile is used. The command is simply:

% make [*options*]

For larger products, it is standard practice to name this target *all* in the Makefile, and in the list of required files to list the individual targets which together actually produce the full product. For example, the first Isajet target definition is:

```
all : isadecay.dat \
      isatext.doc  \
      isajet.a     \
      isaint       \
      isasusy
```

Notice that here no commands are listed. They would appear in the subsequent target definitions.

14.3.3 Usage without a Makefile

The extensive built-in rules let you use **make** quite effectively without having your own Makefile. Section 14.6 provides a brief explanation of the built-in rules. **make** will look for any file whose name matches that specified on the command line and which has a file extension that identifies it as a reasonable source. For example, to produce the executable `foo` (the target) from a `foo.c` or `foo.f` source that exists in the current directory, enter:

% make foo

make will look for the C or FORTRAN file as the source file for this target. Taking the FORTRAN program and no options as an example, this command is equivalent to (see section 13.2.3):

% f77 -o foo foo.f

14.4 "Housekeeping" Targets

It is common practice to have a "housekeeping" target which removes stray files from the working directories. Typically you would run **make** on this target after you've completed the **make** operation on your principle target(s). It is conventional to call this target *clean*.

Here is an example which removes unnecessary generated files from several different directories. The target definition has no required files:

```
clean: ; \
  rm  -f *.bak    ;\
  rm  -f *.lis    ;\
  rm -rf Maketemp ;\
  cd example/isaplt ; rm -f *.lis*   ;\
  cd  ../jet        ; rm -f jet.log*
```

You need to determine what stray files will be generated in your case, and define your commands accordingly. Run **make** on the *clean* target by entering:

```
% make clean
```

You may wish to define different levels of housekeeping targets. One that clears out everything, leaving only the original files you had before running **make**, is often named *clobber*.

14.5 Portability

It is desirable for your Makefile to be portable across different UNIX platforms.

Why might this be a problem to implement? As mentioned earlier, **make** does all macro processing before any commands are executed. Therefore environment variables set in shell scripts executed by **make** have no effect on **make**'s macro definitions. And standard **make** doesn't support conditional macro definitions. So, how can you write a portable Makefile?

A Solution

1) Create a script for setting environment variables.

2) Have **make** run this script ("source" it; see section 4.4) from within the Makefile.

3) After it runs the script, have **make** rerun itself with a different target and the original command line options. A *$(MAKE)* macro which causes **make** to rerun itself is a standard feature.

An example of this technique follows.

Example

Create a portable shell script (named, for example, `Makeenv`) which sets appropriate environment variables for the Makefile. Here is a simple `Makeenv` script which defines the macro `F77` based on the current platform as determined by the command **`uname -s`**:

```
export F77 MACHINE

case `uname -s`  in
IRIX)
        F77="f77  -O2  -w -Olimit 1500 -nocpp "
;;
OSF1)
        F77="f77  -O1  -w -Olimit 1500 -nocpp -static "
;;
esac
```

The Makefile is below. Run **make** with the target `isaint`. The Makefile runs `Makeenv`, then **make** reruns itself with the target `do_isaint` and the correct `F77` value:

```
isaint    : isaint.f ;  . Makeenv; $(MAKE) do_isaint
do_isaint : isaint.f ; $(F77) isaint.f  -o isaint
```

Other Utilities

There are other utilities available for more complicated cases:

- **gmake** (Gnu make), part of the Fermilab **gtools** product, has some nice portability features, and supports other advanced features like parallel compilation on multiprocessor systems.
- There are preprocessors for building locally tailored Makefiles in very sophisticated ways, including **gnu configure**, **premake**, and **imake**.

14.6 make's Built-in Rules

make comes equipped with a long list of built-in defaults to make your job easier. You are free to override any of them in your Makefile. The defaults fall roughly into four categories:

To build a given target, **make** executes a list of commands, called an *action*. Actions may be listed explicitly in the target's `Makefile` entry, or supplied implicitly by **make** via a *built-in rule*.

1) macros that reflect your current environment variables

2) macros that define standard compilers and options

3) suffix rules for finding required files when building targets

4) a list of known suffixes

To get a listing of all the built-in macros and rules, enter the command:

```
% make -p  -f /dev/null [| less]
```

Depending on your platform, there may be nearly a thousand lines of definitions, so you might want to pipe this to **less**, or redirect the output to a file.

14.7 A Few Caveats...

1) Recall that in the standard Makefile target definition format successive commands are entered on successive lines, each starting with a tab. Be aware that each of these command lines runs in a different shell. Two important implications of this are:

   a) if you have issued a change directory (**`cd`**) command, it does not carry over to the following command(s)

   b) environment and shell variables do not carry over to the following command(s)

   The format which uses a single logical line for the entire definition avoids this problem.

2) If you use non-shell commands (for example **`ls`**) in definition statements, be aware that the output may vary from platform to platform. For this reason it is best not to rely on the specific output format of these commands.

Chapter 15: Code Management

This chapter introduces the two recommended code management solutions for UNIX, **CVS** (Concurrent Versions System) and **UCM** (UNIX Code Management). Both use **RCS** (Revision Control System) as the underlying protocol. We provide basic information only, and refer you to the complete manuals for these utilities for detailed information.

RCS provides a version control system with which you can record the history of your source files. An **RCS** file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. Only the differences between versions are kept.

CVS and **UCM** implement the **RCS** features differently, but both assume availability of **RCS** commands. **CVS** allows concurrent development, whereas **UCM** was created to replace **CMS** (a VMS source code management system) and implements the same sequential development philosophy of that system. Both systems provide easy extraction of either a release version or the latest version, and they allow you to create tags for release versions of the software.

See *Code Management* under the **Software Development** heading on the *UNIX Resources* WWW page for more information.

15.1 CVS

The **CVS** product allows many programmers to work on the same code simultaneously, each in his or her own directory, and it merges the changes when they are finished. There is no built-in mechanism to prevent concurrent development.

CVS stores all files in a central repository, a directory populated with a hierarchy of files and directories. The files are organized in modules, where a module is made up of one or more files, and can include files from several directories. It is typical to define one module per project. Although the structure of the repository and modules file interact with your build system (e.g., Makefiles), they are essentially independent.

15.1.1 Accessing CVS and Obtaining the Manual

To set up the **CVS** product:

```
% setup cvs
```

The **CVS** document (in postscript format) can now be found in the directory pointed to by `$CVS_DIR/doc`. Document number GU0009 is also available on WWW from the Computing Division home page under *General Documentation*.

CVS points to the editor defined in your *EDITOR* variable for entering log messages (see the **`cvs commit`** command below). If it is not set, the editor defaults to **vi**.

15.1.2 Basic CVS Commands

Normally you never access files in a repository directly; you use the **CVS** commands to get your own copy of files. The common commands and their functions are listed below (we do not provide information on options here; see the document referenced above):

% cvs import	installs new release of source in **CVS** repository the first time
% cvs co ***module***	creates new directory called ***module***, populates it with source files; this allows you to "checkout" your own working copy of the source ***module***
% cvs checkout ***module***	equivalent to **cvs co** ***module***
% cvs commit	commits changes you have made; opens editing session for entry of log message
% cvs rtag ***release_number module***	tags a release
% cvs export -r ***release_number module***	extracts the specified release without **CVS** administrative directories

15.2 UCM

UCM is a replacement for **CMS** which has been widely used at Fermilab on VMS systems. To enforce sequential development, it uses a reservation system to ensure that a second programmer cannot change a module that another programmer is currently editing.

Note that **UCM** contains tools which extend its use beyond the scope of simple source code management. Two of these tools, **kpp** and **uvmbld**, are discussed below.

A **UCM** library is a single UNIX directory created via the "create library" command listed in section 15.2.2, below. A **UCM** library is designed to be a simple UNIX version of a **CMS** library; its organization and commands are directly analogous to **CMS** libraries and commands. A **UCM** library is automatically created with the subdirectory `/UVM`, which in turn is populated with a transaction log `/UVM/HISTORY` for the library.

A **UCM** element is specified by a library name and a file name (`libraryname/filename`), and consists of **RCS** "version" files located in the **UCM** library's top level directory.

A **UCM** group is a group of elements which can be referenced by the group name. The list of elements comprising a group is maintained in a file called `UVM/`***groupname*****`.GRP`**.

Some of UCM's components are:

- **uvma** (UNIX Version Management Access) is a **Perl** script layered over **RCS** to create and control access to libraries of **RCS** files (the source code repository).
- **uvmi** (UNIX Version Management Inquire) is another **Perl** script which returns information about the **UCM** libraries and files.
- **kpp** is a text preprocessor written in generic C, whose use is much wider than just code management. See section 15.2.3.
- The **uvmbld** utility is designed to simplify the process of updating target files such as archive libraries or executables by automatically generating a Makefile (see Chapter 14) from a list of **UCM** library elements that satisfy a specified wildcard pattern, or that are members of a specified group. The command and its options are described fully in the **UCM** manual.

- The **CHECKIN** call provides a hook which can be used to perform various checks or operations on **UCM** elements before they are inserted into a library. The default version of **CHECKIN** creates an include file dependency list which is used by **uvmbld** when creating Makefiles.

15.2.1 Accessing UCM and Obtaining the Manual

To access **UCM**, you first need to set it up.[1] Include in your login files or enter:

```
% setup ucm
```

Once it is set up, you can find the **UCM** manual in the directory pointed to by `$UCM_DIR/doc`. In addition to providing more in-depth information on the concepts and commands introduced here, it provides examples of updating include file directories, updating an object library, and maintaining release versions. A plan for a structuring a complete working system is also presented.

15.2.2 Basic UCM Commands

You use **UCM** by executing commands provided by two of its constituent utilities, **uvma** and **uvmi**. The commands associated with these utilities can operate on groups of elements, either matching a wildcard pattern or belonging to a named group. The utilities and their commands are described in detail in the **UCM** manual. As mentioned above, the **uvma** commands are used for creating and maintaining libraries, and the **uvmi** commands provide information to the user.

Some common **uvma** commands and their functions are listed below:

% uvma create library ***libraryname***
creates a source code library

% uvma create element ***elementname***
creates an element in the library

% uvma fetch ***elementname*** fetches the requested element

% uvma reserve ***elementname*** reserves the element[2]

% uvma replace ***elementname*** replaces the element

% uvma tag ***elementname*** **-n** ***tagname***
tags a release

% uvma group ***elementname*** **-g** ***groupname***
adds an element to a group

Simply typing **uvma** or **uvmi** without arguments will produce a full list of supported commands and their syntax. This also provides information on the available options.

15.2.3 kpp

The **UCM** product contains **kpp**, a preprocessor written in generic C which handles conditional text and include files in a language-insensitive manner, and is designed to replace the more important functions of **Expand**[3]. **kpp** is much like **cpp** (the C preprocessor) but has been

---

1. On some systems, you will have to setup **CVS** to access **RCS** before using **UCM**. This is not a problem where FUE is installed.
2. Elements must be reserved before they can be changed.
3. A Fermilab program which handles include files and machine dependencies.

generalized so as to be equally effective on FORTRAN code and document files. It is therefore useful in applications besides those involving code management. **kpp** is available for AIX, IRIX, OSF1, SunOS, and VMS. The executable is easily built for other platforms by:

```
% cc -o kpp $UCM_DIR/src/kpp.c
```

The **kpp** command is of the form:

% kpp ***infile outfile*** **-d** ***expand_flag*** **-i** ***include_directory***

The command and its options are described in the **UCM** manual. **kpp** directives are also discussed.

Chapter 16: Tape Handling

In this chapter we introduce you to the principal tape handling software and facilities available at Fermilab, and refer you to detailed documents on each subject for more information. Since many groups and experiments have standardized and/or customized their tape-handling routines, some of the options described here may not be available or appropriate for you. Contact your group leader to find out what procedures have been established for your group.

16.1 Operator Communications Software (OCS)

16.1.1 Accessing OCS and Obtaining the Manual

OCS is a package that performs and manages tape drive allocations, operator-assisted tape mounts and tape drive use statistics. Its logical-to-physical tape device name translation helps not only human communication, but hides many platform specific idiosyncrasies from users. It is not a tape I/O package, but works well with such packages.

You can access **OCS** in three ways:

- FORTRAN/C library of subroutines
- Command line tools
- X Motif tools

The features available in each of these interfaces overlap to a great extent. However, since the different interfaces are by their nature geared towards different uses, the functionality is not 100% duplicated across them. The FORTRAN/C subroutines provide the most flexibility and functionality for users requiring multiple mount requests, the commands are generally used in shell scripts, and the X interfaces are very useful for monitoring tape drive statistics.

Documentation on these tools and **OCS** in general is available wherever **OCS** is installed. To access the currently installed version of **OCS** and its documentation, enter:

```
% setup ocs
```

The environment variable *$OCS_DIR* is now defined. In the directory `$OCS_DIR/doc` you will find several documents files in PostScript or ASCII formats. Two of them in particular will be useful to you for getting acquainted with the system:

`overview.ps`	GA0015 *OCS Overview* (Postscript)
`reference.ps`	GA0012 *OCS Reference Guide* (Postscript)

They are also available also in the Computing Division library and on WWW.

In this chapter we provide a reference for the most commonly used **OCS** commands (section 16.1.3), but we refer you to the above-mentioned documentation for information on the full set of commands and on the FORTRAN/C subroutines. The X interfaces are fairly intuitive, so we give you enough information to bring them up (section 16.1.4), but we do not describe them in detail.

16.1.2 OCS Basics

During the course of your work, you may need to monitor different aspects of the job and possibly contact the operators. **OCS** provides functionality to perform these additional activities. For your convenience, we list the appropriate command next to the function in the table below, and indicate whether you can perform the function using one of the X interfaces. You will need to see the *OCS Reference Guide* for usage information on these commands and for the associated FORTRAN/C subroutines.

Function	Command	X interface
Mark a tape drive broken so that it will not be allocated until it is repaired	**ocs_broken**	**xocs**
Log statistics as to how much the drive was used	**ocs_report_stat** **ocs_init_stat**	
Send an attention message to the operator	**ocs_message**	
Send a request to the operator to run a cleaning tape through a tape drive	**ocs_clean_it**	
Display status of tape drives in the OCS database	**ocs_tape**	**xocs**
Display pending mounts	**ocs_pending**	**xocs** **xtapeview**
Display tape drive statistics	**ocs_devstat** **ocs_stats**	**xocs**
Display tape mount log	**ocs_mrlog**	**xocs**
Display tape drives that may need to be cleaned	**ocs_clean_list**	**xocs**

Here's an outline of a typical user process. The commands associated with these steps are described in section 16.1.3.

1) Display the list of tape drives that you are authorized to allocate (this step is not always needed; it is included for completeness).

2) Request allocation of a tape drive so no other user may access it.

3) Send a request to operations staff to mount a tape on a drive you have allocated, wait for the reply, and verify that the tape was mounted correctly.

4) Based on the tape drive name, request the appropriate device file for reading/writing the tape.

5) Dismount the tape (e.g., rewind and unload it), and optionally request a new tape mount.

6) Deallocate the tape drive so that someone else can use it.

16.1.3 Reference for Selected OCS Commands

In this section we provide explanations, option lists and examples of the basic **OCS** commands that would be used in our "typical user process" outlined in section 16.1.2. The commands appear in the order you would generally use them.

Keep in mind that as new versions of **OCS** are released the command options may change. You can always find up-to-date documentation in *$OCS_DIR*.

ocs_tape

The **`ocs_tape`** command is used for various tape functions. For example, you could use this command to display the list of tape drives you are authorized to allocate. The available options are as follows (defaults shown in parentheses):

`-t`	device group (any)
`-d`	device type (any)
`-a`	allocation (any)
`-s`	status (any)

Examples:

`ocs_tape -o long -h any`	List all tape drives on all hosts, using the long format.
`ocs_tape -o short -a unallocated`	List all unallocated tape drives on the local host, using the short format.
`ocs_tape -a allocated -d 8505`	List all allocated 8505 tape drives in the default (long) format.
`ocs_tape -s broken`	List all broken tape drives on the local host.
`ocs_tape -s working -t CDF_devices -h any`	List all working tape drives in the group *CDF_devices.*

ocs_allocate

The **`ocs_allocate`** command allows you to allocate a tape drive for your own use. Options are (defaults shown in parentheses):

`-d`	device type (any)
`-T`	preferred device (none)
`-t`	device group (any)
`-h`	host group (local host)
`-m`	must get (no)

Examples:

`ocs_allocate -T` ***drive*** **`-m`**	Allocate a specific tape drive, fail if not available.
`ocs_allocate -T` ***drive***	Allocate a tape drive on the local host, preferably the specified drive.
`ocs_allocate -t CDF_drives -h any`	Allocate a tape drive from device group *CDF_drives.*
`ocs_allocate -d 8505`	Allocate an 8505 tape drive on local host.

ocs_setdev

Currently needed only for AIX systems, but may be used on any platform.

The **ocs_setdev** command is used to set device characteristics of tape drives. It may be used to set device characteristics on remote nodes. However, the user *must* have the device allocated before setting its characteristics.

Examples:

ocs_setdev -t ***drive*** **-nv** Set device characteristics to variable block with no compression

ocs_setdev -t ***drive*** **-f 4096 -c**

Set device characteristics to fixed block (4k) with compression.

ocs_request

The **ocs_request** command is used to request a tape mount, and also to send special instructions to the operators along with tape mount requests.

Examples:

ocs_rquest -t ***drive*** **-v** ***tape*** **-r**

Request specified tape be mounted read-only on specified drive.

ocs_request -t ***drive*** **-v** ***tape*** **-w**

Request specified tape be mounted on specified drive, write-enabled.

ocs_request -t ***drive*** **-v** ***tape*** **-w -o "Tape mismounted: make sure write-protected"**

Same as above, but send this message to operators.

ocs_check_tape

The **ocs_check_tape** command checks whether the tape in the tape drive has the appropriate protection, and, optionally, the correct tape label.

Examples:

ocs_check_tape -t ***drive*** **-r**

Verify that a tape has been properly mounted on the specified drive, without checking that the label is correct.

ocs_check_tape -f ***host*** **-w -v** ***tape***

Verify that specified tape has been properly mounted, with label verification.

ocs_pending

The **ocs_pending** command returns “yes” or “no” indicating whether or not a mount is pending on the specified drive.

Example:

ocs_pending -t ***drive*** Determine if a mount is pending on specified drive.

ocs_cancel

The **ocs_cancel** command is used to cancel a pending mount request.

Example:

ocs_cancel -t ***drive***	Cancel a pending mount request for specified drive.

ocs_devfile

ocs_devfile is used to get correct special file for a particular set of device characteristics. The options for this command are (defaults shown in parentheses):

-t	Tape Drive (none)
-s	Byte Swapping (no swap)
-f	Fixed-length records (variable)
-r	Auto-rewind on close (no rewind)
-n	Auto-retention on open (no retention)
-d	8200 or 8500 (8500)

ocs_dismount

The **ocs_dismount** command can be used to rewind a tape and optionally unload the tape from the drive. On operating systems or tape drives that do not support unloading, the option to unload will be ignored and the tape will simply be rewound. Options are (defaults shown in parentheses):

-t	Tape Drive (none)
-o	(unload) or nounload
-h	Host (only needed if two or more drives have the same name but are located on different hosts)

Examples:

ocs_dismount -t ***drive***	Rewind and unload the tape from the drive.
ocs_dismount -t ***drive*** **-o nounload**	Rewind a tape drive without unloading (not recommended, especially for drives in FCC).

ocs_deallocate

The **ocs_deallocate** command is used to deallocate tape drives that have been previously allocated. Options are (defaults shown in parentheses):

-a	all drives owned on local host (none)
-A	All drives owned on all hosts sharing the same **OCS** database (none)
-t	Tape Drive Name (none)

Examples:

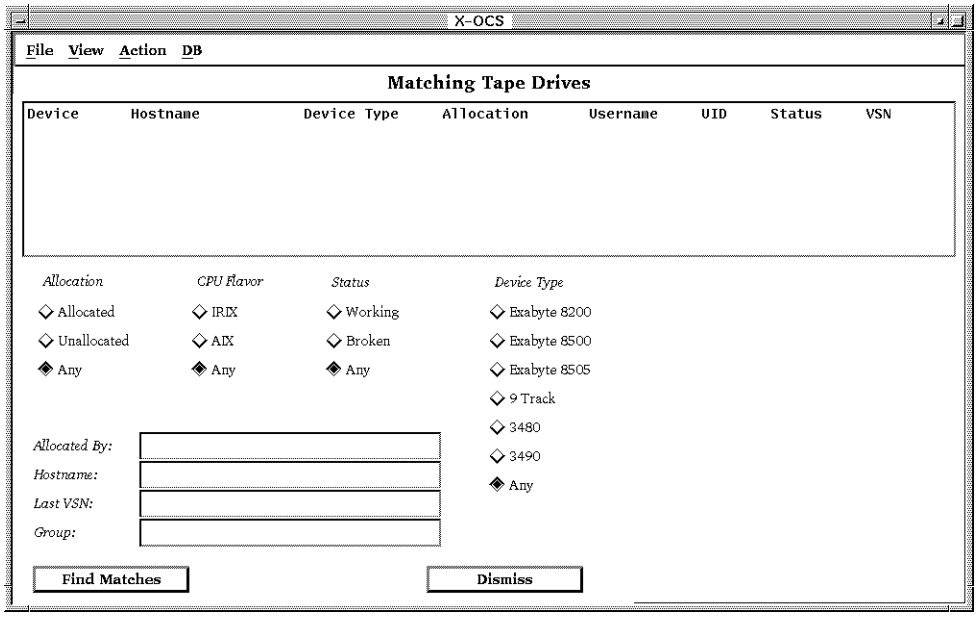
ocs_deallocate -t ***drive***	Deallocate specified tape drive.
ocs_deallocate -a	Deallocate all tape drives owned by me on the local host.
ocs_deallocate -A	Deallocate all tape drives on all hosts.

16.1.4 The OCS X Interfaces

Remember that your *DISPLAY* environment variable needs to be set properly for X windows applications (see section 8.2).

xocs

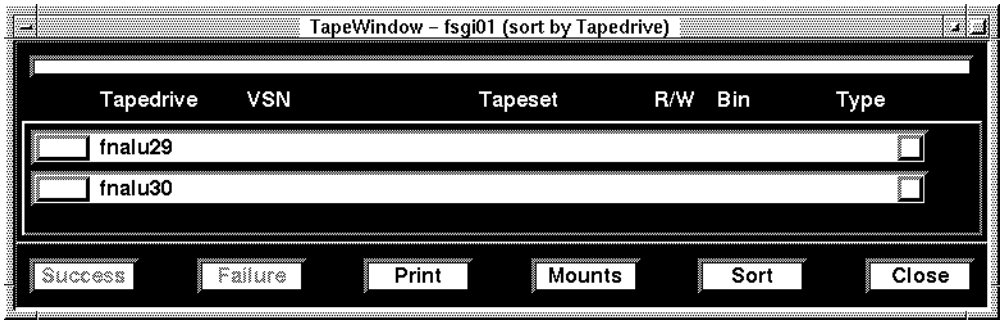
xocs is an X interface that you may find useful for viewing tape statistics and history. It allows you to perform a subset of the functions available through text commands. After setting up **OCS**, enter **`xocs`** at the command line. You'll see the following screen:



The menu options under **VIEW** and **ACTION** include most of the features you will need.

xtapeview

Another X interface to **OCS** which allows you to view pending mount requests is **xtapeview**. As a user, you are not permitted to respond to mount requests, only view them. Invoke this interface with the command **`xtapeview`**. A window will appear from which you can choose a node. Click on the node you want, then you'll see a window like the one below which shows the drives associated with that node, and any pending mounts.



If you have a color screen, the banner at the top is blue when no mounts are pending, and red if one or more are. The status button (to the left of the drive name) will be red if a mount is pending on that drive. If you don't have a color screen, there are grey-tone representations of blue and red. See `$OCS_DIR/doc/xtape.ps` (recall that *$OCS_DIR* is defined during setup) for a full description of this application, and to see how to customize the color scheme.

16.1.5 Using Provided Examples to Get Started

Further examples are provided in the `$OCS_DIR` directory (defined during setup). First print yourself out a copy of `$OCS_DIR/doc/viewgraphs.ps` for more single-command examples and their expected output.

The directory `$OCS_DIR/examples` provides sample programs for FORTRAN, C and Bourne shell script which are intended to help you understand how **OCS** works. The executables are named `ocs_ctest`, `ocs_ftest`, and `ocs_btest` for C, FORTRAN and Bourne shell, respectively. Each program carries almost identical functionality. You may want to use the source of these programs (`ocs_ctest.c`, `ocs_ftest.fs` and `ocs_btest.sh`) as an aid in developing your own programs. The `$OCS_DIR/examples/README` file explains how to use the examples.

These example programs can actually send a mount request to operations staff: please keep this in mind if you experiment with OCS on Fermilab Computing Division systems such as FNALU.

16.2 Raw Buffered I/O (RBIO)

The **RBIO** (Raw Buffered I/O) library provides a FORTRAN-callable, medium-independent I/O interface across UNIX platforms. It allows the FORTRAN program to perform buffered I/O on a file object, which may be a regular file or a labeled tape. **RBIO** reads and writes ANSI standard as well as VMS labeled tapes. It does not explicitly support unlabeled tapes. Disk I/O in the "unformatted" FORTRAN format is supported. The actual reading/writing level is at the primitive logical record level; **RBIO** does not understand record contents.

The **RBIO** library is used in the compilation, linkage, and execution phases. See the *RBIO User's Guide* (document number SU0033) for usage information.

16.3 DAta From Tape (DAFT)

DAFT (DAta From Tape) consists of a set of routines written by the DART collaboration to provide event-level reading and writing from DART-formatted records to/from disk and tape (see section 17.3 for information about DART).

DAFT performs the following functions:

- retrieve data written on tape or disk in the format used by DART
- write data on tape or disk in the DART format

The routines in **DAFT** are widely portable, and currently run on the supported UNIX platforms IRIX, SunOS, and AIX. They offer a FORTRAN interface.

DAFT was written in two layers to allow for a lower level that is easily portable. Therefore, two sets of access routines are provided:

- a "low-level" set that embeds the different levels of platform-dependence and delivers records to the higher level routines
- "high-level" routines that unpack events from the records and provide the event interface

The DART on-line format packs events into logical records, where each record may contain more than one event, in order to keep the drives streaming at full speed. The routines handle large events spanning records (the maximum record length is 64 Kbytes), and correct the byte ordering of data on platforms of different *endianship*.

For more information on **DAFT**, see the *DAFT (DAta From Tape) User's Guide*, document number PN504. As newer documentation on **DAFT** becomes available, it will be accessible through the DART WWW pages.

Chapter 17: Computing Resources and Projects

It is our intention in this chapter to make you aware of resources available through the Fermilab Computing Division, and projects of general interest to Fermilab UNIX users. In particular, we discuss FNALU, CLUBS and DART. We provide references in case you wish to investigate any of these topics further.

A note to CDF and D0 members: Please refer to documentation specific to your experiments regarding use of these resources.

17.1 General Purpose UNIX Resource: FNALU

FNALU is a UNIX computing resource managed by the Computing Division which is available to the laboratory. It provides batch and interactive computing on IBM, Silicon Graphics, Sun, and DEC platforms.

Accounts on the FNALU cluster fall into two categories, based on the user's resource requirements. If your resource requirements are small (< 50Mbytes/1% CPU), you can request an account using the *Computer Account Request Form*. To request an account allowing the use of resources beyond this limit, you need to submit an *FNALU/CLUBS Project Request Form* on which you indicate your estimated resource requirements. Both forms are in the FORMS folder, accessible from the Computing Division home page or in the Computing Division library (WH8NE). Read the *Non-Grant Use of FNALU and CLUBS* (also in the FORMS folder) before filling out the appropriate form. Send your completed form to *compdiv@fnal.gov*, or deliver it to the computing division library at WH8NE.

Refer to the *FNALU User's Guide*, document number GU0008, for a detailed discussion of the hardware and software that make up or are available on FNALU. The most recently updated version of this document can be found on WWW.

17.1.1 Intended Uses

FNALU is intended to be a general-purpose, multi-platform cluster with a focus on physics and engineering requirements. The intended uses of FNALU can be summarized into a few categories:

- development machine for CLUBS (see section 17.2)
- software repository/server and code development platform for work groups in a distributed environment
- platform for proprietary products available for users on a trial basis
- platform for users with low requirements who cannot purchase UNIX system
- distribution platform for Computing Division products and software products used directly by client systems participating in the AFS file system
- mail reader

17.1.2 Hardware

The FNALU cluster is composed of separate interactive, batch, and file serving nodes. The currently available nodes are[1]:

Vendor	O/S	Interactive Nodes	Batch/Translator/File Server Nodes
IBM	AIX	FIBI01	FIBB01
Silicon Graphics	IRIX	FSGI01 FSGI02	
Sun Microsystems	SunOS	FSUI01 FSUI02	FSUT01 FSUS01, 02, 03, 04, 05
DEC	OSF1	FDEI01	

As a guide to interpreting the node names, the first two letters after the F represent the vendor (e.g., SG=Silicon Graphics), the next letter indicates the use of the node (I=interactive, B=batch, T=translator, S=file server), and the last two digits identify the node uniquely if the rest of the designation is shared by other nodes.

This list is subject to change. For the latest list, and more information on the machines (including performance information), see the FNALU hardware document on WWW accessible from *UNIX Resources* under the heading **Systems**.

We recommend that you use FSUI02 as a general login node, and FSUI01 for Oracle applications.

17.1.3 File System and Application Software

FNALU's primary file system is AFS. All systems participating in the AFS file system have the same view of the file system. The files for a set of machines known as a *cell* exist under `/afs/`*`cellname`*. The cellname for Fermilab is `fnal.gov`, so all AFS files are located under `/afs/fnal.gov`. Other cells are accessible such as the CERN cell `/afs/cern.ch`. The AFS system is discussed in section 2.6 and in section 17.1.4 below.

FUE is installed on FNALU, and it handles products in a way that appears the same to all users of the cluster, regardless of platform.

At this time we provide a limited set of text processing tools and no other office automation functions on FNALU. ORACLE is available on FSUI01. Future expansion plans include more text processing software, and some other commercial software packages.

17.1.4 AFS on FNALU

AFS is a modern implementation of distributed file serving. It has several advantages over older file systems such as NFS. The FNALU cluster is running AFS at Fermilab as a distributed file service model in a production environment. AFS file permissions are addressed in section 6.6.2.

Your Kerberos token may expire if you've been logged in for over 24 hours. If this happens, just run the command **`klog`**. This is explained in section 2.8.2.

---

1. Additionally, there is an IRIX batch node called FSGM01 which is currently operational, however it may get shut off in the near future.

Advantages of AFS

We see the advantages of AFS in our environment as being primarily in the areas of:

- performance
- consistency of file organization over distributed servers
- authentication of users (security)
- server management
- volume replication and backup
- server function redundancy

The advantages in each area are discussed in the *FNALU User's Guide*, document GU0008.

AFS and the UNIX find Utility

As mentioned in section 6.4.1, the **`find`** command uses many system resources. When using it on an AFS system, you may accidentally end up searching servers all over the world if the top of the search is at **`/`** (root). **Be careful to search only the part of the tree that interests you.**

17.1.5 FNALU Batch System: LoadLeveler

FNALU provides a basic batch environment for common tasks. If your job requires the processing of a large amount of data, you may want to investigate the CLUBS system (see section 17.2).

The batch system used on FNALU is built upon IBM's **LoadLeveler**.[1] As an advanced scheduling system, **LoadLeveler** matches job requirements to machine resources. **LoadLeveler** runs as a set of daemons (background processes) on each participating node controlling access to the computing resources of participating systems, known as the **LoadLeveler** pool.

A batch job consists of a user shell script (the default shell is Bourne) and a **LoadLeveler** command file. The user shell script describes the steps for processing the desired programs while the **LoadLeveler** command file describes the necessary conditions for the execution of the shell script to the **LoadLeveler** job scheduler. You can have separate user shell scripts and **LoadLeveler** command scripts or you can combine them into the same file. Examples of each are provided in the *FNALU User's Guide*.

The queues currently defined on FNALU are **short** (10 min), **medium** (30 min), **long** (4 hr), and **exlong** (96 hr). You select your queue by specifying a *class* in your command file.

For further information on **LoadLeveler**, see the man pages.

fbatch with LoadLeveler

The FUE utility **fbatch** was developed as a user interface to **LoadLeveler** to allow execution of the **LoadLeveler** commands for local and remote batch systems. **fbatch** uses **rsh** commands to access the remote batch system, therefore the user must either have proper entries in his or her `.rhosts` file (see section 12.1.3), or the system administrator must have proper entries in the `/etc/hosts.equiv` file.

Note that some FNALU nodes, namely the Sun and DEC machines and some of the SGIs, do not support **fbatch** commands. Currently you must submit your jobs from FIBI01 or FSGI02. The jobs will run on FSGM01, FSGI01, FSGI02 or FIBB01.

---

1. The batch system on FNALU will be changing to **LSF** in the near future.

To run batch jobs on FNALU, you must execute **`setup fbatch`** initially. The format of the command to submit a batch job on FNALU is:

`% fbatch_submit [-b fibi01] -c` ***command_file***

where ***command_file*** is the name of the **LoadLeveler** command file.

This command and the others available in the **fbatch** utility are described in the *FNALU User's Guide* and in the man pages.

17.1.6 Tapes: OCS System

Tape mounts on FNALU are requested with the Operator Communications Software (OCS) system. See Chapter 16 for more information.

17.2 Clustered Large UNIX Batch System (CLUBS)

CLUBS (Clustered Large UNIX Batch System) is a scalable RISC-based UNIX cluster that combines a batch processing environment with a strong data serving platform. CLUBS includes high performance I/O capabilities that are not available with a basic batch system, and eliminates the burden of error handling associated with 8mm and other slow serial media, thereby providing a reliable central non-interactive facility for jobs with high I/O data requirements.

CLUBS offers several advantages:

- A hierarchical storage management system provides mass storage capabilities to be used for scientific computing experiments in which many people require access to the same data
- The high I/O capability allows second-pass splitting and filtering of data from the fixed-target experiments.
- A feature that allows the saving of bulk data makes CLUBS well-suited to Monte Carlo generation.[1]
- A highly managed environment is provided for predictable job turn-around.

CLUBS is tightly integrated with FNALU and hence makes use of the Fermilab AFS cell. To access CLUBS, you must request an account on it, although you typically don't log into CLUBS directly. Your CLUBS account is designed to be accessed through an account on FNALU. Generally, the two accounts will share the same username and AFS password. See section 17.1 for information on requesting an account; the forms and process are the same as for FNALU.

Several CLUBS documents are available on WWW. A full discussion of CLUBS is available in the *CLUBS User's Guide*, document number GU0007.

17.2.1 Integration of FNALU and CLUBS

FNALU and CLUBS are tightly integrated to provide a common home area for users. **fbatch**, discussed in section 17.1.5, is the batch scheduler system used by both CLUBS and FNALU. You do not actually log into the CLUBS system, but rather you access it via FNALU, which provides the home area for your applications, libraries and control files. All of the AFS defined areas on FNALU are available to the CLUBS system, and all of the products and versions maintained on FNALU are exported to CLUBS. You should use FNALU to test and debug your applications interactively prior to submitting jobs to CLUBS.

---

1. Most large Monte Carlo will have to be done on farms simply because of the massive amounts of CPU required.

CLUBS uses the same user and group ids as FNALU. This allows **fbatch** to provide a transparent environment for your batch job, running it as though you were logged on to your CLUBS account. When you submit a job, you are required to provide your AFS (Kerberos) password which CLUBS stores in an encrypted form and then uses it to authenticate your batch job to AFS when it is scheduled to start. This gives your batch job the same permissions as a normal login process.

17.2.2 CLUBS Components

The CLUBS system runs IBM's batch product **LoadLeveler**[1] (using the **fbatch** interface) and IBM's Hierarchical Storage Management (HSM) product **NSL Unitree**[2] in addition to several CLUBS-specific systems and interfaces. CLUBS incorporates high speed networking, an import/export server with extensive error handling, and high speed robotics for data storage. It includes the following hardware:

- IBM and SGI RISC-based UNIX computers
- High speed robotic tape library managed by the HSM, providing a reliable mass storage repository
- High speed network providing throughput of 100 MBytes/sec between CLUBS components
- Several types of tape drives (8mm, 9-Track, 3480)[3]

The three node types in CLUBS are the following:

Compute Worker — These nodes provide computing power for user batch jobs. The CLUBS machines and FNALU both use a common AFS file system for the user's home directories and for products.

For each batch job local work disk space is provided for the duration of the batch job.

This system uses the standard Fermilab network between CLUBS and FNALU, and the high speed network for bulk data transfer between CLUBS components.

Import/Export — The import/export subsystem is designed to separate the user's batch job from the import and export of data to and from low performance serial media. Users do not interface directly to the low performance serial media or to the import/export subsystem. The import/export processes are invoked by commands used within a batch job.

Data Server — The Hierarchical Storage Management (HSM) system provides fast data serving for large bulk data by means of a reusable high speed disk and tape cache. The HSM is comprised of several computers, each with a high speed disk and a connection to the high speed robotic tape library.

17.2.3 CLUBS Batch Process Overview

Each submitted job is reviewed by the CLUBS scheduler (node FNCLUB). If necessary, the scheduler invokes a separate, independent task on the import/export subsystem to obtain the required data from serial media. Import/export places a copy of the retrieved data into the HSM via the high speed link. A series of error handling and other processes is then invoked to read the data in the HSM. Meanwhile, the job remains in the hold queue on the scheduler until all the required data is made available in the HSM and a compute worker node is available for processing.

---

1. CLUBS will soon be using **LSF** instead of **LoadLeveler**.
2. **HPSS** will replace **Unitree** in the near future.
3. Limited support for DLT in special cases is planned for the future.

At this point the job is released, and is scheduled on a compute worker node. The data server copies the data from the HSM to the local disk of the compute node via the high speed data link. The job can now access the data at local disk access speed. This technique also allows multiple jobs running on the cluster to access the same data. The user's batch job can save bulk data it has created, and make it available to other jobs or export it to an 8mm tape.

17.2.4 Elements of Using CLUBS

In order to use CLUBS effectively, you will need to become familiar with a few utilities, and you will need to learn how to write the user shell scripts and **LoadLeveler** command files. In accordance with the purpose of this chapter, we provide only a brief overview of these processes, and refer you to the *CLUBS User's Guide* and its referenced documents for further information.

Utilities

fbatch	**fbatch** is a FUE product tailored to IBM's **LoadLeveler** batch system and developed to allow users to execute batch commands on a local (FNALU) or remote (CLUBS) system. **fbatch** is discussed in section 17.1.5. Many of the CLUBS features interface to **fbatch** commands.
needfile	The **needfile** system provides two functions. If the user's data does not yet exist in the HSM, **needfile** tells the system to copy the data there from its original medium, e.g., 8mm tape. **needfile** then provides access to this data for the user. This data remains in the HSM based on a least-recently-used policy. There is no guarantee of how long data will remain in the HSM.
tcache	The **tcache** system is a set of tools that allows users to save bulk data for later retrieval. The data is stored in the HSM. Whereas **needfile** is a read-only system, **tcache** is a facility to read and write data to the HSM with a specified retention period.
clubstools	This FUE product includes: **clubsout** (creates 8mm cassette from data files saved as **tcache** sets), **ndflist** (lists available **needfile** data tapes), **libclubs.a** (set of FORTRAN and C callable functions), and **jobstat** (interrogates the CLUBS system log file).

Running a Batch Job in CLUBS

The process of running a batch job in CLUBS from FNALU includes the following steps, the last two of which are not always necessary:

- Run **`setup fbatch`** and **`setup clubstools`**
- Select batch class according to required CPU (CPU calculation based on 50 VUP machine)
- Select platform and O/S version
- Prepare shell script and command file
- Submit job using the command **`fbatch_submit -b fnclub -c`** ***`command_file`***
- Monitor job using the command **`fbatch_q -b fnclub [-l`** ***`jobnumber`***`]`
- Cancel job using the command **`fbatch_cancel -b fnclub`** ***`jobnumber`***

Note that some FNALU nodes, namely the Sun and DEC machines and some of the SGIs, do not support **fbatch** commands. Currently you must submit your CLUBS jobs from FIBI01[1].

1. We hope to have **LoadLeveler** installed soon on FSGI02 and FSUI01. This will allow you to submit batch jobs from these nodes as well.

17.3 Data Acquisition System (DART)

DART is a UNIX-based data acquisition system developed by a collaboration between the Fermilab Computing Division and a number of Fermilab experiments. The system was designed to meet the requirements of these experiments that are taking data in the 1996-1998 fixed target run. These requirements include:

- rates into level 3 of up to 20KHz (trigger rate) and 200 Mbytes/sec (data rate)
- on-line event filtering CPU power of up to 3000 MIPS for acceptance ratios of up to 1 in 40 events
- data logging rates from 1-20 Mbytes/sec
- support for distributed intelligent processing

To meet these needs, DART provides a parallelized, scalable, VME-based readout, and a fully distributed control and monitoring architecture. DART provides a common integrated set of hardware and software using well-established technologies and techniques. The system hardware and software architecture is simple enough for small experiments, yet extensible and fast enough for larger ones. DART was designed and implemented such that most components can be used as-is for most experiments, but can be easily extended or tailored for individual experiment needs. Both UNIX and the real-time operating system VxWorks are fully supported.

17.3.1 DART Functionality and Supported Interfaces

DART encompasses a wide range of features and functions:

- parallel readout from FASTBUS, CAMAC, FERABUS, and other custom buses into VME
- scalable event buffering in VME
- scalable data flow from VME to UNIX boxes
- event building, local buffer management on UNIX and VxWorks
- fully distributed run control, data acquisition monitoring, application configuration and startup
- graphical control panel and monitoring displays (strip charts, etc.)
- on-line event filtering
- logging data to disk or tape from UNIX
- ancillary distribution of events to back-end workstations
- on-line analysis framework for back-end workstations
- standard readout software for CAMAC and FASTBUS

DART allows a variety of front-end architectures from a single crate of CAMAC to 30 crates of FASTBUS, with experiment-specific front-end subsystems. In addition, it provides for back-end workstation graphical analysis of sampled experiment data with standard HEP tools such as Histoscope and PAW. This enables DART to be applied over a wide range of architectures:

- a high rate event readout using similar hardware and software, without compromising the cost and simplicity of low rate experiments and the overall throughput and extensibility of high rate experiments
- event building from up to 15 streams of data at rates ranging from hundreds of Kbytes/sec to 200 Mbytes/sec.
- in-line event filtering using similar hardware and software, running either the VxWorks real-time operating system or UNIX.

DART promotes the use of components that can be supported and extended into the future. This includes C++, murmur message reporting system, networked scalable run control, **Tcl** command line interpreter, **Tk** graphical interface for **Tcl**, system and software design and development methodologies.

17.3.2 More about DART

Experimenters can find out more about DART from the On-line Systems Department in the Computing Division. DART collaboration technical information and documents are available on WWW. You'll find *DART Project* on the Computing Division home page.

If your experiment wishes to use DART and receive support for it from the Computing Division, this needs to be negotiated as part of the experiment's MOU agreement with the Laboratory.

Appendix A. VMS Migration for the Impatient

So, you've decided you're ready to convert (or you've run up against a deadline!), but you don't know the first thing about UNIX. Here's enough information to get you moved over. You can use the rest of the manual to learn about UNIX afterwards.

Read this entire appendix before running any commands!!

Keep these typeface conventions in mind as you read the commands:

`typewriter-bold`	In text, used to indicate commands and prompts. In command formats, indicates what the user types "as is".
bold-italic	In command formats, indicates variables for which the user must make context-specific substitutions.

A.1 The Two Necessary Commands

To convert to UNIX, you need to move your mail and your files over from your VMS node (***vmsnode***). To accomplish this while preserving the mail folder and directory structures you had set up on VMS, enter the following two commands exactly as shown from your new UNIX node. (In case you really haven't read any part of this manual yet, the `%` represents the UNIX prompt.)

For your mail, use the command:

`% setup mh ; fvms2mh -vms` ***vmsnode***

Note that ***vmsnode*** must be an individual node, not a cluster alias. For your files, the command is:

`% mkdir vms ; rcp -r` ***vmsnode*****`:'[...]*.*' vms`**

A.2 OK, What's the Catch?

You're right, there's a little more to it. You must have selected a UNIX host to which you will copy your mail and other files. On the FNALU cluster, we suggest that you use the FDEI01 or FSGI02 systems, which each have more than 10 times the capacity of the default FIBI01. You must have learned to log in to UNIX, edit and manage your files, and must have selected a mail reader (presumed to be **mh**, **exmh** or **pine**).

In addition:

- You must have **rcp** access from your UNIX to your VMS account. This takes two steps:

1 Create a `.RHOSTS` file in your VMS `SYS$LOGIN` directory containing one line for each UNIX host that needs automatic access to your VMS account. If your username is different on any machine, include a second line with your username. For example, the contents might look like:

```
fsgi02.fnal.gov
fsgi02 aheavey
fdei01.fnal.gov
fdei01 aheavey
```

2 Make sure your `SYS$LOGIN:LOGIN.COM` does not print anything out. To ensure this, include the following line at the top of the file:

```
IF ( F$MODE() .EQS. "OTHER" ) THEN EXIT !  'F$VERIFY(0)
```

If you can no longer log in to edit your files, send mail to *compdiv@fnal*, and someone can fix it for you.

- For the mail copy, be sure you are using **mailtools v2_2** or later, as installed on the FNALU nodes and most other public nodes at Fermilab. Earlier **mailtools** versions do not support the **`fvms2mh`** command name described here, and may have quota and cleanup problems. To see whether **mailtools v2_2** is current, type **`ups list -a mailtools`**. If it is not current, select it explicitly by typing **`setup mailtools v2_2`**.
- The utility **fvms2mh** brings your mail folders over in **MH** format. If you plan to use **pine** but want to leave your folders in **MH** format, **pine** must be configured to recognize them. To do this, go to **pine**'s SETUP CONFIGURATION menu. There are many options available. Your configuration list should include at least the following:

```
inbox-path          = #mh/inbox
user-domain         = fnal.gov
folder-collections  = mail/[]
                      #mh/[]
[X]  enable-aggregate-command-set
[X]  enable-alternate-editor-cmd
[X]  quell-user-lookup-in-passwd-file
```

- If you choose to use **pine** exclusively, you may want your folders in the **pine** format. First complete the configuration above (with the exception of `inbox-path`). You can move entire folders between the **pine** and **MH** collections as follows:

1 While in **pine**, go to the `FOLDER INDEX` screen for the desired folder.

2 Enter **`;`** (semi-colon) to issue the **pine** Select command.

3 Enter **`A`** to select All messages.

4 Enter **`A`** to Apply the command.

5 Enter **`S`** for Save.

6 Use **<CTRL-N>** or **<CTRL-P>** as necessary to choose the destination folder collection, then enter the new destination folder name. Or just type in a folder collection and name. Respond `Y` to create the new folder.

A.3 Whoa! Too Fast!

If converting in two easy steps doesn't suit your style, you can take somewhat smaller steps.

1) You should probably first forward your current new mail to UNIX, and get used to the **exmh** or **pine** mail readers.

2) Once you've stopped receiving VMS mail for a while, run the first command in section A.1. As a yet more conservative option, check out the "semi-automatic" mail conversion method in Appendix H.

3) Then, when you're happy with the UNIX copy of your mail, delete the original VMS mail files. You may wish to back them up or archive them before deleting.

4) Now to convert your files, run the second command in section A.1 *exactly as shown*. Changes to any of the wildcarding is likely to do nasty things (for instance, moving the oldest rather than most recent version of each file, or dumping everything into a single directory rather than keeping your directory hierarchy intact).

Appendix B. FUE Conformance Levels

The Fermilab Computing Division has defined a set of FUE conformance levels. We list them in order of increasing conformance:

1) **None**: The system cannot be made FUE-compliant because it doesn't meet at least one of the minimum requirements: 1) a supported architecture or OS release, and 2) sufficient local system management support.

2) **Minimal**: The system does meet the minimum requirements listed under **None**, but no FUE software has been installed. At this level, the Computing Division may provide OS support. FUE can be installed at any time.

3) **Core FUE**: The system has enough FUE software installed to allow access to products in **ups** format. This includes **funkern** and **ups**. **funkern** provides the core portion of the Fermi files, as well as programs called from the Fermi files. **ups** provides the setup program, allowing users to gain access to other products.

4) **Full FUE**: This level is regarded as fully FUE compliant. In addition to the components of **Core FUE**, it includes **upd**, **futil**, **systools**, and **shells**. **upd** provides access to other **ups** products. **futil** contains printing utilities, compression programs, the **less** pager program, **Info**, and other utilities. The **systools** product contains the majority of the Fermi files as well as system administration utilities. The **shells** product contains alternative login shells and UNIX scripting languages (**tcsh**, **bash**, **perl**, **gawk**).

5) **FUE plus**: The system has the **Full FUE** installed as well as products related to user communication (e.g., e-mail and Web access). The specific products required are: **www**, **ghostview**, **ximagetools**, and the mail utilities **exmh**, **mh**, **mailtools**, and **tk**. In addition, **FUE plus** requires installation of **X11** and **Motif** releases from the Operating System vendor, since many of the products specified in **FUE plus** are graphical in nature.

We recommend installation of FUE at the Core FUE level or above on all the UNIX systems in use at Fermilab.

These conformance levels are discussed in more detail in the document DR0009 *Computing Division UNIX Environment Standards*.

In addition to the FUE Conformance levels, DR0009 defines the Language Support Environments for FORTRAN, C, and GNU C, which specify the compiled language environments that Computing Division products may require. These are meant to be used in conjunction with the **Full FUE** or **FUE Plus** environments.

Appendix C. awk's Programming Model

This appendix is adapted from a section of the same name in the book *sed & awk*, published by O'Reilly & Associates. It describes the generic structure and organization of an **awk** program.

An **awk** program consists of what is called a *main input loop*. A loop is a routine that is executed over and over again until some condition exists that terminates it. You don't write this loop, it is given; it exists as the framework within which the code that you do write will be executed. The main input loop in **awk** is a routine that reads one line of input from a file and makes it available for processing. The actions you write to do the processing assume that there is a line of input available. In another programming language, you would have to create the main input loop as part of your program.

The main input loop is executed as many times as there are lines of input. This loop does not execute until there is a line of input. It terminates when there is no more input to be read.

awk allows you to write two special routines that can be executed before any input is read and after all input is read. These are the procedures associated with the BEGIN and END rules, respectively. In other words, you can do some pre-processing before the main input loop is ever executed using the BEGIN procedure, and you can do some post-processing with the END procedure after the main input loop has been terminated. The BEGIN and END procedures are optional and they do not need to be defined.

You can think of an **awk** script as having potentially three major parts: what happens before, during, and after processing the input. The "what happens during processing" part is where most of the work gets done. Inside the main loop, your instructions are written as a series of pattern/action procedures. A pattern is a rule for testing the input line to determine whether or not the action should be applied to it. The actions can be quite complex, consisting of statements, functions, and expressions.

Appendix D. VMS to UNIX Command Reference

This appendix is intended as a convenient UNIX command reference for the migrating VMS user. The UNIX equivalents of commonly used VMS commands are listed in tabular format. No information on syntax, options, or arguments is presented here, however some of the listed commands are described elsewhere in *UNIX at Fermilab.*

Commands which are Fermilab-specific or AFS-specific are indicated as such.

D.1 UNIX Equivalents for VMS Commands

Symbols used in table:

These symbols are not part of any UNIX command with which they appear.

* for systems running AFS, the command may not work as documented

** your system may be configured to use an alternate Fermilab command to perform this action

\+ AFS commands

\# local Fermilab commands

Actions	VMS	UNIX
Actions on files and directories		
create directory	CREATE/DIR	**mkdir**
change working directory	SET DEFAULT	**cd**
display working directory	SHOW DEFAULT	**pwd**
display list of files	DIRECTORY DIRECTORY/FULL	**ls** **ls -l**
display list of subdirectories only		**ls -l|grep '^d'** or **find * -type d \ -prune -print**
display contents of file	TYPE	**cat**
display file with pauses	TYPE/PAGE	**more, less**

Actions	VMS	UNIX
display first few lines of file	TYPE/PAGE	**head**
display last few lines of file	TYPE/TAIL	**tail**
copy a single file	COPY	**cp**
copy all files oldfile.* to newfile.*	COPY OLDFILE.* NEWFILE.*	Create shell script; see section D.2.
find file	DIRECTORY	**find**
find file containing a specified expression or string	SEARCH	**grep** **egrep** **fgrep**
compare files	DIFF	**diff** **cmp**
rename file	RENAME	**mv**
rename all files oldfile.* to newfile.*	RENAME OLDFILE.* NEWFILE.*	Create shell script; see section D.2.
delete file or directory	DELETE	**rm** **rmdir**
merge files	COPY or APPEND	**cat**
three-way file merge	MERGE	**merge**
sort or merge files	SORT	**sort**
assign (link) files	ASSIGN DEFINE SET FILE/ENTER	**ln** **setenv**
deassign files (remove link)	DEASSIGN	**rm**
convert and copy file	EXCHANGE	**dd**
update last modified date of file	OPEN/APPEND	**touch**
print file	PRINT	**lpr** **flpr** #
format text file	RUNOFF	**nroff** **troff** **groff** #
octal, decimal, hex, or ascii dump of file	DUMP	**od**
change file protection	SET FILE/PROT	**chmod** *

Actions	VMS	UNIX
change file ownership	SET FILE/OWNER	**chown** * **chgrp** *
list ACL settings on an AFS directory		**fs listacl** +
set ACL settings on an AFS directory		**fs setacl** +
General Commands		
get help	HELP	**man** **apropos**
logout	LOGOUT	**logout** **exit**
change password	SET PASSWORD	**passwd** * **kpasswd** + **yppasswd**[a]
obtain AFS (Kerberos) authentication		**klog** +
discard AFS tokens		**unlog** +
display logged-in users	SHOW USERS	**who** **finger**
display environment settings	SHOW LOGICAL	**alias** **setenv** **printenv** **set**
display date and time	SHOW TIME	**date**
display free disk space	SHOW DEVICE	**df** **du**
display disk quota	SHOW QUOTA	**quota -v**
write to screen	WRITE SYS$OUTPUT	**echo**
stop process	STOP	**kill**
display print queue	SHOW QUEUE	**lpq** **flpq** #
display print entries	SHOW ENTRY	**lpq** **flpq** #
delete print entry	DELETE/ENTRY	**flpk** #
display processes	SHOW SYSTEM	**ps**
tape archiver[b]	BACKUP	**tar**

Actions	VMS	UNIX
change terminal settings	SET TERMINAL	**stty**
talk to another user	PHONE	**talk**
disable messages	SET NOBROADCAST	**mesg n**
connect to remote node	SET HOST	**telnet** **rlogin**
copy file to a remote node	COPY FTP	**rcp** **ftp**
Commands for Executables, Procedures, and Batch Jobs		
compile FORTRAN program	FORTRAN	**f77**
link FORTRAN program	LINK	**f77**
see the entry point names in a library	LIBRARY/LIST/NAMES	**odump -c**[c]
execute program	RUN	***prog_name***
debug program	RUN/DEBUG	**dbx**
create or manipulate library	LIBRARY	**ar**
execute a command procedure (shell script)	*@prog_name*	***prog_name***
submit a batch job	SUBMIT/QUEUE	**qsub** **fbatch_submit** #[d]
check batch queue	SHOW QUEUE	**qstat** **fbatch_q** #
stop batch job	DELETE/ENTRY	**qdel** **kill** **fbatch_cancel** #

a. for use with NIS (see section 2.7)
b. See section D.3 for information on VMS backup save-sets.
c. under IRIX and OSF1
d. **fbatch** is a FUE interface to IBM's **LoadLeveler** job scheduling product.

D.2 Shell Scripts for Copying/Renaming Many Files

In the above table, the UNIX command equivalents for copying/renaming many files require short shell scripts. In this section, we give you the script formulas.

The most efficient way to copy all files `oldfile.*` to `newfile.*` in UNIX is to create a short shell script. Its contents are shown for both the Bourne and the C shell families.

sh, **ksh**, or **bash**:

```
for file in oldfile.*
do
   newname=`echo $file | sed -e 's/oldfile/newfile/'`
   mv $file $newname
done
```

csh or **tcsh**:

```
#!/bin/csh
foreach file (oldfile.*)
   set newname=`echo $file | sed -e 's/oldfile/newfile/'`
   mv $file $newname
end
```

Note that the shell variable *newname* in each case is set to the output of the command that follows the equals sign (**=**). This entire command string must be enclosed in back quotes (`...`). The argument for the **`-e`** option in the **`sed`** command is enclosed in normal single quotes (apostrophes). **sed** is not covered in this manual; most texts discuss it, and the book *sed & awk* published by O'Reilly & Associates covers it thoroughly.

D.3 Unpacking VMS Backup Save-sets

If you made VMS backup save-sets, the UNIX utility **vmsbackup** is a tool that lets you unpack the save-set, converting the files to UNIX format. **vmsbackup** is a separate **ups** product which must be installed first (see Chapter 9). The default operation of the program is to go through an entire VMS-generated tape, extract each file, and write it to disk. Several options are provided; we refer you to the man pages for a discussion of them.

Once **vmsbackup** is installed on your machine, you need to run **`setup vmsbackup`** before using it.

You cannot make save-sets with this utility; you can only unpack *existing* save-sets.

Appendix E. Fermi Login Files

This appendix contains file listings of the FUE-customized default login files (the "Fermi Files") used to set up your UNIX environment.[1] If you are on a FUE-compliant system, you are supplied with a copy of each file in your home directory (except for the `fermi.*` and `setup.*` files which are executed directly from `/usr/local/etc`). The files are reprinted here in their entirety except for the copyright disclaimers.

E.1 C Shell Family

E.1.1 .cshrc

The default FUE `.cshrc` file is found in `/usr/local/etc/stdcshrc`.

```
# .cshrc default settings for all users

# @(#) stdcshrc 1.10 Delta: 93/08/19 14:51:27 Extraction 94/07/06 15:16:15 @(#)

# execute fermi.cshrc first
if ( -r /usr/local/etc/fermi.cshrc ) then
        source /usr/local/etc/fermi.cshrc
endif

# Place any items that you want executed even for non-interactive use here

#skip if not interactive shell
if ( $?USER == 0 || $?prompt == 0 ) exit


set noclobber            #prevent overwrite when redirecting output
set ignoreeof            #prevent accidental logouts

# Have mail point to fermimail which is a Berkely mail version. Some packages
# expect mail to be a System V version of mail. If you are having problems
# of this type, you may need to remove the alias.
alias mail          fermimail       #Fermi recommended Mail

#Define various aliases; user selects desired alias by removng the # sign
#alias a            alias
#alias killit       kill -9         #guarantees that a process is killed
#alias h            'history | tail'
#alias ll           ls -l
#alias la           ls -a           #see hidden files
#alias lf           ls -CF          #check file TYPE (exe, dir ..)
#alias rmi          rm -i           #confirm before deletion
#alias home         cd              #HOME
```

1. The `setpath.*` and `setups.*` files reprinted here are part of the FUE product **funkern v5.0**. All other files are part of FUE's **systools v4.3**.

```
#alias side         'cd ../\!*'     #side
#alias down         'cd \!*'        #down
#alias up           cd ..           #up
#alias cpi          cp -i           #no overwrite of output file
#alias cd          'cd \!*;echo $cwd'
#alias mvi          mv -i           #confirm before moving
#Next alias replaces standard info command on SGI platforms
#alias info         Info            #get list of info articles
#
#VMS type commands
#
#alias dir          ls -l
#alias copy         cp
#alias rename       mv
```

`.cshrc` executes `/usr/local/etc/fermi.cshrc`.

E.1.2 fermi.cshrc

The default FUE `fermi.cshrc` file is found in `/usr/local/etc/fermi.cshrc`.

```
# @(#) fermi.cshrc 4.32 Delta: 94/09/29 22:22:46 Extraction 94/09/29 22:34:39 @(
#)

# fermi.cshrc settings for all users; will be called by the default .cshrc

set MACH_ID=`/usr/local/bin/funame -n`
set MACH_TYPE=`/usr/local/bin/funame -m`
set MACH_OS=`/usr/local/bin/funame -s`

#Determine if this is the first time the fermi.cshrc file has been executed
#on this machine
if ($?FERMICSHRC) then
        if ( $FERMICSHRC == "$MACH_ID" ) then
                set firsttime=0
        else
                set firsttime=1
                setenv FERMICSHRC "$MACH_ID"
        endif
else
        setenv FERMICSHRC "$MACH_ID"
        set firsttime=1
endif

#Execute the things that are only done the first time through
if ( $firsttime == "1" ) then
        # set path
        if ( -r /usr/local/etc/setpath.csh ) then
                source /usr/local/etc/setpath.csh
        endif
        #
        # Establish MANPATH
        #
        setenv MANPATH /usr/products/catman:/usr/products/man:/afs/fnal/products
/catman:/afs/fnal/products/man:/usr/catman:/usr/man:/usr/share/catman:/usr/share
/man:/usr/local/catman:/usr/local/man

endif

#Execute the items done each time (basically only aliases)

#
# Setup UPS
```

```
#
if ( -r /usr/local/etc/setups.csh ) then
        source /usr/local/etc/setups.csh
endif

switch ($MACH_OS)
case IRIX:
        alias fermimail '/usr/sbin/Mail'        # Berkely mail
breaksw
case SunOS:
        alias fermimail '/usr/ucb/Mail'         # Berkely mail
        alias man       'man -F'
breaksw
case AIX:
        alias fermimail '/usr/ucb/Mail'                 #Berkely Mail
breaksw
case ULTRIX:
        alias fermimail '/usr/ucb/Mail'                 # Berkely Mail
breaksw
case OSF1:
        alias fermimail '/usr/bin/Mail'                 # Berkely Mail
breaksw
case HP-UX:
        alias fermimail '/usr/bin/mailx'        #Berkely Mail
breaksw
default
        if ( -x /usr/ucb/Mail ) then
                alias fermimail '/usr/ucb/Mail'         #Berkely Mail
                endif
        if ( -x /usr/sbin/Mail ) then
                alias fermimail '/usr/sbin/Mail'        #Berkely Mail
        endif
        if ( -x /usr/bsd/Mail ) then
                alias fermimail '/usr/bsd/Mail'         #Berkely Mail
        endif
        if ( -x /usr/bin/mailx ) then
                alias fermimail '/usr/bin/mailx'        #Berkely Mail
        endif
        if ( -x /usr/bin/Mail ) then
                alias fermimail '/usr/bin/Mail'        #Berkely Mail
        endif
breaksw
endsw
#
# Check for existance of local.cshsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if ( -r /usr/local/etc/local.cshrc ) then
        source /usr/local/etc/local.cshrc
endif
```

`fermi.cshrc` calls `/usr/local/etc/setpath.csh` to set the shell variable *path,* and it executes `/usr/local/etc/setups.csh` to set up **ups**.

E.1.3 setpath.csh

The default FUE `setpath.csh` file is found in `/usr/local/etc/setpath.csh`.

```
# @(#) setpath.csh 1.7 Delta: 95/03/22 22:13:29 Extraction 95/03/22 22:13:53 @(#
)

if ($?HOME == 0 ) then
        setenv HOME /
```

```
endif

set path=""
foreach DIR ( /usr/sbin \
        /opt/SUNWspro/bin \
        /usr/ccs/bin \
        /usr/lang \
        /usr/bsd \
        /bin \
        /usr/bin \
        /usr/lbin \
        /usr/ucb \
        /etc \
        /usr/etc \
        /usr/afsws/bin \
        /usr/openwin/bin \
        /usr/bin/X11 \
        /usr/kinet/bin \
        /usr/local/bin \
        /usr/bin/mh \
        /usr/sccs )
        if ( -d $DIR ) then
                set path=($path $DIR)
        endif
end
#
# Add your HOME/bin and bin."flavor"
# Put your Sun executables in a bin named bin.SunOS,
#     your IRIX executables in bin.IRIX,
#     your AIX     "         " bin.AIX,
#  ...etc...
#
# Do not do this if you are root, because //bin is your
# $HOME/bin, and do not put current directin in root's path
if ($?MACH_OS == 0) then
        set MACH_OS=`funame -s`
endif
if ( "$HOME" != "/" ) then
        foreach DIR ( $HOME/bin.$MACH_OS $HOME/bin )
                if ( -d $DIR ) then
                        set path=($DIR $path)
                endif
        end
        set path=($path .)
endif
```

E.1.4 setups.csh

The default FUE `setups.csh` file is found in `/usr/local/etc/setups.csh`.

```
#Set environment for UPS
if (! -r ~/.noupsproducts) then #start if .noupsproducts
        set MACH_OS=`funame -s`
        set setprod=no
        if ( $?PRODUCTS == 0 ) then
                set setprod=yes
        else
                 set fchar=`echo $PRODUCTS | cut -c1`
                 if ( "x/" != x"$fchar" ) then
                        set setprod=yes
                 endif
        endif
        if ( $setprod == "yes" ) then #start if $PRODUCTS already set
           if ( $?UPS_EXTRA_DIR == 0 ) then
```

```
            set UPS_EXTRA_DIR=""
         endif
              unsetenv PRODUCTS
              # Add other directories areas that might be product databases
              # to the foreach list
              foreach PROD ( $UPS_EXTRA_DIR \
                      /`funame -n`/products/ups_database/declared \
                      `logdir products`/ups_database/declared \
                      /usr/products/ups_database/declared \
                      /afs/fnal/products/ups_database/declared)
                      if (-d $PROD) then
                              if ($?PRODUCTS) then
                                      set exists=`echo $PRODUCTS | grep -c $PR
OD`
                                      if ( $exists == "0" ) then
                                              setenv PRODUCTS "$PRODUCTS $PROD
"
                                      endif
                               else
                                      setenv PRODUCTS $PROD
                               endif
                      endif
              end
      endif #end if $PRODUCTS already set

      if ($?PRODUCTS) then #set if $PRODUCTS set
              set setups=no
              if ( $?UPS_DIR == 0 ) then
                      set setups=yes
              else
                      set fchar=`echo $UPS_DIR | cut -c1`
                      if ( "x/" != x"$fchar" ) then
                              set setups=yes
                      endif
              endif
              if ( $setups == "yes" ) then #start if $UPS_DIR already set
                      set setups=no
                      set full=${MACH_OS}+`funame -r`
                      foreach PROD ( $PRODUCTS )
                      if ( -r $PROD/ups ) then
                              if ( `grep -c "current.$MACH_OS" $PROD/ups` == 0
 ) then
                                      continue
                              endif
                              foreach i ( `grep "current.$MACH_OS" $PROD/ups|a
wk '{print $2}'|sort -r` )
                                      if ( $full =~ $i* ) then
                                              set flavor=$i
                                              break
                                      endif
                              end
                              if ( $?flavor == 0 ) then
                                      continue
                              endif
#characater after ${flavor} and before " is tab (next 2 lines)
                              set CURRENT=`grep "current.${flavor}    " $PROD/
ups|awk '{print $3}'`
                              setenv UPS_DIR `grep "^Instance:.*${flavor}
" ${PROD}/ups|grep \"${CURRENT}\"|awk '{print $4}'`
                              set fchar=`echo $UPS_DIR | cut -c1`
                              if ( "x/" == x"$fchar" ) then
                                      set setups=yes
                                      break
                              endif
                      endif
```

```
                        end
                        if ($?UPS_DIR) then
                                if ( -r $UPS_DIR/ups/ups_init ) then
                                        source $UPS_DIR/ups/ups_init
                                        setup ups
                                endif
                        endif
                endif #end if $UPS_DIR already set
#Even though UPS_DIR may have already been setup, aliases aren't always
# passed along, thus we need to always do the source of ups_init and
# the setup.csh file. We can't simply do a setup of ups, because then you
# get the current version of ups instead of the version of UPS when you
# entered this process. The setup of ups relies on UPS_PROD_VERSION being
# set (although not in a serious way). This process assumes the version is
# the last component of the $UPS_DIR directory. This isn't guaranteed to be
# correct, but will work ok for this situation.
                if ($?UPS_DIR) then #start if $UPS_DIR is set
                        if ( -r $UPS_DIR/ups/ups_init ) then
                                setenv UPS_PROD_VERSION `basename $UPS_DIR`
                                source $UPS_DIR/ups/ups_init
                                if ( -r $UPS_DIR/ups/setup.csh ) then
                                        source $UPS_DIR/ups/setup.csh
                                endif
                                unsetenv UPS_PROD_VERSION
                        else
                                echo '$UPS_DIR set, but $UPS_DIR/ups/ups_init do
esnt exist, ups environment not setup'
                        endif
                else
                        echo 'Unable to set $UPS_DIR, ups environment not setup'
                endif #end if $UPS_DIR set
        else
                echo 'Unable to set $PRODUCTS, ups environment not setup'
        endif #end if $PRODUCTS set

endif #end if .noupsproducts
```

E.1.5 .login

The default FUE `.login` file is found in `/usr/local/etc/stdlogin`.

```
# .login default settings for all users

#@(#) stdlogin 1.6 Delta: 92/11/05 14:17:45 Extraction 94/07/06 15:16:16 @(#)

#
# .login default settings for all users
#
# execute fermi.login first
#
if ( -r /usr/local/etc/fermi.login ) then
        source /usr/local/etc/fermi.login
endif
#
# defines alias for cedit, comment out if not wanted
#
if ( ! -e ~/.noupsproducts ) then
        setup cedit
endif
#
# Next line sets prompt to <machine_name>
#
set prompt="<$MACH_ID> "
```

```
#
# The standard fermi.profile does not attempt to include /usr/5bin
# (in SunOS) in your PATH.  /usr/5bin contains the UNIX System V version
# of several commands.  By including /usr/5bin first, you will get the
# System V version before the BSD version.  Uncomment one of the following
# if you so desire:
#
#set path=(/usr/5bin $path)
#set path=($path /usr/5bin)
#
# The savehist & history settings maybe expanded to 100
#
set savehist=20      #save last 20 commands for next session
set history=20       #retain the last 20 commands
#
# Uncomment the following to change...
#
#stty kill '^u'          # sets line kill to <ctrl-u>
#stty erase '^H'  # sets erase to backspace
#stty intr '^?'   # sets interrupt to delete
```

The `.login` file executes the file `/usr/local/etc/fermi.login`.

E.1.6 fermi.login

The default FUE `fermi.login` file is found in `/usr/local/etc/fermi.login`.

```
# @(#) fermi.login 4.31 Delta: 94/09/29 22:31:20 Extraction 94/09/29 22:34:39 @(
#)

#
# fermi.login settings for all users; called by the default .login
# Set the umask so that newly created files and directories will be readable
# by others, but writable only by the user.
#
umask 022
#
# Following put in to handle NQS
#
if ( $?ENVIRONMENT ) then
        if ( $ENVIRONMENT == "BATCH" ) exit
endif
#
# Determine terminal type
#
set ttype=`echo $term |cut -c1`
switch (x$term)
case x:
case xunknown:
case xarpanet:
case xnetwork:
case xnet:
case xdialup:
case xdumb:
        set term=vt100
        stty erase '^?'
breaksw
default
        if ( -r /usr/lib/terminfo/$ttype/$term || -r /usr/share/lib/terminfo/$tt
ype/$term ) then
                switch ($term)
                case iris-ansi:
                case iris-ansi-net:
```

```
                case hp:
                        stty erase '^h'
                breaksw
                default
                        stty erase '^?'
                breaksw
                endsw
        else
                set bterm=`echo $term | cut -c1-3`
                if ( -r /usr/lib/terminfo/$ttype/$bterm || -r /usr/share/lib/ter
minfo/$ttype/$bterm ) then
                        set term=$bterm
                        stty erase '^?'
                else
                        switch ($MACH_OS)
                        case IRIX:
                        case SunOS:
                        case AIX:
                                switch ($bterm)
                                case vt2:
                                case vt3:
                                        set term=vt220
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        case ULTRIX:
                                switch ($bterm)
                                case vt3:
                                        set term=vt300
                                        stty erase '^?'
                                breaksw
                                case vt2:
                                        set term=vt200
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        case OSF1:
                                switch ($bterm)
                                case vt2:
                                case vt3:
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        default
                                set term=vt100
                                stty erase '^?'
                        breaksw
                        endsw
                endif
        endif
```

```
breaksw
endsw
#
# Set DISPLAY
#
if ( ! $?DISPLAY ) then
        set TTYPORT=`tty`
        if ( $TTYPORT == /dev/console ) then
                setenv DISPLAY "localhost:0"
        else if ( $?REMOTEHOST ) then
                setenv DISPLAY "${REMOTEHOST}:0"
        else
                set TTYNAME=`echo $TTYPORT |cut -c6-`
                set REMOTEHOST=`who|grep "$TTYNAME"|awk '{print $6}'|sed 's/(//'
|sed 's/)//'`
                setenv DISPLAY "${REMOTEHOST}:0"
        endif
endif
#
# Who are you?
#
if ( $?LOGNAME ) then
        setenv LOGNAME "$LOGNAME"
else if ( $?USER ) then
        setenv LOGNAME "$USER"
else if ( $?REMOTEUSER ) then
        setenv LOGNAME "$REMOTEUSER"
endif
#
# The switch statement is to set the mail variable
# based on machine type.  Also motd and news are
# dependent on system type. Some systems automatically
# display motd (and thus shouldn't be repeated here.
# Others don't have news.
#
switch ($MACH_OS)
case IRIX:
        setenv TZ CST6CDT
        if ( `funame -r|cut -c1` == 3 ) then
                set mail=/usr/mail/$LOGNAME
                if ( { /bin/mail -e } ) then
                        echo 'You have mail.'
                endif
        endif
        if ( { test -d /usr/news } ) then
                /usr/bin/news
        endif
breaksw
case SunOS:
        setenv TZ CST6CDT
        set mail=/usr/spool/mail/$LOGNAME
breaksw
case AIX:
        setenv TZ CST6CDT
        /bin/news
        setenv MAIL "/usr/spool/mail/$LOGNAME"
        setenv MAILMSG "[YOU HAVE NEW MAIL]"
        if ( { test -s $MAIL } ) then
                echo 'You have mail.'
        endif
        setenv EMULATE none
breaksw
case ULTRIX:
        set mail=/usr/spool/mail/$USER
breaksw
```

```
case OSF1:
        /usr/bin/news
        set mail=/usr/spool/mail/$USER
breaksw
case HP-UX:
        setenv TZ CST6CDT
        cat -s /etc/motd
        /usr/bin/news
        set mail=/usr/mail/$LOGNAME
        if ( { /bin/mail -e } ) then
                echo 'You have mail.'
        endif
breaksw
case "":
        #If for some reason funame -s returns NULL, as opposed to returning
        #something that isn't recognized, do nothing
breaksw
default
        cat -s /etc/motd
        if ( -x /usr/bin/news) then
                /usr/bin/news
        endif
        if ( -x /bin/news) then
                /bin/news
        endif
        if ( -d /usr/spool/mail ) then
                set mail=/usr/spool/mail
        endif
        if (    -d /usr/mail ) then
                set mail=/usr/mail/$LOGNAME
        endif
        if ( -x /bin/mail ) then
                if ( { /bin/mail -e } ) then
                        echo 'You have mail.'
                endif
        endif
breaksw
endsw
#
# Establish PAGER
#
if ( -r /usr/local/bin/less ) then
        setenv PAGER /usr/local/bin/less
else
        setenv PAGER more
endif
#
# Common terminal characteristics
#
stty intr '^c'  # set interrupt key to <ctrl-c>
stty kill '^x'  # set kill key to <ctrl-x>
stty echoe                      # erase ERASEd characters
echo "Terminal Type is $TERM"
/usr/local/bin/Info -new
#
# Some systems the user doesn't own his tty device ( Sun OpenWindows) so
# redirect stderr
#
/bin/chmod 622 `tty` >& /dev/null
#
# Check for existance of local.cshsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if ( -r /usr/local/etc/local.login ) then
```

```
        source /usr/local/etc/local.login
endif
```

E.2 Bourne Shell Family

E.2.1 .profile

The default FUE `.profile` file is found in `/usr/local/etc/stdprofile`.

```
# @(#) stdprofile 1.6 Delta: 92/11/05 14:19:26 Extraction 94/07/06 15:16:17 @(#)

#
# .profile default settings for all users
#
# Execute fermi.profile first!
#
if [ -r /usr/local/etc/fermi.profile ]
then
  . /usr/local/etc/fermi.profile
fi
#
# PS1 sets prompt to <machine_name>
# PS2 sets secondary prompt to <more>
#
export PS1;             PS1="<${MACH_ID}> "
export PS2;             PS2="<more> "
#
# HISTSIZE is the number of commands retained in history
#
export HISTSIZE;        HISTSIZE=20       #retain the last 20 commands
#
# EDITOR is used as your default editor, emacs or vi typically
# VISUAL is used as your command line editor, emacs or vi typically
# To use the emacs version, comment out or remove the vi references
# and uncomment the emacs verisons...
#
#export EDITOR;         EDITOR=emacs
#export VISUAL;         VISUAL=emacs
#
export EDITOR;          EDITOR=vi
export VISUAL;          VISUAL=vi
#
# The standard fermi.profile does not attempt to include /usr/5bin
# (in SunOS) in your PATH.  /usr/5bin contains the UNIX System V version
# of several commands.  By including /usr/5bin first, you will get the
# System V version before the BSD version.  Uncomment one of the following
# if you so desire:
#
# PATH=/usr/5bin:$PATH
# PATH=$PATH:/usr/5bin
#
# ENV is the location of your shell alias file
#
export ENV;             ENV=~/.shrc
#
# Uncomment the following to change...
#
#stty kill ^U   # sets line kill to <ctrl>U
#stty erase  # sets erase to backspace
#stty intr ^?   # sets interrupt to delete
#
```

```
if [ "`basename $SHELL|sed 's/[0-9]//'`" = "sh" ] && [ -r $HOME/.shrc ]
then
  . $HOME/.shrc
fi
```

This file calls `/usr/local/bin/fermi.profile`.

E.2.2 fermi.profile

The default FUE `fermi.profile` file is found in `/usr/local/etc/fermi.profile`.

```
# @(#) fermi.profile 1.33 Delta: 94/10/31 22:14:25 Extraction 94/11/02 16:56:08
@(#)

# fermi.profile settings for all users; called by the user bourne/korn .profile
# Set the umask so that newly created files and directories will be readable
# by others, but writable only by the user.
umask 022
MACH_ID="`/usr/local/bin/funame -n`"
MACH_TYPE="`/usr/local/bin/funame -m`"
MACH_OS="`/usr/local/bin/funame -s`"
#
# Set the PATH appropriately
#
if [ -r /usr/local/etc/setpath.sh ]
then
  . /usr/local/etc/setpath.sh
fi

#
# Set a respectable MANPATH
#
MANPATH=/usr/products/catman:/usr/products/man:/afs/fnal/products/catman:/afs/fn
al/products/man:/usr/catman:/usr/man:/usr/share/catman:/usr/share/man:/usr/local
/catman:/usr/local/man
#
# Following put in to handle NQS
# Only do rest if non-interactive session
#
if [ "$ENVIRONMENT" != "BATCH" ] && [ "x`logname`" != "x" ]
then
#
# Determine the terminal type
#
TTYPE=`echo $TERM | cut -c1`
case $TERM in
""|arpanet|net|network|dialup|unknown|dumb)
        TERM=vt100
        stty erase '^?'
;;
*)
        if [ -r /usr/lib/terminfo/$TTYPE/$TERM -o -r /usr/share/lib/terminfo/$TT
YPE/$TERM ]
        then
                case $TERM in
                iris-ansi*|hp)
                        stty erase '^h'
                ;;
                *)
                        stty erase '^?'
                ;;
                esac
        else
```

```
                BTERM=`echo $TERM | cut -c1-3`
                if [ -r /usr/lib/terminfo/$TTYPE/$BTERM -o -r /usr/share/lib/ter
minfo/$TTYPE/$BTERM ]
                then
                        TERM=$BTERM
                        stty erase '^?'
                else
                        case $MACH_OS in
                        IRIX|SunOS|AIX)
                                case $BTERM in
                                vt[2|3])
                                        TERM=vt220
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        ULTRIX)
                                case $BTERM in
                                vt3)
                                        TERM=vt300
                                        stty erase '^?'
                                ;;
                                vt2)
                                        TERM=vt200
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        OSF1)
                                case $BTERM in
                                vt[2|3])
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        HP-UX)
                                case $BTERM in
                                vt3)
                                        TERM=vt320
                                        stty erase '^?'
                                ;;
                                vt2)
                                        TERM=vt200
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac

                        ;;
                        *)
```

```
                                TERM=vt100
                                stty erase '^?'
                        ;;
                        esac
                fi
        fi
;;
esac
#
# Set the DISPLAY
#
if [ x"$DISPLAY" = x ]
then
        TTYPORT=`tty`
        if [ "$TTYPORT" = "/dev/console" ]
        then
                DISPLAY="localhost:0"
        elif [ "$REMOTEHOST" = "" ]
        then
                TTYNAME=`echo $TTYPORT | cut -c6-`
                REMOTEHOST=`who|grep "$TTYNAME"|awk '{print $6}'|sed 's/(//'|sed
 's/)//'`
                DISPLAY="${REMOTEHOST}:0"
        else
                DISPLAY="${REMOTEHOST}:0"
        fi
fi
#
# Who are you?
#
if [ "x$LOGNAME" = "x" ]
then
        if [ "x$USER" = "x" ]
        then
        if [ "x$REMOTEUSER" != "x" ]
        then
                LOGNAME="$REMOTEUSER"
        fi
        else
                LOGNAME="$USER"
        fi
fi
#
# The case statement is to set the mail variable
# based on machine type.  Also motd and news are
# dependent on system type. Some systems automatically
# display motd (and thus shouldn't be repeated here.
# Others don't have news.
#
case $MACH_OS in
IRIX)
        TZ=CST6CDT
        if [ "`funame -r | cut -c1`" = "3" ]
        then
                MAIL=/usr/mail/$LOGNAME
                /bin/mail -e
                if [ $? = 0 ]
                then
                        echo 'You have mail.'
                fi
        fi
        if [ -d /usr/news ]
        then
                /usr/bin/news
        fi
```

```
;;
SunOS)
        TZ=CST6CDT
        MAIL=/usr/spool/mail/$LOGNAME
;;
AIX)
        TZ=CST6CDT
        /bin/news
        MAIL="/usr/spool/mail/$LOGNAME"
        export MAILMESG;MAILMESG="[YOU HAVE NEW MAIL]"
        if [ -s $MAIL ]
        then
                echo 'You have mail.'
        fi
        EMULATE=none
        export EMULATE
;;
ULTRIX)
        MAIL=/usr/spool/mail/$USER
;;
OSF1)
        /usr/bin/news
        MAIL=/usr/spool/mail/$USER
;;
HP_UX)
        TZ=CST6CDT
        cat -s /etc/motd
        MAIL=/usr/mail/$LOGNAME
        /bin/mail -e
        if [ $? = 0 ]
        then
                echo 'You have mail.'
        fi
        /usr/bin/news
;;
"")
        # Catch for the "funame returns null" bug
        :
;;
*)
        cat -s /etc/motd
        if [ -x /usr/bin/news ]
        then
                /usr/bin/news
        fi
        if [ -x /bin/news ]
        then
                /bin/news
        fi
        if [ -d /usr/spool/mail ]
        then
                MAIL=/usr/spool/mail
        fi
        if [    -d /usr/mail    ]
        then
                MAIL=/usr/mail/$LOGNAME
        fi
        if [ -x /bin/mail ]
        then
                /bin/mail -e
                if [ $? = 0 ]
                then
                        echo 'You have mail.'
                fi
        fi
```

```
;;
esac
#
#  Establish PAGER
#
if [ -x /usr/local/bin/less ]
then
        PAGER=/usr/local/bin/less
else
        PAGER=more
fi
#
# Common terminal characteristics
#
stty intr '^c'  # set interrupt key to <ctrl-c>
stty kill '^x'  # set kill key to <ctrl-x>
stty echoe                      # erase ERASEd characters
echo "Terminal Type is $TERM"
if [ -x /usr/local/bin/Info ]
then
        /usr/local/bin/Info -new
fi
# Some systems the user doesn't own his tty device ( Sun OpenWindows) so
# redirect stderr
/bin/chmod 622 `tty` >/dev/null 2>&1
export DISPLAY MAIL TZ TERM PATH PAGER MANPATH

fi      # This is end of if statement for interactive use.
#
# Check for existance of local.profile file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if [ -r /usr/local/etc/local.profile ]
then
  . /usr/local/etc/local.profile
fi
```

`fermi.profile` executes `/usr/local/etc/setpath.sh` to set the *path* variable.

E.2.3 setpath.sh

The default FUE `setpath.sh` file is found in `/usr/local/etc/setpath.sh`.

```
# @(#) setpath.sh 1.9 Delta: 95/01/18 15:29:38 Extraction 95/03/22 22:13:53 @(#)

if [ "x"$HOME = "x" ]
then
        HOME=/
fi

PATH=""
for DIR in /usr/sbin \
                                        /opt/SUNWspro/bin \
                                        /usr/ccs/bin \
                                        /usr/lang \
                                        /usr/bsd \
                                        /bin \
                                        /usr/bin \
                                        /usr/lbin \
                                        /usr/ucb \
                                        /etc \
                                        /usr/etc \
                                        /usr/afsws/bin \
```

```
                                        /usr/openwin/bin \
                                        /usr/bin/X11 \
                                        /usr/kinet/bin \
                                        /usr/local/bin \
                                        /usr/bin/mh \
                                        /usr/sccs
do
        if [ -d ${DIR} ]
        then
                PATH=${PATH}${DIR}:
        fi
done
#
# zsh bug fix
#
PATH=`echo $PATH|/bin/sed 's/\.//g'`
if [ "x"$MACH_OS = "x" ]
then
        MACH_OS=`funame -s`
fi
# do not do this if you are root, because //bin is your $HOME/bin, and
# do not put current directory in root's path
if [ "$HOME" != "/" ]
then
        #
        # Add your HOME/bin and bin."flavor"
        # Put your Sun executables in a bin named bin.SunOS,
        #     your IRIX executables in bin.IRIX,
        #     your AIX     "         " bin.AIX,
        #  ...etc...
        #
        for DIR in ${HOME}/bin.${MACH_OS} $HOME/bin
        do
                if [ -d ${DIR} ]
                then
                        PATH=${DIR}:${PATH}
                fi
        done
        PATH=${PATH}.:
fi

export PATH
```

E.2.4 .shrc

The default FUE `.shrc` file is found in `/usr/local/etc/stdshrc`.

```
# .shrc default settings for all users

#execute fermi.shrc first
if [ -r /usr/local/etc/fermi.shrc ]
then
        . /usr/local/etc/fermi.shrc
fi

#Place items that you want run even when not interactive shell here

#skip if not interactive shell
if [ "x$USER" = "x" ] || [ "x$PS1" = "x" ]
then
        :
else
```

```
        if [ "`basename $SHELL|sed 's/[0-9]//'`" != "sh" ]
        then
          set -o noclobber            #prevent overwrite when redirecting output
          set -o ignoreeof            #prevent accidental logouts

  # Have mail point to fermimail which is a Berkely mail version. Some packages
  # expect mail to be a System V version of mail. If you are having problems
  # of this type, you may need to remove the alias.
        alias mail='fermimail'

  #Define various aliases; user selects desired alias by removng the # sign
  # ONLY IF YOUR SHELL SUPPORTS ALIASING!
          #alias        a='alias'
          #alias        logout='exit'
          #alias        killit='kill -9'        # guarantees that a process is k
illed
          #alias        ll='ls -l'              # generate a long listing
          #alias        la='ls -a'              # see hidden files
          #alias        lf='ls -CF'             # check file TYPE (exe, dir ..)
          #alias        rmi='rm -i'             # confirm before deletion
          #alias        cpi='cp -i'             # no overwrite of output file
          #alias        mvi='mv -i'             # confirm before moving
          #alias        more='less'             # when you explicitly "more" a f
ile

  #Next alias replaces standard info command on SGI platforms
          #alias        info='Info'            #get list of info articles

        #VMS type commands
          #alias        dir='ls -l'
          #alias        copy='cp'
          #alias        rename='mv'

        else
  # Have mail point to fermimail which is a Berkely mail version. Some packages
  # expect mail to be a System V version of mail. If you are having problems
  # of this type, you may need to remove the alias.
          mail ()
          {
            fermimail "$@"
          }
  #Define various functions; user selects desired functions by removng the # sig
n
  #Provided for shells that do not support aliasing.
          #killit ()
          #{
            #  kill -9  # guarantees that a process is killed
          #}
          #ll ()
          #{
            #  ls -l    "$@"     # generate a long listing
          #}
          #la ()
          #{
            #  ls -a    "$@"     # see hidden files
          #}
          #lf ()
          #{
            #  ls -CF "$@"      # display file type, columnwise
          #}
          #rmi ()
          #{
            #  rm -i "$@"       # confirm before delete
          #}
```

```
          #cpi ()
          #{
            #  cp -i "$@"       # no overwrite of output file (see also set nocl
obber)
          #}
          #mvi ()
          #{
            #  mv -i "$@"       # confirm before moving
          #}
          #more ()
          #{
            #  less "$@"    # when you explicitly "more" a file
          #}

  #Next function replaces standard info command on SGI platforms
          #info ()
          #{
            #  Info "$@"        # get list of info articles
          #}

  #VMS type commands
          #dir ()
          #{
            #  ls -l "$@"
          #}
          #copy ()
          #{
            #  cp "$@"
          #}
          #rename ()
          #{
            #  mv "$@"
          #}
        fi
fi
```

The `.shrc` file calls `/usr/local/etc/fermi.shrc`.

E.2.5 fermi.shrc

The default FUE `fermi.shrc` file is found in `/usr/local/etc/fermi.shrc`.

```
# @(#) fermi.shrc 1.10 Delta: 94/09/29 22:22:52 Extraction 94/09/29 22:34:43 @(#
)

# fermi.shrc settings for all users; will be called by the default .profile

#
# Setup UPS
#
if [ -r /usr/local/etc/setups.sh ]
then
        . /usr/local/etc/setups.sh
fi

#Do machine dependent functions:
case $MACH_OS in
IRIX)
  fermimail ()
  {
    /usr/sbin/Mail "$@"
  }
;;
SunOS)
```

```
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
  man ()
  {
        /usr/bin/man -F "$@"
  }
;;
AIX)
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
;;
ULTRIX)
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
;;
OSF1)
  fermimail ()
  {
    /usr/bin/Mail "$@"
  }
;;
HP-UX)
  fermimail ()
  {
    /usr/bin/mailx "$@"
  }
;;
*)
  if [ -x /usr/ucb/Mail ]
  then
    fermimail ()
    {
      /usr/ucb/Mail "$@"
    }
  elif [ -x /usr/sbin/Mail ]
  then
    fermimail ()
    {
      /usr/sbin/Mail "$@"
    }
  elif [ -x /usr/bsd/Mail ]
  then
    fermimail ()
    {
      /usr/bsd/Mail "$@"
    }
  elif [ -x /usr/bin/mailx ]
  then
    fermimail ()
    {
      /usr/bin/mailx "$@"
    }
  elif [ -x /usr/bin/Mail ]
  then
    fermimail ()
    {
      /usr/bin/Mail "$@"
    }
  fi
```

```
;;
esac
#
# Check for existance of local.shsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if [ -r /usr/local/etc/local.shrc ]
then
  . /usr/local/etc/local.shrc
fi
```

`fermi.shrc` executes `/usr/local/etc/setups.sh`.

E.2.6 setups.sh

The default FUE `setups.sh` file is found in `/usr/local/etc/setups.sh`.

```
# @(#) setups.sh 2.2 Delta: 95/01/09 15:48:34 Extraction 95/03/22 22:13:54 @(#)

#Set environment for UPS
if [ ! -r $HOME/.noupsproducts ] #start if .noupsproducts
then
        MACH_OS=`funame -s`
        fchar=`echo ${PRODUCTS:-}|cut -c1`
        if [ "x$fchar" != "x/" ] #start check if $PRODUCTS already set
        then
                prodset=no
                unset PRODUCTS
                # Add any other directories that are UPS databases in the
                # for list
                for PROD in $UPS_EXTRA_DIR \
                    /`funame -n`/products/ups_database/declared \
                         `logdir products`/ups_database/declared \
                         /usr/products/ups_database/declared \
                         /afs/fnal/products/ups_database/declared
                do
                        if [ -d $PROD ]
                        then
                                if [ "$prodset" = "no" ]
                                then
                                        PRODUCTS=$PROD
                                else
                                        exists=`echo $PRODUCTS | grep -c $PROD`
                                        if [ $exists = "0" ]
                                        then
                                                PRODUCTS="$PRODUCTS $PROD"
                                        fi
                                fi
                                prodset=yes
                        fi
                done
                export PRODUCTS
        else
                prodset=yes
        fi #end check if $PRODUCTS already set
        if [ "$prodset" = "yes" ] #start if $PRODUCTS is set
        then
                fchar=`echo ${UPS_DIR:-}|cut -c1`
                if [ "x$fchar" != "x/" ] #start if $UPS_DIR already set
                then
                        setups=no
                        full=${MACH_OS}+`funame -r`
```

```
                        for PROD in $PRODUCTS
                        do
                        if [ -f $PROD/ups ]
                        then
                                if [ `grep -c "current.$MACH_OS" $PROD/ups` -eq
0 ]
                                then
                                        continue
                                fi
                                flavor=x
                                for i in `grep "current.$MACH_OS" $PROD/ups|awk
'{print $2}'|sort -r`
                                do
                                        case "$full" in
                                        $i* )   flavor=$i
                                                break;;
                                        *) ;;
                                        esac
                                done
                                if [ $flavor = x ]
                                then
                                        continue
                                fi
#character after ${flavor} and before quote is tab (next two lines)
                                CURRENT=`grep "current.${flavor}        " $PROD/
ups|awk '{print $3}'`
                                UPS_DIR=`grep "^Instance:.*${flavor}    " ${PROD
}/ups |grep \"${CURRENT}\"|awk '{print $4}'`
                        fi
                        fchar=`echo ${UPS_DIR:-}|cut -c1`
                        if [ "x$fchar" = "x/" ]
                        then
                                setups=yes
                                break
                        fi
                        done
                        if [ "$setups" = "yes" ]
                        then
                                export UPS_DIR
                                if [ -f $UPS_DIR/ups/ups_init.sh ]
                                then
                                        . $UPS_DIR/ups/ups_init.sh
                                        setup ups
                                fi
                        fi
                else
                        setups=yes
                fi #end if $UPS_DIR already set
#Even though UPS_DIR may have already been setup, aliases aren't always
# passed along, thus we need to always do the source of ups_init and
# the setup.sh file. We can't simply do a setup of ups, because then you
# get the current version of ups instead of the version of UPS when you
# entered this process. The setup of ups relies on UPS_PROD_VERSION being
# set (although not in a serious way). This process assumes the version is
# the last component of the $UPS_DIR directory. This isn't guaranteed to be
# correct, but will work ok for this situation.
                if [ "$setups" = "yes" ]
                then
                        if [ -f $UPS_DIR/ups/ups_init.sh ]
                        then
                                . $UPS_DIR/ups/ups_init.sh
                                UPS_PROD_VERSION=`basename $UPS_DIR`
                                if [ -f $UPS_DIR/ups/setup.sh ]
                                then
                                        . $UPS_DIR/ups/setup.sh
```

```
                                fi
                                unset UPS_PROD_VERSION
                        else
                                echo "\$UPS_DIR set, but \$UPS_DIR/ups/ups_init.
sh doesn't exist. No ups environment setup."
                        fi
                else
                        echo "Unable to set \$UPS_DIR, ups environment not setup
"
                fi
        else
                        echo "Unable to set \$PRODUCTS, ups environment not setup"
        fi #end if $PRODUCTS is set
fi #end if .noupsproducts
```

Appendix F. mh and exmh Customization

This appendix contains information for further customizing the **mh** and/or **exmh** mail readers. Given the multitude of customizable features in these mail readers, we cannot provide you with a comprehensive treatment of the subject here. The information presented in Chapter 11 and this appendix should be sufficient for most users.

We refer you to the on-line help and to the O'Reilly book, *MH & xmh - Email for Users & Programmers* by Jerry Peek, for further information.

F.1 Forwarding and Notification

The `.forward` file is a general mail forwarding file, and is used no matter which mail package you choose to use. It tells the system what to do with your mail; whether to leave it in the system spool area, forward it to the spool area on a different node, forward it to a different system entirely, or (in some cases) deliver it to your preferred local mailbox automatically. It may also be used as a means of notifying you that mail has arrived.

Note that comment lines are not permitted in the `.forward` file.

F.1.1 Forwarding Address

The first line of your `.forward` file tells the system where to put your mail. On your chosen mail node, it should look like:

***username*@*node*.fnal.gov**

where ***username*** is your username, and ***node*** is the name of the node where you plan on reading mail. The backslash prevents infinite loops in a "clustered" environment where more than one node sees the same `.forward` file in your login area.[1]

On all other systems, we recommend that you forward your mail to FNAL, the lab's central mail server. For these systems, the first line of each `.forward` file will look like:

***username*@fnal.gov**

Note that this doesn't have the backslash or the node name. The backslash is not necessary if you are forwarding to a node that does not share the same login area. See sections 2.6 and 2.7 for a discussion of shared login areas, or what we call UNIX *clusters*.

---

1. If you choose to use *unattended autoincorporation*, which delivers your mail directly into your login area rather than leaving it in the system spool area, the first line of your `.forward` file will look significantly different from this example. Unattended autoincorporation is covered in section F.3.

F.1.2 Mail Notification

Another use of the `.forward` file is to allow you to customize the type of notification you receive when new mail arrives. Usually, the second line of your `.forward` file will control if and how you receive notification.

FUE provides minimal notification of incoming mail by default, however most people like to receive notification of individual messages as they come in.

The use of the **MH** program **rcvtty** provides notification of new mail whether or not you have a mail reader active. To set this up, you need to edit your `.forward` file.

Basic Configuration

Even if you don't know what **biff/comsat** is, you will need to determine if it is running on your system. How do you tell? Follow these instructions and you should get one concise notification message per incoming mail message if it is not running. If you get two notifications, one of which is quite verbose, then **biff/comsat** is running, and you'll need to change your configuration as described below.

If you're on a cluster where some nodes run **biff/comsat** and some don't, set up notification according to your chosen mail node. This procedure assumes that **MH** is installed and configured on all nodes within the cluster.

First, you need to make sure you have set up the proper file access permission. Activate tty user execute permission by using the command:

```
% chmod u+x `tty`
```

where the quotes around **`tty`** are both single back quotes.

You can include this in your `.login` or `.profile` file to run it automatically on all sessions. Be sure to include it *after* the call to `fermi.login` or `fermi.profile` since these files set it differently. Or, if you prefer to have notification only in one window, just run this command in that window.

The second line of your `.forward` file should look like:

```
"| /usr/local/products/mh/current/lib/rcvtty -biff"
```

When mail arrives, **rcvtty** should send a notice containing the time, sender, subject, and message sample to all user tty sessions with proper file access permission in the default format:

```
time:  SENDERNAME   subject << first line of text (truncated)
```

If this type of notification is what you get, skip to "Customizing rcvtty Output", below. If instead you get a multiline notice, **biff/comsat** is running, and you'll have to make a couple of changes.

Some systems run **biff** with **comsat** for notification purposes. **biff/comsat** and **rcvtty** (without the `biff` option) can work together, however this yields double notification (both **comsat**'s verbose notice and **rcvtty**'s single line notice) because the `r-w-x` permissions (see section 6.6.1) need to be different for **biff/comsat** and **rcvtty**.

To get **rcvtty** to work properly on systems running **biff/comsat** follow these steps:

1) Enter **`biff n`** to deactivate **comsat** (best to include in `.login` or `.profile` file).

2) In `.forward`, replace:

```
"| /usr/local/products/mh/current/lib/rcvtty -biff"
```

with:

```
"| /usr/local/products/mh/current/lib/rcvtty"
```

3) Enter **`ls -lL `tty``** to see your default tty setting. If `o+w` (that is, if *write* permission

for *other*; see section 6.6.1) is not set, use the command **`chmod o+w `tty``** to set it (again, best to include in `.login` or `.profile` file).

Customizing rcvtty Output

You can reformat the output of **rcvtty** if you like. To do this you need to create a form file describing the syntax of the output you want to see, and you need to modify the **`rcvtty`** command. For example, if you wish to use the same format as `scan-form` (see section F.2.5) the form file used for incorporating or scanning mail, the line in `.forward` would look like this (shown in the format used when **biff/comsat** is *not* running):

```
"| /usr/local/products/mh/current/lib/rcvtty -biff -form scan-form"
```

F.2 Customizing mh and exmh

Your personal configuration for **MH** is primarily defined via the `.mh_profile` file. It defines, among other things, where mail is stored and what editor to use. Some additional files are available if you wish to customize further.

Each **MH** mail composition command, (**`comp`**, **`repl`**, **`forw`** for **mh**; **SEND**, **REPLY**, **FORWARD** for **exmh**) uses its own template file to define the message header format:

- **`comp`** or **SEND** uses the `components` file
- **`repl`** or **REPLY** uses `replcomps`
- **`forw`** or **FORWARD** uses `forwcomps`

If you don't have these template files in your `$HOME/Mail` directory the system default files in the **MH** library are used.

Options for incorporating mail were discussed in Chapter 11. The `scan-form` and `inc-form` files provide the format of the display when you scan and incorporate mail. You can control how local delivery is performed by creating a `.maildelivery` file.

F.2.1 .mh_profile

It also allows you to specify options to **MH** commands. If you have run **`setup mh`** or **`setup exmh`**, you already have a simple one in your home directory. You can tailor your `.mh_profile` file to suit your taste. Following are some of the extra lines you can add:

`Path: Mail`	Directory for your **MH** transactions.
`Unseen-Sequence: unseen`	Name of the unseen sequence of messages.
`Draft-folder: drafts`	Name of the default folder for drafted, unsent mail.
`Editor: emacs`	Sets up emacs as editor for **`comp`**, **`repl`**, **`forw`**, **`dist`** (use any editor you want).[1]
`rmmproc: /bin/rm`	Causes mail to be truly deleted when requested (see section 11.3.8). **Recommended!**
`Aliasfile: mh_alias`	File containing **MH** aliases (distribution lists).
`Scan: -form scan-form`	Provides format of display when you scan mail.
`inc: -form inc-form`	Provides format of display when you incorporate mail.
`Signature: John Peoples`	Your signature. Appears in From: header field.

1. **exmh** ignores any editor defined here (defining an editor is optional for **MH**)

The `Path` is taken to be relative to your `$HOME` directory. The files/directories used for `Draft-folder`, `Aliasfile`, `Scan`, and `inc` are assumed to be relative to the directory specified by `Path`.

F.2.2 components

The default template file for composing (sending) a message looks like:

```
To:
cc:
Subject:
-------
```

You can add other fields to the header by making your own components file. For example, an enhanced components file may include the additional fields shown below:

```
Reply-to: username@fnal.gov
To:
cc:
Bcc:
Fcc: yourfolder
Subject:
-------
```

The `Reply-to:` field should give your FNAL address. It is not needed if you want the reply to come to the node from which you are sending the message. The **`repl`** command in **MH** (and some other mail agents as well) will route replies to this address rather than the `From:` address.

The "blind carbon copy" `Bcc:` field can be used to send copies of the message to addresses that are not seen by the people specified in the `To:` and `cc:` fields. This is like forwarding a copy of the message to someone else after sending it, except that here you can do it at the time you send the original message. `Bcc:` can also use distribution lists or aliases.

You can send a copy to yourself by specifying a folder for your copies in the `Fcc:` field. *Fcc* stands for folder copy. This copy goes directly to the specified folder, and only to that folder.

F.2.3 replcomps

The `replcomps` file controls the header in replies. The default file is given below:

```
%(lit)%(formataddr %<{reply-to}%?{from}%?{sender}%?{return-path}%>)\
%<(nonnull)%(void(width))%(putaddr To: )\n%>\
%(lit)%(formataddr{to})%(formataddr{cc})%(formataddr(me))\
%<(nonnull)%(void(width))%(putaddr cc: )\n%>\
%<{fcc}Fcc: %{fcc}\n%>\
%<{subject}Subject: Re: %{subject}\n%>\
%<{date}In-reply-to: Your message of "\
%<(nodate{date})%{date}%|%(pretty{date})%>."%<{message-id}
                %{message-id}%>\n%>\
--------
```

The first two lines in the file build the `To:` address field for your reply draft. It takes the first available address by searching the `Reply to:`, `From:`, `Sender:` and `Return-path:` fields, in that order. Lines 3 and 4 build a `cc:` address field by taking all the `To:` and `cc:` addresses from the original message and including your address as well. It then builds an `Fcc:`

field by testing a **-fcc** switch on the command line or in the .mh_profile. The subject: field is taken from the subject in the incoming mail preceded by Re:. The last three lines make up the field that says "*In-reply-to: Your message of ...*" with the date and message number at the end. The default reply header takes the form:

```
To:
cc:
Subject:
In-reply-to: Your message of "Day, date time"
             <message-id>
--------
```

You can customize this formatting by modifying your own replcomps file (note that no spaces are allowed after the backslash (\) on each line). You will most likely need to reference the O'Reilly **MH** text for this task.

As you now know, replcomps controls the format of the message header. You can also customize the format of the message body for replies via a file called .mh_filter in your home directory. The O'Reilly **MH** book describes the different possible ways of setting this up. The contents of the default .mh_filter is:

```
body:component="> ",compwidth=0
```

F.2.4 forwcomps

The forwcomps file is used as the template header for forwarded messages. This file may look very similar to the components file, in fact the default files are identical. If you create your own components file and you want to use it for both sending and forwarding, include the following line in your .mh_profile file:

```
forw: -form components
```

You would only need a separate forwcomps file if you wanted the headers on forwarded and original mail to be different.

F.2.5 scan-form and inc-form

These files provide the format of the display when you scan and incorporate mail. If you aren't satisfied with the default scan format, we recommend that you simply set up a scan-form file, and use it for both situations (set both Scan: and inc: to scan-form in .mh_profile). The scan-form file's format is also used by **rcvtty** to provide messages. An example of a simple scan-form file showing incoming mail only and set up in a VMS-style format is shown below:

```
%4(msg)\
%21(addr{from})\
%02(mday{date})-%3(month{date})-%4(year{date})\
%02(hour{date}):%02(min{date})\
%{subject}
```

This example produces output in the format:

```
msg# sender_address  date  time  subject
```

Some sample output follows:

```
18 helpdesk              13-Nov-1995 16:30 1017699 Problem Report
19 lauri                 14-Nov-1995 10:56 VY0124 damaged
20 STUART@fnald.fnal.gov 14-Nov-1995 11:28 Problems with an fnaldb tape drive
21 wolbers@fsui01.fnal.g 14-Nov-1995 15:31 Department Meeting
```

An slightly more complicated example is shown below. It allows display of both incoming and outgoing messages:

```
%4(msg)\
%<(cur)+%| %>\
%<{forwarded}F%| %>\
%<{replied}R%| %>\
%<(mymbox{from}) <- %19(friendly{to}) %| -> %20(addr{from})%> \
 %02(mday{date})-%03(month{date})-%4(year{date})\
 %02(hour{date}):%02(min{date})\
 %{subject}
```

This example produces output in the format:

`msg# (arrow) (sender or recipient address)  date  time  subject`

The arrow indicates whether the mail is from you (<–) or to you (–>). If it is from you, the address field shows who you sent it to. Sample output looks like:

```
 258    -> bhat                     14-Oct-1995 12:15 batch job 341: <run_small> D
 260    <- joe@fnal                 13-Oct-1995 17:24 Program and Working Groups -
 261    <- lhn@sld.slac.stanford.edu 13-Oct-1995 16:31 e+w- Workshop, November 16-1
 262    <- martha@fnal              13-Oct-1995 17:24 re: revised mail chapter
 263    -> STROVINK@d0sfa           16-Oct-1995 09:21 Re: FYI re directions in NN
```

F.2.6 .maildelivery

The `.maildelivery` file controls how local delivery is performed. This file can be quite simple, or very complex. Within the file you can configure options to:

- restrict notification to mail received only from certain senders
- control notification format and content
- specify further processing such as delivery to one or more specific folders
- start up certain processes depending on information in the mail message header

Note that if you use the **mh** command **`inc`** (at the shell prompt) instead of incorporating within **exmh**, your mail will not be sorted. It will be filed into your `inbox` folder. The `.maildelivery` file is only used when you 1) incorporate within **exmh**, and 2) check the **PRESORT** box under **PREFERENCES**. The `.maildelivery` file is also used with autoincorporation (see section F.3).

Simple Case: Send All Incoming Mail to inbox Folder

If you simply want to send all incoming mail to your `inbox` folder, your `.maildelivery` file needs only to consist of the following one line:

```
* - ^ A "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

If you want to do something more sophisticated, continue reading this section.

File Format

Lines starting with `#` are comments. Fields are separated by one or more spaces. Each record in the file has the following fields in this order:

header field	the message header field that you're sorting on (e.g., `from`, `to`, `subject`)
match pattern	the text string found in the header field that you want to match (enclosed in quotes if it includes spaces). If the header field argument is `default` or `*`, this should be `-` (a dash).
action	what to do if pattern is matched. Set to `file` or `>` to send the matched message to a file. Set to `qpipe` or `^` to directly execute a command without invoking a Bourne shell. Set to `pipe` or `|` to invoke a Bourne shell to execute the command. If your command has special characters in it, you will need to use `pipe` or `|`, otherwise `qpipe` or `^` is more efficient.
result	a code indicating whether to set matched message to "delivered", or not. `A`: if field and pattern matched, action will be performed, and if successful, then set message to "delivered". `R`: same as A, but don't set message to "delivered" so that further entries in `.maildelivery` can act on it. `?`: perform the action if message not yet set to "delivered" by a previous entry. If action succeeds, set message to "delivered".
file or command	depending on action, this is either a file (if action says "send message to a file"), or a command (if action says "execute a command"). Always include full pathname.

Note that some **MH** systems expect this file to have the mode 644 (**`u+rw`**, **`g+r`**, **`o+r`**).

Examples

The following examples should provide you enough information to modify your `.maildelivery` file to suit your tastes. We'll start with an annotated version of the simple `.maildelivery` file discussed above in which all your incoming mail gets placed in `inbox`:

```
# execute rcvstore to put all as-yet-undelivered incoming mail into inbox (sets messages
# to "delivered")
#hdr  pattern  action  result  command
 *    -        ^       A       "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

Here is a slightly more complex example. The comments explain what is being done in each line:

```
# File mail with fsui in the To: line into file $HOME/fsui.log
To fsui file A fsui.log

# Pipe messages from daemon@fnck to the (local) program err-message-archive
From daemon@fnck pipe A err-message-archive

# Put anything with Sender: address "mh-users" into $HOME/mh.log if not yet filed
Sender mh-users file ? mh.log

# If the address is "uas-admin", send an acknowledgment copy back
addr uas-admin ^ A "/bin/resendR-r $(reply-to)"

# Destroy anything from daemon@fnck!
From daemon@fnck destroy A -

# For anything not matched yet, execute rcvstore to put into inbox folder
* - ^ ? "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

If you plan to create a significantly different set of criteria, you may need to see the man pages for **slocal** or the O'Reilly **MH** text referenced at the start of this appendix.

F.3 Unattended Autoincorporation

It is possible, though not recommended in a *clustered* environment (see section 2.7 about clusters), to use the `.forward` file to automatically deliver your mail from the system mail spool area directly into your mail area, whether or not you are reading mail, or whether or not you are even logged in! This is called *unattended autoincorporation*. It has a number of attractive features:

- since mail doesn't really stop in the system spool area, but rather is delivered directly to your `inbox` file, you are no longer restricted to reading and receiving mail on one particular node of a cluster
- reduces the startup time (and de-iconizing time) of **exmh** because it doesn't have to "catch up" on lots of spooled messages
- mail can be automatically filed to specified folders as it is being delivered (otherwise only possible for **exmh**, not for the line-mode **mh**)

However, unattended autoincorporation also has a number of drawbacks and caveats which make it a less attractive option:

- in a clustered environment where several separate nodes (hence several separate mail spool areas) see the same mail destination `inbox` file, file locking can become a serious problem. It can happen that a deadlock results in *no* mail being delivered while an overload of processes attempt to deliver mail in vain. This problem is especially serious for people who receive a lot of mail, and is exacerbated by the use of **rcvtty** (mail notification; see section F.1.2).
- **exmh** does not update the visual display as new mail gets autoincorporated. Therefore, while you may have saved the time of incorporating at startup, you will need to manually **RESCAN ALL FOLDERS** (under the **MORE...** button in the middle window) in order to see that you have new mail (and in which folders) when you are running **exmh**.

Because of the detrimental effect that unattended autoincorporation can have on a system, and the fact that when it breaks, mail is not delivered, **we do not recommend its use in most circumstances**. In particular, we believe that it should not be used in clustered environments. It is documented here for completeness only, and should not be taken as an endorsement.

To configure unattended autoincorporation on a node:

1) create a `.maildelivery` file describing your mail sorting preferences (see section F.2.6)

2) remove the line of your `.forward` (generally the first line) which contains something similar to `\username@node.fnal.gov`

3) replace it with `"| /usr/local/products/mh/current/lib/slocal -user username"` (include the double quotes)

where username is your login id.

F.4 Unattended Autoincorporation in AFS Environment

Proper ACL settings (AFS file permissions) must be made for autoincorporation in **MH**. The easiest way to set your mail folder ACLs is to run the utility **fixmailperms** from your `$HOME` directory. This allows the mail folders that you've designated for receiving mail to work with **slocal**, and it sets *all* your mail folders to be readable only by your account and the system administrator's. **fixmailperms** is part of the **ups** product **mailtools**. To see a brief explanation, enter:

```
% fixmailperms -help
```

The command is entered as:

```
% fixmailperms [folder_1 folder_2 ... folder_n]
```

The folder specifications are taken as relative to your `Mail` subdirectory. You need to specify all the folders into which you plan to automatically incorporate mail, and only those folders.

If you do not specify a minimum of one folder when you execute **fixmailperms**, unattended autoincorporation will not work.

For example, if you have a simple setup in which all incoming mail gets sent to your `inbox` folder, you'd enter:

```
% fixmailperms inbox
```

If this is satisfactory to you, skip to the next section. If instead you choose to set your ACLs individually (and not set your folders private), these are the ACLs you will need:

system:*anyuser* rl	for `$HOME`
system:*anyuser* l	for `Mail`
system:*anyuser* li	for `Mail/`*`folder`* [1]

Go to the directory whose ACL you want to change, and use the command syntax:

```
% fs setacl . system:anyuser xy
```

where ***xy*** is **rl**, **l**, or **li**, depending on the directory. Run **fs listacl** to see the current setting.

Refer to the *FNALU User's Guide,* GU0008, for details on ACL settings.

---

1. This folder should be your *inbox* or equivalent, e.g., `Mail/inbox`. You may need to set this for several folders, depending on the complexity of your autoincorporation set up.

Appendix G. mh Command Reference

This appendix provides an alphabetical reference to the subset of **mh** commands discussed in Chapter 11.

Remember to use the man pages if you need help on any of these commands.

Command	Description
% comp [-e *editor*]	Compose a message. This brings up a **vi** (or specified editor) session. Then save the message file, and exit the editor (see box below for more information).

Emergency **vi** exit: hit the **ESCAPE** key (or **<CTRL-[>**) and then enter:

:x	to save and exit, or
:q!	to quit without saving

At the `What Now?` prompt, you can enter a carriage return to get a list of options. Normally you will enter one of these options:

send	to send the message
quit	to save the message in `Mail/drafts` but not send it
quit -delete	to quit and not save

To send a message in which you want to include a file, compose the message, and use the "include file" command for **vi**:

:r *filename*

% folder +*new_folder_name*	Create a new folder.
% forw [*n*]	Forward the current message (or the specified message).
% inc	Incorporate new incoming message(s).
% next	Display the next message.
% pick -help	Display a list of pick options.
% pick *options*	Select messages by content, date, sender, recipient, etc.
% prev	Display the previous message.
% refile +*folder* [*n m ...*]	Move the current message (or specified messages) from the current folder to a different folder.
% repl [*n*]	Reply to the current (or specified) message. By default, everyone who got the original message gets a copy (the sender, plus everyone in `To:` and `cc:`). Respond to the `What Now?` prompt.
% repl_inc [*n*]	Like **repl**, but include the original message. Respond to the `What Now?` prompt.
% rmf +*folder*	Delete the specified folder.

% rmm [*n m* ...]	Delete the current (or specified) message(s).
% scan [+*folder_name*]	Display message headers from current folder (or ***folder_name***, and set this folder to current).
% send *file*	Send ***file*** to all the destinations defined, namely `To:` at a minimum, and `cc:`, if it's filled it in. (If you change your header to include `Bcc:` (blind copy), `Fcc:` (folder copy), it will send it to addresses in these categories as well.)
% setup mh	Setup the **mh** product.
% show +*folder_name*	Display the current message in ***folder_name*** and set this folder to current.
% show [*n*]	Display the current (or specified) message.
% show *n* | flpr -q *queue*	Print message number ***n*** on ***queue***.
% show *n* > *filename*	Extract current (or specified) message into a file.
% show -showproc mhl *n* | flpr -q *queue*	Print and format the message a little differently so that it's easier to read (and nicer for printing), and pipe it to **more**. Then send it to the printer.
% show -showproc pr *m n* ... | flpr -q *queue*	Print messages and separate them onto successive pages. Then send it to the printer.

Appendix H. Mail Conversion from VMS

This appendix is intended to guide you through a mail conversion process from VMS to the UNIX **MH** (Message Handling) system. Two options for conversion are presented.

H.1 Preparation for Conversion

Moving your VMS mail folders to UNIX is a process initiated from the UNIX system. It uses **ftp** (see section 12.1.1) or the **rcp** and **rsh** commands (see sections 12.1.2 and 12.1.3). Therefore, before you start, you need to set up two files:

- on your VMS machine (the remote node for this operation), a file named `.RHOSTS` that includes records for the UNIX node you're using as your mail destination. Section 12.1.3 describes the contents of this file, appropriate for both UNIX and VMS.
- `.mh_profile` on your UNIX mail node. To create this automatically, run **setup mh** or **setup exmh**. The file will contain the line `Path: Mail`, and that's all you need.

Make sure that v2_2 of **mailtools** is current. Enter:

```
% ups list -a mailtools
```

to see which version is current. If it is not v2_2 or higher, select the right version explicitly by typing:

```
% setup mailtools v2_2
```

If you are moving from FNALV, you should encounter no disk quota problem. The `USR$SCRATCH` area on FNALV is used to hold temporary files. If that is not set up on your VMS machine, you need to ensure that you have enough disk space on your UNIX system and enough disk quota on the VMS node to hold duplicate copies of all of your mail. For VMS, we suggest the commands:

```
$ SHOW QUOTA
```

to see how much space you have available, and

```
$ DIR/SIZE=ALL/GRAND *.MAI;*
```

(from the directory containing your mail) to see how much space you'll need.

For the UNIX side, as usual, it's not so cut and dried. You may be able to use **quota -v** or **df** to provide you the information; check the man pages on your system. On some systems (for instance FNALU running AFS), individual quotas may not be established.

H.2 Choosing the Process to Use

Now you're ready to convert your VMS mail folders to UNIX **MH** mail folders. You can use a "semi-automatic" process, or a totally automatic one.

For either process, the mail messages will be read and extracted into individual filenames in folders for use in **MH** or **exmh**. The filenames are incremented numbers starting at one (1), or the next highest available number for an existing **MH** folder. The **MH** folder name will be the same as the name of the mail folder on the VMS node (unless you rename it during a semi-automatic transfer); e.g., folder `SMITH` will become `/Mail/smith`.

The script used in both processes is `fvms2mh`. For full documentation on the **`fvms2mh`** command enter:

```
% fvms2mh -h |less
```

As mentioned above for those of you for whom `USR$SCRATCH:[`*`username`*`]` is not available or lacks sufficient quota, you need to have enough space on VMS for duplicate copies of your mail while you're doing the conversion. If you use the automatic method, you cannot use any scratch space (there is no "standard" way of assigning scratch space, therefore the automatic method has no way of knowing where it is). However, if you use the semi-automatic method, you can use scratch space for the EXTRACT/ALL second copy, the copy you bring over to your UNIX node.

H.3 Using the Semi-Automatic Process

This process won't take long unless you have many mail folders on VMS to convert. You can see and control all the steps using this method. Perform the following the steps:

1) Log in to the VMS node where your mail is kept.

2) For each folder in your MAIL file (and for each MAIL file that you maintain), enter:

```
MAIL> SET FOLDER foldername
```

```
MAIL> EXTRACT/ALL tempfile
```

3) Log in to the UNIX node where you intend to store your mail and use **ftp** or **rcp** to copy over to your `$HOME` directory the ***tempfile*** containing the extracted VMS mail messages. Sections 12.1.1 and 12.1.2 describe these file transfer utilities.

Do not create a `$HOME/Mail` directory and copy files directly there! Step 4 takes care of this.

4) Now you're ready to convert the copied mail files to **MH** mail folders. If for some reason you haven't run either **`setup mh`** or **`setup exmh`** on UNIX, do so now. Then, for each ***tempfile*** you copied over, enter:

```
% fvms2mh tempfile
```

H.4 Using the Automatic Process

This process simply automates all the steps of the semi-automatic process.

The `fvms2mh` script invoked in its simplest form:

```
% fvms2mh -vms vmsnode
```

performs the “EXTRACT/ALL” command on each of the VMS mail folders in your `MAIL.MAI` file in your default mail directory. Each folder is extracted into a separate file on the VMS node. Each file is then copied to the UNIX node and converted into **MH** format.

Note that ***vmsnode*** must be an individual node, not a cluster alias.

If you have other mail files, you’ll need to convert them separately. For example, if all of your mail files are in the same (default) mail directory on ***vmsnode***, you can enter the command as:

```
% fvms2mh -vms vmsnode -file "\*"
```

We use the UNIX quoting syntax here to pass the asterisk wildcard to VMS so that all mail files are handled.

If your login id on VMS is different from your UNIX id, you’ll need to add the **`-l`** switch:

```
% fvms2mh -vms vmsnode -l vms_login_id
```

Appendix I. Programming Examples

This appendix contains examples of programs that illustrate information presented in Chapters 13 and 14.

I.1 Interfacing C and FORTRAN

The following C/FORTRAN program illustrates some of the points discussed in section 13.9. Our executable is called `cfort`. We assume use of the **make** utility and Makefiles in this example. These are described in chapter 14.

The Makefile:

```
SHELL=/bin/sh
cfort : cf.o fc.o ; f77 -o cfort cf.o fc.o
fc.o  : fc.f      ; f77 -c fc.f
cf.o  : cf.c      ; cc  -c cf.c
```

The FORTRAN routine:

```
      PROGRAM CFORT

*   Test C/FORTRAN mixed programming techniques

*   Place COMMON variables in order REAL*8 , REAL/INTEGER , INTEGER*16 , BYTE
      COMMON /MIXED/ DCOMM, FCOMM, ICOMM
      REAL*8       DCOMM
      REAL         FCOMM
      INTEGER      ICOMM(2)

*   Arguments used in ICF function call
      INTEGER      INTE(2)
      REAL         FLO
      CHARACTER*10 STR

*   Initialize

      DCOMM    = 1.
      FCOMM    = 2.
      ICOMM(1) = 3
      ICOMM(2) = 30
      INTE(1)  = 4
      INTE(2)  = 40
      FLO      = 5.
      STR      = 'From f77'

C   Write initial values

      WRITE (*,*) ' '
      WRITE (*,*) ' / / / From FORTRAN / / / '
      WRITE (*,*) ' DCOMM  = ', DCOMM
      WRITE (*,*) ' FCOMM  = ', FCOMM
```

```
      WRITE (*,*) ' ICOMM  = ', ICOMM
      WRITE (*,*) ' INTE   = ', INTE
      WRITE (*,*) ' FLO    = ', FLO
      WRITE (*,*) ' STR    = ', STR
      WRITE (*,*) ' '

C   Use C function icf_, which prints, changes, and reprints COMMON and
C   Argument values

      IRET = ICF ( INTE , FLO , STR )

C   Write values after call to ICF
      WRITE (*,*) ' / / / Back in FORTRAN / / / '
      WRITE (*,*) ' DCOMM  = ', DCOMM
      WRITE (*,*) ' FCOMM  = ', FCOMM
      WRITE (*,*) ' ICOMM  = ', ICOMM
      WRITE (*,*) ' INTE   = ', INTE
      WRITE (*,*) ' FLO    = ', FLO
      WRITE (*,*) ' STR    = ', STR
      WRITE (*,*) ' IRET   = ', IRET
      WRITE (*,*) ' '

      STOP
      END
```

The C Routine (`icf_`):

```
           /*  Test of mixed C/FORTRAN features    */

int icf_( int* inte , float* flo , char* str , int lenstr ) {
extern struct {
    double dcomm ;
    float  fcomm ;
    int    icomm[2] ;
    } mixed_ ;

printf ( "  / / / Into C / / /\n\
  DCOMM  = %f\n\
  FCOMM  = %f\n\
  ICOMM  = %d %d\n\
  INTE   = %d %d\n\
  FLO    = %f\n\
  STR    = '%.10s'\n\
  LENSTR = %d\n\n"
  ,  mixed_.dcomm , mixed_.fcomm , mixed_.icomm[0] , mixed_.icomm[1]
  , *inte , *(inte+1) , *flo , str , lenstr ) ;

mixed_.dcomm    = 10000. ;
mixed_.fcomm    = 20000. ;
mixed_.icomm[0] = 30000  ;
mixed_.icomm[1] = 330000 ;
*inte           = 40000  ;
*(inte+1)       = 440000 ;
*flo            = 50000. ;

strcpy ( str , "Out of C" ) ;

printf ( "  / / / Out of C / / /\n\
  DCOMM  = %f\n\
  FCOMM  = %f\n\
  ICOMM  = %d %d\n\
  INTE   = %d\n\
  FLO    = %f\n\
  STR    = '%.10s'\n\n"
```

```
  ,  mixed_.dcomm , mixed_.fcomm , mixed_.icomm[0] , mixed_.icomm[1]
  , *inte , *flo , str ) ;

return ( lenstr ) ;

}
```

Program Execution (see section 13.10):

```
  / / / From FORTRAN / / /
  DCOMM  =    1.000000000000000
  FCOMM  =    2.000000
  ICOMM  =            3          30
  INTE   =            4          40
  FLO    =    5.000000
  STR    = From f77

  / / / Into C / / /
  DCOMM  = 1.000000
  FCOMM  = 2.000000
  ICOMM  = 3 30
  INTE   = 4 40
  FLO    = 5.000000
  STR    = `From f77  `
  LENSTR = 10

  / / / Out of C / / /
  DCOMM  = 10000.000000
  FCOMM  = 20000.000000
  ICOMM  = 30000 330000
  INTE   = 40000
  FLO    = 50000.000000
  STR    = `Out of C'

  / / / Back in FORTRAN / / /
  DCOMM  =    10000.00000000000
  FCOMM  =    20000.00
  ICOMM  =        30000      330000
  INTE   =        40000      440000
  FLO    =    50000.00
  STR    = Out of C
  IRET   =           10
```

I.2 Makefiles and the make Process

I.2.1 A Simple make Process

Our "Hello World" example is quite simple, and illustrates many of the features we have discussed in Chapter 14. First we'll run a file listing to see the files that are involved:

```
-rw-r--r--   1 aheavey  g020         515 Dec 20 16:27 Makefile
-rw-r--r--   1 aheavey  g020          91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model2.f
```

The contents of the FORTRAN source files that will be used as required files follows:

```
<fsui01> cat mcprog.f
      WRITE (*,*) ` Hello world `
      CALL MODEL1
      CALL MODEL2
      STOP
```

```
      END
```

```
<fsui01> cat model1.f
      SUBROUTINE MODEL1
      WRITE (*,*) ' Hello from MODEL 1 '
      RETURN
      END
```

```
<fsui01> cat model2.f
      SUBROUTINE MODEL2
      WRITE (*,*) ' Hello from MODEL 2 '
      RETURN
      END
```

Our Makefile illustrates the use of macro definitions, iterative target defintions, the use of a library in addition to other required files, comment lines, and macro expansion (described in the man pages). It provides two levels of "housekeeping" at the end, *clean* to remove just the "stray" files, and *clobber*, which gets rid of all resulting files thus allowing you to run **make** again under the same initial conditions.

```
<fsui01> cat Makefile
# use /bin/sh within make
SHELL = /bin/sh

#   Standard FORTRAN command and flags
F77    = f77
FFLAGS = -O2

# Build the 'mcprog' program from mcprog.f and libmc.a
mcprog   : mcprog.f   libmc.a ; $(F77) $(FFLAGS) -o mcprog mcprog.f libmc.a

# Move any missing modules into libmc.a
libmc.a  : model1.o  model2.o ; ar -r libmc.a $(?)

# Compile missing or out-of-date modules
model1.o : model1.f ;     $(F77) $(FFLAGS) -c $(?)

model2.o : model2.f ;     $(F77) $(FFLAGS) -c $(?)

clean    : ; rm -f *.o

clobber : ; rm -f *.o ; rm -f mcprog ; rm -f libmc.a
```

Run **make** without specifying a target, and it will use the first target it encounters, `mcprog`. Notice that **make** displays its progress on the screen, creating the library `libmc.a` prior to building the executable `mcprog`:

```
<fsui01> make
f77 -O2 -c model1.f model1.f
model1.f:
        model1:
model1.f:
        model1:
f77 -O2 -c model2.f model2.f
model2.f:
        model2:
model2.f:
        model2:
ar -r libmc.a model1.o model2.o
ar: creating libmc.a
f77 -O2 -o mcprog mcprog.f libmc.a
mcprog.f:
 MAIN:
```

Run another file listing to see that our target and some intermediate files were created:

```
-rwxr-xr-x   1 aheavey  g020       13452 Dec 20 16:34 mcprog*
-rw-r--r--   1 aheavey  g020        2712 Dec 20 16:34 libmc.a
-rw-r--r--   1 aheavey  g020        1248 Dec 20 16:34 model2.o
-rw-r--r--   1 aheavey  g020        1248 Dec 20 16:34 model1.o
-rw-r--r--   1 aheavey  g020          91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model2.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020         515 Dec 20 16:27 Makefile
```

Run the executable, `mcprog`:

```
<fsui01> mcprog
  Hello world
  Hello from MODEL 1
  Hello from MODEL 2
```

Run **`make clean`** to remove the object files:

```
<fsui01> make clean
rm -f *.o
```

A directory listing shows the object files were deleted:

```
-rwxr-xr-x   1 aheavey  g020       13452 Dec 20 16:34 mcprog*
-rw-r--r--   1 aheavey  g020        2712 Dec 20 16:34 libmc.a
-rw-r--r--   1 aheavey  g020          91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model2.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020         515 Dec 20 16:27 Makefile
```

Finally, run **`make clobber`** to remove all the created files:

```
<fsui01> make clobber
rm -f *.o ; rm -f mcprog ; rm -f libmc.a
```

... and check the final directory listing, which should match the original one:

```
-rw-r--r--   1 aheavey  g020         515 Dec 20 16:27 Makefile
-rw-r--r--   1 aheavey  g020          91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model2.f
```

I.2.2 A Physics Makefile

The following is an example physics Makefile with FORTRAN and C sources.

```
###############
#   MACROS    #
###############

# Always force the shell to be /bin/sh, for portability

SHELL=/bin/sh

# some variables come from the environment existing when make is run

#    CRNLIB       (from 'setup cern')
#    HISTO_DIR    (from 'setup histo')
#    MCFAST_DIR   (a group library from 'source ~bphyslib/mcfast/mcenv')
#    MCFIO_DIR    (a group library from 'source ~bphyslib/mcfast/mcenv')

# macros for directories containing INCLUDE files:

INC1 = $(MCFAST_DIR)/inc/event
```

```
INC2 = $(STDHEP_INC)

# macro for MCFAST object library directory

LIB = $(MCFAST_DIR)/lib.IRIX

# macro listing all USER object routines

OBJS  = \
  usr_analysis.o \
  usr_before_trigger.o \
  assign_off_mass.o \
  costh.o \
  fin_init.o \
  find_b_psiks.o

# macros for object libraries from products HISTO and CERN

HISTLIB = $(HISTO_DIR)/lib/libFHistoHB.a
PACKLIB = $(CRNLIB)/libpacklib.a
MATHLIB = $(CRNLIB)/libmathlib.a
KERNLIB = $(CRNLIB)/libkernlib.a

# macro for the SGI exception handler library

FP_EXCEPTION = /usr/lib/libfpe.a

# macro for all linked libraries

LOADLIBS= \
  $(LIB)/libgen.a      \
  $(LIB)/libdatafile.a \
  $(LIB)/libio.a       \
  $(LIB)/libutil.a     \
  $(MCFIO_DIR)/lib.IRIX/libFmcfio.a \
  $(HISTLIB) \
  $(PACKLIB) \
  $(MATHLIB) \
  $(KERNLIB) \
  $(FP_EXCEPTION)

# macros for compiler option flags

CFLAGS = -g
FFLAGS = -g -I$(INC1) -I$(INC2)

###############
#   TARGETS   #
###############

# f77 is used to run the link editor here.
# FORTRAN and C compilation is done via the rules defined below.

mcfast: $(OBJS) $(LOADLIBS); f77 -o mcfast $(OBJS) $(LOADLIBS)

###############
#  SUFFIXES   #
###############

# You must list all suffixes used in this Makefile's rules

.SUFFIXES : .c .F .f .o

###############
#    RULES    #
```

```
###############

# RULES for building *.o files from *.f or *.c

#     The $? internal macro expands to the list of *.f or *.c files which are
#     newer than their corresponding *.o file.

#     Macros CFLAGS and FFLAGS are defined above with compiler options

.f.o: ; f77 -c $(FFLAGS) $?

.c.o: ; cc  -c $(CFLAGS) $?
```

Index

Symbols

4-5
! 5-2, 5-5
- 6-4
!! 5-5
!$ 5-5
!? 5-5
!?text? 5-5
!n 5-5
!text 5-5
" 2-4, 5-14
4-4, 8-10
#! 4-4
$ 2-3, 2-4, 4-2, 5-2, 5-4, 5-5, 5-14, 8-2
% 2-3, 4-2, 5-16
& 2-5, 5-15
&& 2-5
***** 5-14, 6-5
+ 5-14
, 6-4
. 4-5, 5-14, 6-1, 6-3, 6-4, 8-3, 8-8
.* 5-14
.. 6-1, 6-3, 6-4, 6-15
./ 4-5, 8-3
/ 2-4, 6-1, 6-3, 6-4
; 2-5
< 5-7, 5-9
> 5-7, 5-8, 5-9
>! 5-7, 5-8
>& 5-7, 5-8
>&! 5-7, 5-8
>> 5-7, 5-8, 5-9
>>& 5-7, 5-8
? 5-14, 6-4
@ 8-10
[] 5-14
**** 2-4, 2-5, 5-2, 5-14, 13-10
^ 5-14, 8-10
^- 8-10
^> 8-10
_ 6-4
| 2-5, 5-7, 5-9
|| 2-5
~ 6-2, 6-3, 8-3, 13-14
' 2-4

Hidden Files

.cshrc file 2-2, 4-5, 6-4, 6-18, 8-1, 8-5, 8-6, 8-7, E-1
.exmh_defaults file 11-10
.flpprc file 7-3
.forward file 11-2, 11-8, 11-20, 11-21, F-2, F-3
.login file 2-2, 2-3, 2-4, 5-7, 6-4, 8-5, 8-6, 8-7, 8-9, E-6, F-2
.logout file 2-2, 6-4, 8-5, 8-6
.mailcap file 3-5, 8-8, 8-9
    format 8-8
.maildelivery file 11-21, F-6
    examples F-7
    format F-7
.mh_filter file 11-14, 11-21, 11-23
.mh_profile file 11-8, 11-10, 11-13, 11-15, 11-20, 11-23, F-3, H-1
.mime.types file 8-8
    format 8-9
.pinerc 11-6
.pinerc file 11-5
.plan file 3-7, 3-8
.profile file 2-2, 2-3, 2-4, 5-7, 6-18, 8-7, 8-9, E-11, F-2
.rhosts file 12-3, 12-5, 17-3, A-2, H-1
    sample file 12-4
.shrc file 2-2, 4-5, 8-7, E-17

A

a2ps 7-3
Absolute pathname 6-1
Access a UNIX system 2-1
Access Control Lists 6-18
Access mode 6-16
Access permission 6-16
Accessing **ups** products 9-2
ACL rights 6-18
ACL settings
    list D-3
ACLs 6-18, D-3, F-9
    list F-9
    set F-9
Add command to search path 5-7
Adobe acrobat 8-9
AFS 1-2, 2-5, 2-6
    ACLs 6-18, F-9
    CLUBS 17-4
    default shell 4-2
    determine if installed on system 2-5

FNALU 17-2
mail forwarding 11-2
translator mode 2-5
AFS authentication D-3
AFS directory
list ACLs D-3
set ACLs D-3
AFS file permissions
fixmailperms F-9
AFS password 2-7, 17-5
AFS permissions
change 6-19
list 6-19
AFS token
discard D-3
expiration 2-7, 17-2
obtain D-3
AIX 1-9, 13-14
ali 11-23
Alias 5-2
alias 5-1, 6-13, 8-5, D-3
Alias file 11-21, 11-23
Andrew File System 2-5
Anonymous **ftp** 9-5
ANSI C standard 13-2
ANSI labeled tapes 16-7
ANSI standards 13-2
answerbook 3-4
Applications 9-1, 9-2
apropos 3-3, D-3
ar D-4
Archive tapes D-3
Arguments 5-4, 13-18
asa 13-12
Assembler 13-1, 13-2
Assign files D-2
at 5-17
awk 5-10, 5-11, 5-12, 5-14, 13-3, C-1
features 5-12
main input loop C-1
pattern matching features 5-13
program structure and organization C-1
programming model 5-13, C-1

B

Back quotes 2-4
Background jobs 5-15
output redirection 5-15
Background processes 5-1, 5-15
Backslash character 2-4, 13-10
Backspace character 2-3
Backspace key 5-4
bash 4-3
Basic flavor 9-5
Batch job
kill D-4
stop D-4
submit 17-4, D-4
Batch queue
check D-4
Batch system
CLUBS 17-4
fbatch 17-3
FNALU 17-3
LoadLeveler 17-3
Berkeley Mail 11-1, 11-7, 11-24
read a message 11-24
send a file 11-24
send a message 11-24
bg 5-16
biff/comsat F-2
Bourne shell family 4-1, 8-5
. 8-8
.profile 2-2
.shrc 2-2
alias 8-7
execute command to affect current shell 8-8
Fermi files 8-7
shell functions 8-5
Bourne shell variables 8-2
define 8-2
export to environment 8-2
Browser commands 3-7
Browsers 3-4, 8-8
Buffered I/O 16-7
bufio 13-14
Built-in command help 5-2
Built-in commands 4-3, 5-1
platform-specific 5-2

C

C 6-3, 13-1, 13-4, 13-18
arguments 13-18
cache usage 13-16
calling FORTRAN routines 13-18
compiler options 13-6
dbx 13-19
debugging 13-8, 13-19
execute program D-4
extern struct 13-18
external names 13-17
feedback 13-16
file extensions 13-4
floating point errors 13-15
Gnu C 13-2
indexes 13-17
inlining 13-16
interfacing with FORTRAN 13-16
library references 13-5
link editor 13-4
linking 13-18
list active compiler options 13-5
load map 13-9
pointers 13-18
speed optimization 13-8, 13-15
variables 13-17
word length 13-16
C compilers 13-4
C language 13-1, 13-2
C preprocessor 13-10
C programs 13-4
C shell family 4-1
.cshrc 2-2
.login 2-2

alias 8-4
execute commands to affect current shell 8-7
Fermi files 8-5
fork new process 8-6
invoke new shell 8-6
variables set at startup 8-2
C++ 13-1, 13-2, 13-6
ANSI standards 13-2
compiler options 13-6
dbx 13-19
debugging 13-19
file extensions 13-6
ISO standards 13-2
C++ language 13-1
Cancel tape mount request 16-5
Case sensitivity 13-14
filenames 6-4
Case-sensitivity
commands 5-4
CASEVision 13-21
cat 5-9, 6-7, 7-4, D-1, D-2
format 6-7
CC 13-2, 13-6
cc 13-4
cd 2-5, 5-1, 5-2, 6-1, 6-2, 6-3, 6-15, D-1
CDF UNIX information 1-1
cedit 5-5
C-FORTRAN
linking 13-18
mixed I/O 13-18
Change directory D-1
Change file ownership D-3
Change file permissions D-2
Change mail folders 11-18
Change mode 6-17
Change password 2-6, D-3
Change terminal settings D-4
Character class 6-5
Character matching 5-14, 6-4
Characters, special 2-4, 2-5, 6-4
Check batch queue D-4
CHECKIN 15-3
chgrp D-3
Child process 5-1, 8-1
chmod 4-5, 6-16, 6-17, 6-19, D-2, F-2
absolute form 6-17
alternate form 6-17
Choosing a UNIX platform 1-8
chown D-3
CLUBS 17-1, 17-4
access to 17-4
accounts 17-1, 17-4
AFS 17-4
AFS password 17-5
batch job process 17-6
batch system 17-3, 17-4
compute worker nodes 17-5
data server nodes 17-5
fbatch 17-4
FNCLUB 17-5
import/export subsystem 17-5
Kerberos password 17-5
LoadLeveler 17-3, 17-4
LoadLeveler command files 17-6
scheduler 17-5
with FNALU 17-4
clubsout 17-6
clubstools 17-6
clubsout 17-6
jobstat 17-6
ndflist 17-6
Cluster 2-6
cmp D-2
CMS 15-1
Code management 15-1
concurrent development 15-1
Command
built-in 5-1
Command arguments 5-4
optional 5-3
Command files 4-1
Command format 5-3
Command interpretation by shell 5-2
Command interpreter 1-7
Command language
Tcl/Tk 13-3
Command line editing 4-3, 5-4
Command line errors
correct typing 5-4
Command options 5-4
Command procedure, execute D-4
Command recall 5-4
cedit 5-5
Command separator 5-4
& 2-5
&& 2-5
; 2-5
| 2-5
|| 2-5
Commands 1-7, 5-1, 9-1
alias 5-2
ambiguous 6-3
arguments 5-3, 5-4
built-in 4-3, 5-1
case-sensitivity 5-4
conditional 5-4
continue to next line 5-4
determining executable file 6-3
directory shortcuts 6-3
edit 5-4
execute remotely 12-5
filter 5-10
find 5-7
group 5-4
group options 5-4
history list 4-3
interactive 5-1
locate via **path** variable 8-3
looping 5-4
option listing 3-2
options 5-3, 5-4
parentheses in 5-2
platform differences 5-3
recall 5-4
scripted 5-1
sequence of 4-4
type ahead 5-4
comp 11-12, F-3, G-1
Compare files D-2
Comparison of editors 10-1

Comparison of shells 4-3
Compile FORTRAN program D-4
Compiled language environments B-1
Compilers
C 13-4
C++ 13-2
FORTRAN 13-4
Completion mechanism 4-3
components file 11-10, 11-21, 11-23, F-4, F-5
customize F-4
Compose a mail message 11-10, G-1
compress 6-10
Compression, file 6-10
Computing Division support 1-9
Computing Division WWW page 1-10
Computing resources 17-1
Computing systems updates 3-7
Concatenate files 6-7, D-2
Concurrent Versions System 15-1
Conditional commands 5-4
Configuration files for MH 11-20
Conformance levels, FUE B-1
Connect to remote host 12-3
Connect to remote node D-4
Continue commands to new line 2-4
Control characters 2-3, 8-10
backspace 2-3
ctrl-c 2-3
ctrl-d 2-4
ctrl-o 2-3
ctrl-q 2-3
ctrl-r 2-3
ctrl-s 2-3
ctrl-w 2-3
ctrl-x 2-3
ctrl-z 2-3
DEL 2-3
Conventions, notational 1-2
Convert and copy file D-2
Converting from VMS A-1, H-1
Converting mail H-1
Copy a file 6-8, D-2
Copy file to remote node D-4
Copy set of files D-4
Core file 13-19
Core FUE B-1
Correct command typos 5-4
Count characters in file 6-13
Count lines of file 6-13
Count words in file 6-13
cp 6-8, 6-9, 8-4, 11-17, D-2
format and options 6-8
cpp 13-10, 15-3
Create a directory 6-15, D-1
Create a library D-4
Create mail folders 11-18
Create **mh** folder G-1
cron 5-18
crontab 5-18
csh 4-3, 5-5
reexecution commands 5-5
ctrl-] 5-6
ctrl-c 2-3
ctrl-d 2-4, 4-2, 5-7
ctrl-o 2-3
ctrl-q 2-3
ctrl-r 2-3
ctrl-s 2-3
ctrl-w 2-3
ctrl-x 2-3
ctrl-z 2-3, 5-17
Current directory 6-1, 6-4, 6-15
Current shell, determine 4-1
Customize mail message headers F-3
Customize mail reply header 11-14
Customized code, storage of 8-8
cvd 13-21
CVS 15-1
common commands 15-2
documentation 15-1
editor 15-1
cvs checkout 15-2
cvs commit 15-2
cvs export 15-2
cvs import 15-2
cvs rtag 15-2

D

D0 UNIX information 1-1
DAFT
FORTRAN interface 16-7
DAFT 13-14, 13-15, 16-7
DART
features and functions 17-7
DART 16-7, 17-1, 17-7
features and functions 17-7
hardware and software 17-7
requirements 17-7
Data acquisition system
DART 17-7
Database manipulation 5-11
Date D-3
date D-3
dbx 13-19, D-4
commands 13-19
format 13-19
dd D-2
Debug program D-4
Debugging 13-19
CASEVision 13-21
dbx 13-19
purify 13-20
Default home page 3-6
Default login files E-1
.cshrc E-1
.login E-6
.profile E-11
.shrc E-17
fermi.cshrc E-2
fermi.login E-7
fermi.profile E-12
fermi.shrc E-19
setpath.csh E-3
setpath.sh E-16
setups.csh E-4
setups.sh E-21
Default shell 8-4

Default shell (AFS,NIS) 4-2
DEL 2-3
Delete a directory 6-10, 6-16, D-2
Delete a file 6-10, D-2
Delete a mail message 11-17, G-2
Delete key 5-4
Delete mail folders 11-18
Delete **mh** folder G-1
Delete print entry D-3
Delimiters 2-5
Delivery of mail, customize 11-21
Devices
    logical 5-7
    null 5-8, 5-9
df 2-5, D-3, H-1
diff D-2
Directories, **ups** product 9-2
Directory 3-7
    change D-1
    change working directory 6-15
    create 6-15, D-1
    current 6-1
    delete 6-16, D-2
    home 6-2, 6-3, 6-15, 13-14
    link 6-7, 6-9
    list contents 6-6, 6-14
    list subdirectories D-1
    make 6-15
    manipulate 6-14
    mode 6-7
    move 6-16
    parent 6-15
    permissions 6-19
    print working directory 6-14
    reference 6-9
    remove 6-16
    root 2-4, 2-5
    shortcuts 6-3
    working 6-1
Directory files 6-4
Directory list format 6-7
Directory permissions 6-16
    AFS 6-18
    change 6-17
Directory protection 6-16
Directory shortcut commands 6-2
Directory, display current D-1
Disable messages D-4
Disk I/O 16-7
    FORTRAN format 16-7
    read/write DART-formatted records 16-7
Disk quota
    display D-3
Disk space
    display D-3
Display current directory D-1
Display current terminal settings 2-4
Display date and time D-3
Display disk quota D-3
Display environment settings D-3
Display file contents 6-7, D-1
Display free disk space D-3
Display logged in users D-3
Display message G-2
Display message headers G-2
Display multimedia files 8-8
Display next message G-1
Display output 5-10
Display previous message G-1
Display print entry D-3
Display print queue D-3
Display processes D-3
Display status of processes 5-1
Display to screen D-3
DISPLAY variable 3-5, 8-3, 10-3, 16-6
Display variables
    set 8-2
Display, restart 2-3
Distributed file systems 2-5
    AFS 2-5
Distribution lists for mail 11-21, 11-23
Distribution, products 9-1
Document numbers, Fermilab 1-10
Documents, Fermilab 1-10
Domain name 3-4
domainname 2-6
Dot (in pathname) 6-1
Double quotes 2-4, 5-2
Double-sided printing 7-4
du D-3
Dump a file 6-13, D-2
Duplex print mode 7-4
dxbook 3-4

E

echo 4-1, 5-2, 6-2, 6-3, 6-5, 6-10, 8-1, 8-2, D-3
ed 5-14
Edit commands 5-4
Editing
    command line 4-3
    multiline 4-3
EDITOR variable 5-6
Editors 1-8, 10-1
    comparison of 10-1
EDT+ 7-4, 10-1
    pros and cons 10-3
    setup 10-7
EDT, UNIX version of 10-1
egrep 5-14
emacs 5-6, 7-4, 10-1
    commands 10-4
    flow control 10-5
    help facility 10-4
    key bindings 10-5
    pros and cons 10-2
End-of-file 5-7
Entry points in library D-4
env 4-1, 6-13, 8-2
ENV variable 8-7
Environment
    customize 8-5
Environment for UNIX process 8-1
Environment settings
    display D-3
Environment variables 8-1, 8-2
Environments
    compiled language B-1

language support B-1
eof 2-2, 2-3, 2-4, 5-7
EVE, UNIX version of 10-1
exec 5-1
Executable files 5-7
Executables 5-1
run from current directory 8-3
Execute a program 13-18, D-4
Execute command procedure D-4
Execute permission 6-7, 6-16
directory 6-19
file 6-16
Execute shell script 4-5, D-4
exit 2-2, 4-2, D-3
Exit a shell 4-2
Exit UNIX 2-2
exmh 8-8, 11-1, 11-4
.maildelivery 11-21
abort a message 11-11
alias file 11-21, 11-23
change current folder 11-18
change editor 11-12
commit an operation 11-17, 11-19
compose a message 11-10
configuration files 11-20
create drafts folder 11-8
create new folder 11-18
customize 11-1
customize message headers F-3
default editor 11-11
define basic configuration 11-20
define forwarding header format 11-21
define message header format 11-21
define reply header format 11-21
display a message 11-13
displayed message header 11-24
distribution lists 11-21, 11-23
extract a message 11-17
folder display order 11-24
folder window 11-9
font size 11-10
forward a message 11-16
glimpse 11-20
immediate message incorporation 11-22
inc-form 11-21
incorporate messages 11-13
insert file in message 11-11
interfaces 11-4
manual message incorporation 11-13
message display window 11-9
message header window 11-9
nested folders 11-18
periodic message incorporation 11-13, 11-22
pick messages by attributes 11-19
print a message 11-16
print multiple messages 11-16
refile messages 11-19
remove messages 11-17
reply to a message 11-15
scan-form 11-21
search for messages 11-19
sedit 11-11
select multiple message headers 11-10
send a message 11-11
sequence of picked messages 11-19
set up and invoke 11-8
unattended message incorporation 11-13
use of the window 11-9
What Now? dialog box 11-12
Expand 15-3
Expansion, file 6-10
Expiration of Kerberos token 2-7
export 2-3, 8-2
Expressions, regular 5-13
Extended flavor 9-5
Extensions on filenames 6-4
External names 13-17
Extract
tar file 6-11
Extract a message 11-17

F

f77 13-4, 13-14, D-4
option passing 13-5
fbatch 17-3, 17-4, 17-5, 17-6, D-4
fbatch_cancel 17-6, D-4
fbatch_q 17-6, D-4
fbatch_submit 17-4, 17-6, D-4
Fermi files 1-9, 2-2, 8-5, E-1
Bourne shell family 8-7
.profile 8-7
.shrc 8-7
fermi.profile 8-7
fermi.shrc 8-7
C shell family 8-5
.cshrc 8-5, 8-6
.login 8-5, 8-6
.logout 8-5, 8-6
fermi.cshrc 8-6
fermi.login 8-6
Fermi UNIX Environment 1-9
fermi.cshrc file 8-6, E-2
fermi.login file 8-6, E-7, F-2
fermi.profile file 8-7, E-12, F-2
fermi.shrc file 8-7, E-19
Fermilab at Work WWW page 1-10, 3-6
Fermilab computing resources 17-1
Fermilab document numbers 1-10
Fermilab documents 1-10
Fermilab home page 3-6
Fermilab mail server 11-1
Fermilab product numbers 1-10
Fermilab products 1-10
Fermilab standard mail address 11-2
Fermilab systems-related information 3-7
Fermilab URLs 1-10
fermimail 11-24
fermitpu 7-4, 10-1
pros and cons 10-2
setup 10-7
fg 5-16
fgrep D-2
file 3-5, 6-11, 6-14
format 6-14
File dump
ascii D-2
decimal D-2

hex D-2
octal D-2
File expansion 2-5
File overwrite protection 8-4
File permissions 6-16
change 6-17
File protection 6-16
File specifications 6-4
File systems 1-7, 2-5
AFS 2-5
distributed 2-5
NIS 4-2
primary 2-5
root 6-1
Standard UNIX 2-5
standard UNIX 4-2
File transfer 12-1
.rhosts file 12-3
from VMS cluster 12-4
FZ file 12-2
RZ file 12-2
File Transfer Protocol (**ftp**) 12-1
File type, determine 6-14
Filename expansion 5-2, 5-14
turn off (in csh) 6-5
Filename extensions 6-4
Filenames 6-1, 6-4, 6-7
extensions 6-4
using dash (-) in 6-4
Files 6-4
access permission 6-16
ascii dump D-2
assign D-2
audio 8-8
browse 6-8
change ownership D-3
command 4-1
compare D-2
compression 6-10
concatenate 6-7, D-2
convert and copy D-2
copy 6-8, D-2
copy a set of D-4
copy to remote node D-4
deassign D-2
decimal dump D-2
delete 6-10, D-2
determine file type 6-14
directory 6-4
display beginning 6-8
display contents 6-7, D-1
display end 6-8
display first lines D-2
display last lines D-2
display list D-1
dump 6-13
executable 5-7
expansion 6-10
find 6-11, D-2
find differences D-2
find those containing string D-2
FZ 12-2
hex dump D-2
hidden 6-4, 8-5
image 8-8
link 6-7, 6-9, D-2
list contents 6-8
list directory contents 6-6, 6-14
list in reverse date order 6-6
merge D-2
MIME types and subtypes 8-8
mode 6-7
move 6-9
octal dump D-2
ordinary 6-4
overwrite 6-4
overwriting 6-4
permissions D-2
print 7-1, D-2
protection D-2
reference 6-9
remove 6-10
remove link D-2
rename 6-9, D-2
rename a set of D-4
RZ 12-2
search contents for pattern 5-11
search for 6-11
searching 6-12
send G-2
set permission at file creation 6-18
size 6-7
sort D-2
sort lines in 5-13
tar 6-10
text, formatting D-2
transfer 12-1
update last modified date D-2
versions 6-4
video 8-8
Filters 5-10
awk 5-11
grep 5-11
less 3-2, 5-10
more 5-10
sort 5-13
find 6-11, 17-3, D-2
format and options 6-11
Find file D-2
Find file containing string D-2
finger 3-7, 4-1, D-3
.plan 3-7, 3-8
fixmailperms F-9
Flavor 1-8, 9-1, 9-5
basic 9-5
extended 9-5
FLINT 13-12
FLPHOST variable 7-1
flpk 7-1, D-3
flpq 7-1, D-3
FLPQUE variable 7-1
flpr 7-1, 7-5, 11-16, D-2
defaults 7-3
options 7-1
flpr.defaults file 7-3
fmh2pine 11-5
fMHvms2mh H-2
FNAL 11-1, 11-2
set forwarding 11-3
FNALU 1-8, 2-6, 17-1, 17-2

accounts 17-1
AFS 2-5, 17-2
batch system 17-3
FUE 17-2
hardware 17-2
LoadLeveler 17-3
nodes 17-2
password 2-7
products 17-2
submit batch job 17-4
tapes 17-4
Folder
create G-1
delete G-1
remove G-1
folder 11-18, 11-22, G-1
Font size
exmh 11-10
Foreground jobs 5-15
Fork a process 5-1
Format
command 5-3
Format text file D-2
Formatting
man pages 3-2
FORTRAN 6-3, 13-1, 13-2, 13-4, 13-14
adding library symbol tables 13-14
AIX source code notes 13-11
AIX XLF 2 13-13
AIX XLF 3 13-13
arguments 13-18
asa 13-12
automatic variables 13-10
BLOCKDATA 13-14
BLOCKDATA modules 13-11
cache usage 13-16
calling C routines 13-18
carriage control characters 13-12
character special files 13-15
command line arguments 13-12
COMMON 13-18
compile D-4
compiler options 13-6
compiling files separately 13-11
conditional compilation 13-10
converting output to ASCII text 13-12
copying value of variables 13-12
cpp 13-10
DAFT 13-15
dbx 13-19
debugging 13-8, 13-19
environment variables 13-12
executing a program 13-18
external names 13-17
external references 13-11
extracting programs 13-11
feedback 13-16
file extensions 13-4
FLINT 13-12
floating point errors 13-15
force loading of a library routine 13-14
formatted records 13-14
fsplit 13-11
GETENV 13-12
I/O 13-14, 13-18
include files 13-10
INCLUDE statement 13-10
indexes 13-17
initializing variables in COMMON 13-11
inlining 13-16
interfacing with C 13-16
library references 13-5
link D-4
linking 13-18
list active compiler options 13-5
load map 13-9
macro expansion 13-10
option passing 13-5
passing options to next stage 13-5
printing 13-12
program execution 13-18
RBIO 13-15
record handling 13-14
retaining local variables 13-10
SAVE statements 13-10
source code analyzer 13-12
source code listing 13-9
source inclusion 13-10
speed optimization 13-8, 13-15
standard input and output 13-15
static variables 13-10
tape access 13-15
tape files 13-15
unformatted records 13-14
variables 13-17
word length 13-16
xlf 13-14
FORTRAN 90 13-13
FORTRAN compilers 13-4
FORTRAN language 13-1
FORTRAN, execute program D-4
FORTRAN-C
linking 13-18
mixed I/O 13-18
forw 11-16, F-3, G-1
Forward a message 11-15, G-1
Forward slash character 2-4
Forwarded mail header format 11-21
Forwarding mail 11-2
forwcomps file 11-15, 11-21, 11-23, F-5
fs listacl 6-19, D-3, F-9
fs setacl 6-19, D-3, F-9
fsplit 13-11
ftp 3-4, 3-5, 12-1, D-4, H-1
command list 12-1
ntuples 12-2
sample session 12-2
ftp, anonymous 9-5
FUE 1-9, 2-2, 8-5, 9-1
default login files E-1
FUE conformance levels B-1
FUE plus B-1
Full FUE B-1
funame 8-3, 8-8
fvms2mh A-1, A-2, H-2
FZ files 12-2

G

g++ 13-2
gawk 5-11
gcc 13-2
Getting help D-3
GhostScript 8-9
GhostView 8-9
glimpse 11-20
Gnu C 13-2
GNU make 14-7
gopher 3-5
grep 2-5, 4-2, 5-10, 5-11, 5-14, 6-11, 6-12, D-2
    format and options 6-12
groff 3-2, D-2
Group permission 6-17
Grouping commands 5-4
gtools 14-7
gunzip 6-10
gzip 6-10

H

Hard link 6-10
hash 4-5, 5-7
Hash table 4-5
head 6-7, 6-8, D-2
    format and options 6-8
Help on filenames 3-3
Help on shell commands 5-2
Help, online D-3
Hidden files 6-4, 8-5
Hierarchical Storage Management 17-5
history 5-5
History list 4-3
history variable 8-6
Home directory 6-1, 6-2, 6-3, 6-15, 13-14
    ~ 8-3
    HOME variable 8-3
Home page 3-5
HOME variable 6-3, 8-3
home variable 8-2
Host name 3-4
HSM 17-5
    NSL Unitree 17-5
    tcache 17-6
HTML files 3-4, 3-6
http 3-5
HyperText Markup Language 3-4

I

I/O 5-9, 13-18
    C 13-18
    FORTRAN 13-18
I/O redirection 5-2, 5-7, 5-8
    failure 5-8, 5-9
    grouping 5-8, 5-9
    input 5-8
    suppress output 5-8
I/O,buffered 16-7
Identify system 8-3
ignoreeof 2-2, 2-3
inc 11-13, G-1
inc-form file 11-21, F-5
Incorporate mail messages 11-13, 11-22, G-1
Info 3-7
info 3-4
Information distribution system 2-6
Inode number 6-4
Inode table 6-4
Input redirection 5-7, 5-8
Input/Output redirection 2-5
insight 3-4, 13-21
Instance 9-1, 9-5
Interactive command entry 5-1
Interfacing C and FORTRAN 13-16
Internet 3-4
    domain name 3-4
    ftp 3-4
    host name 3-4
    IP address 3-4
    mail 3-4
    navigation tools 3-4
    news 3-4
    services 3-4
    telnet 3-4
    WWW 3-4
Interpretation of commands by shell 5-2
Interpretive language
    awk 5-11
Interpretive programming language 4-1, 4-4
Interrupt process 2-3
IP address 3-4
IRIX 1-9
ISO standards 13-2

J

Jobs
    background 5-15
    foreground 5-15
    kill 5-17
    list 5-16
    move to background 5-17
    move to foreground 5-16
    priority 5-15
    scheduling 5-17, 5-18
    start in background 5-15
    stop 5-16
    stop 5-17
    suspend 5-16, 5-17
    terminate 5-17
jobs 5-16
jobstat 17-6

K

Kerberos authentication D-3
Kerberos password 17-5
Kerberos token 2-7, 17-2
    expiration 2-7
Kernel 1-7, 4-1

Keys, special 2-3
kill 5-16, 5-17, D-3, D-4
Kill batch job D-4
Kill process D-3
klog 2-7, 17-2, D-3
knews 3-7
kpasswd 2-6, 2-7, D-3
kpp 15-2, 15-3
    text preprocessor 15-2
ksh 4-3

L

LAN 2-5
Language support environments B-1
Languages 13-1
ld 13-4
less 3-2, 5-10, 6-7, 6-14, D-1
    /pattern 5-11
    format 6-8
lex 13-3
Library
    create D-4
    manipulate D-4
Library entry points D-4
Link directories 6-9
Link editor 13-4
Link files 6-9, D-2
Link FORTRAN program D-4
Links 6-7, 6-9
    hard 6-10
    symbolic 6-9, 6-10
List ACLs D-3
List files D-1
List jobs 5-16
List subdirectories D-1
Literal interpretation of character 2-4
Literal string interpretation 2-4
ln 6-9, D-2
    format and options 6-9
Load map 13-9
LoadLeveler 17-3, 17-4
    fbatch 17-3, 17-5
LoadLeveler D-4
Local area network 2-5
Log in 2-1, 8-1
Log in remotely 12-4
Log out 2-2, D-3
    ctrl-d 2-2, 2-3
    exit 2-2
    terminate processes 2-2
logdir 6-2, 13-14
Logical devices 5-7
Login directory 6-1, 6-15
Login files 8-1, 8-5, E-1
Login id 2-1
Login name 2-1
Login scripts 1-9, 2-2
Login to remote systems 12-3, 12-4
login.com (VMS) A-2
logout 2-2, D-3
Looping commands 5-4
lp 7-5
lpq D-3
lpr 7-1, 7-5, D-2
ls 3-1, 5-8, 5-9, 6-4, 6-5, 6-6, 6-14, 6-16, 11-22, D-1
    format and options 6-6
lynx 3-5

M

m 5-5
Machine identification 8-3
Macro 14-2
    format in Makefile 14-2
Magic file 6-14
Magic numbers 6-14
Mail 3-4
    alias file 11-21, 11-23
    Berkeley 11-24
    choose editor F-3
    choose node to read mail 11-1
    configuration files 11-20, F-3
    define basic configuration 11-20
    define forwarding header format 11-21
    define message header format 11-21
    define reply header format 11-21
    delete a message 11-17
    delivery customization 11-21
    distribution lists 11-21, 11-23
    extract a message 11-17
    folders 11-5, 11-18, 11-21
    forwarding 11-2
    forwarding at Fermilab 11-1
    incorporate format 11-21
    incorporate messages 11-13
    multilevel folders 11-22
    refile messages 11-19
    remove a message 11-17
    reply header, customize 11-14
    scan format 11-21, F-5
    search for a message 11-20
    search for messages 11-19
    signature lines 11-23
    signature lines for forwarding 11-23
    signature lines for replies 11-23
    standard Fermilab address 11-1
    text search
        glimpse 11-20
mail 11-24
Mail conversion 11-1, H-1
    automatic process H-2
    disk space required H-2
    fMHvms2mh H-2
    fvms2mh H-2
    MH folders H-2
    semi-automatic process H-2
    VMS scratch space H-2
Mail folders 11-5, 11-18, 11-21
    change 11-18
    create 11-18
    MH vs **pine** 11-4
    multilevel 11-22
    nested 11-22
    remove 11-18
    renumber messages 11-22

reorder messages 11-22
Mail forwarding 11-2, 11-8
choose mail node 11-8
file-sharing systems 11-2
set on chosen mail node 11-2
set on external nodes 11-3
set on FNAL 11-3
Mail holding area 11-22
Mail notification F-2
customize output F-3
default format F-2
double F-2
Mail readers 1-8
Mail systems 1-8
exmh 11-1
MH 11-1
mh 11-1
pine 11-1
MAIL variable 11-22
Mailbox directory 11-22
mailto 3-5
mailtools A-2, F-9
Maintain customized code 8-8
make 14-1
$MAKE macro 14-7
action 14-8
Bourne shell 14-3
built-in rules 14-8
command syntax 14-5
defaults 14-8
environment variables 14-7
GNU make 14-7
list built-in macros and rules 14-8
macro 14-2
Makefile 14-2
options 14-5
remove stray files 14-6
SHELL variable 14-3
suffix definition 14-2
suffix rule 14-2
target 14-2
use without Makefile 14-6
Makefile 14-1, 14-2, 15-1
automatic generation by **uvmbld** 15-2
blank lines in 14-2
commands and shells 14-8
definition types 14-2
environment variables 14-8
file modification times 14-4
first target (default) 14-6
housekeeping target 14-6
implied target 14-4
including control files 14-5
macro 14-2
macro format 14-2
macro sources 14-3
portability 14-7
preprocessors 14-7
remove stray files 14-6
required files 14-3
shell commands in 14-2
shell variables 14-8
special macros 14-3
SUFFIX declaration 14-5
suffix definition 14-2, 14-5
suffix rule 14-2, 14-4
tabs in 14-4
tabs vs. blanks 14-4
target 14-2
target "all" 14-6
target definition 14-3, 14-4
format 14-3
platform variance 14-8
successive commands 14-8
target usage 14-4
use of line continuation character 14-4
variables 14-2
man 3-1, D-3
Man page directories 8-4
man page formatting 3-2
man pages 3-1
built-in commands 5-2
exit 3-2
filename option 3-3
keyword option 3-3
quit 3-2
search for pattern 3-2
shell commands 5-2
MANPATH variable 8-4, 8-7
Match characters in regular expressions 5-14
Match single character 6-4
Memory corruption 13-20
Memory leaks 13-20
merge D-2
Merge files D-2
mesg n D-4
Message
compose G-1
delete G-2
display G-2
display headers G-2
display next G-1
display previous G-1
forward G-1
incorporate G-1
print G-2
refile to different folder G-1
remove G-2
reply G-1
reply and include original text G-1
select by criteria G-1
send G-1
Message header format 11-21
Messages
disable D-4
Metacharacters 2-4, 2-5, 5-7, 5-13, 6-4
MH 11-1
command line implementation 11-4
configuration files 11-20
convert mail to 11-1
exmh 11-4
folders 11-5, 11-21
graphical interface 11-4
mh 11-4
mh 11-1, 11-4
.maildelivery 11-21
.mh_profile 11-13
alias file 11-21, 11-23
change current folder 11-18
change editor 11-13

choose editor 11-12, F-3
command reference 11-1
comp 11-12
compose a message 11-10, 11-12
configuration files 11-20
create a folder 11-18
customize 11-1
customize message headers F-3
define basic configuration 11-20
define forwarding header format 11-21
define message header format 11-21
define reply header format 11-21
display a message 11-14
display next message 11-14
display previous message 11-14
distribution lists 11-21, 11-23
extract a message 11-17
forward a message 11-16
inc-form 11-21
include a file 11-13
incorporate messages 11-13
pick 11-20
print a message 11-16
quit 11-13
refile messages 11-19
remove folder 11-18
remove messages 11-18
reply to a message 11-15
save a message but don't send 11-13
scan message headers 11-14
scan-form 11-21
send 11-13
send a file 11-13
send a message 11-13
sequence of picked messages 11-20
set up and invoke 11-10
use UNIX shell features with 11-4
use with UNIX shell commands 11-14
vi as default editor 11-12
mh commands G-1
MIME 8-8, 8-9, 11-4
mime.types file 3-5, 8-9
Minimal FUE B-1
mkdir 6-15, A-1, D-1
Mode 6-7
Mode bits 6-16
Monitor batch queue D-4
more 5-10, 6-7, 6-8, 6-14, D-1
format 6-8
Mosaic 3-5
mosaic 8-8
Mount point 2-5
Mount tape
request 16-4
Mount tapes 16-1, 16-7
Move a directory 6-16
Move a file 6-9
Multimedia support 8-8
MultiNet 3-8, 12-3, 12-4, 12-5
Multiple commands on a line 5-4
Multipurpose Internet Mail Extensions 8-8
mv 6-9, 6-16, 8-4, D-2
format and options 6-9
mvdir 6-16
format 6-16

N

Navigating to WWW documents 1-10
nawk 5-11
ndflist 17-6
NEdit 7-4, 10-1
commands 10-6
pros and cons 10-2
needfile 17-6
Nested mail folders 11-22
NetNews 3-6
Netscape 3-5
Network Information System 2-6
News 3-4
News 3-7
news 3-5
Newsgroups 3-6
Newsreaders 3-6
next 11-14, G-1
NFS 2-6
mail forwarding 11-2
nice 5-15
NIS 2-6, D-4
default shell 4-2
NIS password file 2-7
nn 3-7
noclobber variable 5-8, 5-9, 8-4
noglob variable 6-5
Non-printable characters, in file dump 6-14
Notational conventions 1-2
Notification of incoming mail F-2
nroff 3-2, D-2
NSL Unitree 17-5
nu/TPU 7-4, 10-1
pros and cons 10-2
setup 10-6
Null device 5-8, 5-9

O

oawk 5-11
obtain printcap 7-5
OCS 16-1, 17-4
X interface 16-6
ocs_allocate 16-3
ocs_broken 16-2
ocs_cancel 16-5
ocs_check_tape 16-4
ocs_clean_it 16-2
ocs_clean_list 16-2
ocs_deallocate 16-5
ocs_devfile 16-5
ocs_devstat 16-2
ocs_dismount 16-5
ocs_init_stat 16-2
ocs_message 16-2
ocs_mrlog 16-2
ocs_pending 16-2, 16-4
ocs_report_stat 16-2
ocs_request 16-4
ocs_setdev 16-4
ocs_stats 16-2

ocs_tape 16-2, 16-3
od 6-11, 6-13, D-2
    format and options 6-13
odump -c D-4
Online help D-3
    apropos 3-3
    man pages 3-1
    UNIXHelp 3-6
    vendor-provided utilities 3-4
Operating System flavor 1-8
Operator Communications Software 17-4
Operator Communications Software (OCS) 16-1
Option passing 13-5
Options 5-4
    grouping 5-4
    separator 5-4
Oracle product database 9-2
Ordinary files 6-4
OSF1 1-9
Other permission 6-17
Output
    display screen by screen 5-10
Output redirection 5-7, 8-4
    background jobs 5-15
    force overwrite 5-8
    overwrite existing file 5-8
Output, suppress 5-8, 5-9
Overwrite file protection 8-4
Overwrite files 6-4
Owner permission 6-17
Ownership of file D-3

P

p 5-5
pack 6-10
Parent directory 6-4, 6-15
Parent process 5-1, 8-1
Parentheses in commands 5-2
passwd 2-6, 2-7, D-3
Password 2-1, 2-6
    AFS 17-5
    AFS and standard UNIX 2-7
    change 2-6, D-3
    FNALU 2-7
    Kerberos 2-7, 17-5
    multiple 2-7
Password entry file 4-2
Password file, NIS 2-7
Password, AFS 2-7
Password, changing 2-6
Path 5-7
Path name separator 2-4
PATH variable 4-5, 5-7, 8-3, 8-7
    include dot **.** 8-3
path variable 5-7, 8-2
Pathname 6-1
    absolute 6-1
    relative 6-1
Pattern matching 5-13
Pattern, search for 5-11
Pattern-matching characters 2-5
perl 13-1, 13-3
Permissions 6-7, 6-16, D-2
    AFS 6-16, 6-18
    change 6-17
    determine current settings 6-16
    directory 6-19
    file access 6-16
    group 6-17
    other 6-17
    owner 6-17
    set when file is created 6-18
Personnel directories 3-7
Phone user D-4
pick 11-20, G-1
pine 8-8, 11-1, A-2
    configuration 11-5
    folders 11-5, 11-6
    printing configuration 11-6
    suggested options 11-6
Pipe output of commands on a line 2-5
Pipeline 5-10
Pipes 5-9, 5-10
Platform
    operating system flavor 9-1
Pointers 13-18
POSIX standards 13-2
Postscript files
    printing duplex 7-4
pr 7-3
Pre-print options 7-3
    a2ps 7-3
    pr 7-3
    psnup 7-3
Preprocessor
    cpp 15-3
    kpp 15-3
prev 11-14, G-1
Prevent interpretation of special characters 5-2
Preview command 5-5
Print a message 11-16
Print entry
    delete D-3
    display D-3
Print file D-2
Print message G-2
Print queue
    display D-3
printcap file 7-5
printenv 8-2
printf 8-9
Printing 7-1
    ascii to postscript 7-3
    both sides of paper 7-4
    check queue 7-1
    duplex mode 7-4
    flpr 7-1
    kill job 7-1
    postscript files in duplex mode 7-4
    text files in duplex mode 7-4
Priority 5-15
Process 5-14
    background 5-1, 5-15
    child 5-1, 8-1
    display D-3
    display status 5-1
    fork 5-1

interrupt 2-3
kill D-3
move to background 2-3
parent 5-1, 8-1
stop D-3
subprocess 5-1
suspend 2-3
Product databases 9-1, 9-2
Product distribution 1-9, 9-1, 9-4
anonymous **ftp** 9-5
upd 9-4
Product distribution services, register for 9-1
Product instance 9-1
Product numbers, Fermilab 1-10
Products 9-2
Products, Fermilab 1-10
Program execution 13-18, D-4
Program, debug D-4
Programming languages 13-1
Programs 5-1
Prompt
default 4-2
FNALU 4-2
shell 4-2
Protection 6-16, D-2
ps 4-1, 5-1, 5-16, 6-13, D-3
psnup 7-3
Public domain shells 4-3
purify 13-20
pwd 5-2, 5-8, 5-9, 6-14, D-1

Q

qdel D-4
qstat D-4
qsub D-4
quit 11-13, G-1
quota -v D-3, H-1
Quotes
back 2-4
double 2-4
single 2-4
Quoting character (backslash) 2-4, 5-4

R

RBIO 13-14, 13-15, 16-7
r-commands 12-4
rcp 12-2, A-1, D-4, H-1
.rhosts file 12-3
non-interactive access 12-3
options 12-3
proxy login 12-3
VMS system 12-3
RCS 15-1
rcvtty F-2
customize output F-3
scan_form file F-5
Read permission 6-7, 6-10, 6-16
directory 6-19
file 6-16
Recall commands 5-4
Redirect output 5-4
to file and to a command 5-10
Redirection metacharacters 5-7
Redirection of I/O 5-2, 5-7, 5-8, 8-4
failure 5-9
force overwrite 5-8
grouping 5-8, 5-9
overwrite existing file 5-8
suppress error 5-8
suppress output 5-9
Reexecution commands 5-5
preview 5-5
substitute 5-5
Reference directories 6-9
Reference files 6-9
refile 11-18, 11-19, G-1
Refile mail messages 11-19, G-1
Registering for product distribution services 9-1, 9-6
Regular expressions 2-5, 5-2, 5-13, 6-13
rehash 4-5, 5-7
Relative pathname 6-1
Remote command execution 12-5
Remote connections 12-1
Remote copy 12-2
Remote login 12-3, 12-4
rlogin 12-5
telnet 12-4
Remote node, connect D-4
Remove a directory 6-10, 6-16, D-2
Remove a file 6-10, D-2
Remove a mail message 11-17, G-2
Remove file link D-2
Remove mail folders 11-18
Remove **mh** folder G-1
Remove print entry D-3
Rename a file 6-9, D-2
Rename set of files D-4
Renumber messages in folder 11-22
Reorder messages in folder 11-22
repl 11-15, F-3, F-4, G-1
repl_inc 11-15, G-1
Replace variable name with value 5-2
replcomps file 11-14, 11-21, F-4
customize F-5
Reply header format 11-21
Reply to a message 11-14, G-1
Reply to message G-1
Request tape mount 16-4
Required file 14-1
Resources, computing 17-1
Restart display 2-3
Revision Control System 15-1
RISC-based UNIX systems 13-2
rlogin 2-1, 12-3, 12-5, D-4
.rhosts file 12-5
from VMS system 12-5
password 12-5
rm 6-10, 6-16, D-2
format and options 6-10
rmdir 6-16, D-2
rmf 11-18, G-1
rmm 11-18, G-2
rn 3-7
Root 6-1
Root directory 2-4, 2-5

rsh 12-3, 12-5, 17-3, H-1
password 12-3
shell script usage 12-5
Run executables from current directory 8-3
RZ files 12-2

S

s 5-5
savehist 5-5
savehist variable 8-6
scan 11-14, 11-18, 11-20, G-2
scan-form file 11-21, F-3, F-5
example F-5, F-6
Scheduling jobs 5-17, 5-18
Script execution 4-5
Script, shell 4-4
Scripted command entry 5-1
Scripting language
Tcl/Tk 13-3
Scripts
as job 5-14
default shell 4-4
execution 4-5, D-4
shell 4-1
source 4-5
Search engines 3-6
Search for files 6-11
Search for mail messages 11-19
Search for pattern 5-11, 6-12
Search on text patterns 5-13
Search on users 3-7
Search path 5-7
add command 5-7
sed 5-14, 13-3, C-1, D-5
sedit 11-11
line breaks 11-11
set preferences for 11-11
send 11-13, G-1, G-2
Send a mail message 11-10, G-1
Send file G-2
Separate commands on a line 2-5
Separator, path name 2-4
set 5-1, 8-1, 8-2, D-3
Set ACLs D-3
set correct 5-6
set prompt 2-3
setenv 7-2, 8-2, D-2, D-3
setpath.csh 8-3, E-3
setpath.sh 8-3, E-16
Setting default home page 3-6
setup 9-2
setups.csh E-4
setups.sh E-21
sh 4-3, 6-2
Shell 1-7, 1-8
default 8-4
Shell choice 4-3
Shell command 5-1
Shell command help 5-2
Shell commands, platform-specific 5-2
Shell comparison 4-3
Shell features 4-3
Shell functions 8-5
Shell program 4-2
Shell prompt 2-3, 4-2
Shell script execution 4-5
Shell scripts 4-1, 4-4, 5-1
affect current shell 4-5
default shell 4-4
execution 4-5, D-4
source 4-5
Shell support policy 4-4
SHELL variable 8-4
Shell variables 4-3, 8-1
PS1 variable 2-3
Shell, current 4-1
Shell, default (AFS,NIS) 4-2
Shells 4-1
bash 4-3
Bourne family 4-1
C family 4-1
command interpretation 5-2
completion mechanism 4-3
csh 4-3
exit 4-2
features 4-3
finding commands 8-3
interpretive programming language 4-4
Korn 4-1
ksh 4-3
public domain 4-3
sh 4-3
start 4-2
supported 4-4
tcsh 4-3
vendor 4-3
zsh 4-3
shells 13-3
show 11-14, 11-16, 11-17, 11-18, G-2
Signature lines for forwarding 11-23
Signature lines for mail 11-23
Signature lines for replies 11-23
Simple Editor 11-11
Single back quotes 2-4
Single quotes 2-4, 5-2, 6-13
Size of file 6-7
software
FNALU 17-2
sort 5-10, 5-13, D-2
Sort files D-2
Sort lines in files 5-13
sortm 11-22
source 4-5, 5-1, 8-7, 8-8
Special characters 2-4, 2-5, 6-4, 6-13
extended set 5-14
prevent interpretation 5-2
Special keys 2-3
Special symbols 5-13
Specify terminal type 8-9
Standard error 5-7, 5-8
Standard input 5-7, 5-8
get from stdout of previous command 5-9
Standard output 5-7, 5-8
connect to stdin of next command 5-9
Start a shell 4-2
Start display 2-3
start key 2-3

stderr 5-7, 13-15
stdin 5-7, 13-15
stdout 5-7, 13-15, 13-18
stop 5-16, 5-17
Stop batch job D-4
Stop process D-3
Stop terminal output 2-3
Stopped jobs 2-2, 5-16
Storing temporary files 6-19
String, literal interpretation 2-4
stty 2-1, 2-4, 5-16, 8-9, 8-10, D-4
    ^- 8-10
    ^> 8-10
    tostop 5-16
Submit batch job 17-4, D-4
Subprocess 5-1
Subshell 4-2
Substitute output for string 2-4
Substitute string in command 5-5
Suffix definition 14-2
Suffix rule 14-2
SunOS 1-9
Support policy 4-4
Supported operating systems 1-9
Supported platforms 1-9
Supported products and applications 9-1
Supported shells 4-4
Suspend process 2-3
Suspended jobs 5-16
Symbolic link 6-9, 6-10
System calls 5-1
System identification 8-3
Systems updates 3-7

T

Tabs 2-4
tabs 2-4
tail 6-7, 6-8, D-2
    format and options 6-8
talk D-4
Talk to user D-4
tape access 13-15
Tape archiver D-3
Tape drive
    allocate 16-1, 16-3
    cancel mount request 16-5
    deallocate 16-5
    dismount 16-5
    list 16-3
    manipulate 16-6
    mount request 16-7
    mount tapes 16-1
    pending mount 16-4
    request mount 16-4
    rewind tape 16-5
    set device characteristics 16-4
    unload tape 16-5
    use statistics 16-1
    verify mount 16-4
Tape mount 16-1, 16-7
    request 16-4
Tapes
    ANSI labeled 16-7
    archive D-3
    OCS 17-4
    operator assisted mounts 16-1
    read/write DART-formatted records 16-7
    VMS labeled 16-7
tar 6-11, D-3
Tar files 6-10, 6-11, 9-1, 9-4, 9-5
Target 14-1, 14-2
tcache 17-6
Tcl/Tk 13-1, 13-3
tcsh 4-3
tee 5-10
telnet 2-1, 3-4, 12-4, 12-5, D-4
    password 12-4
Temporary disk 6-19
term variable 8-2
Terminal control functions 2-3
    set 8-9
Terminal output, stop 2-3
Terminal settings
    change D-4
    display 2-4
Terminal type 2-2
    specify 8-9
Terminate batch job D-4
Terminate process D-3
Terminate processes at logout 2-2
Text file
    format D-2
Text patterns
    search on 5-13
Text preprocessor
    kpp 15-2
Tilde 6-2, 8-3, 13-14
Time D-3
TMPDIR variable 6-19
Token
    expiration 2-7
Token, Kerberos 2-7
tostop 5-16
touch D-2
Transfer files 12-1
    .rhosts file 12-3
    from VMS cluster 12-4
trn 3-7
troff D-2
tty F-2
Type ahead 5-4

U

UCM 15-1, 15-2
    CHECKIN 15-3
    element 15-2
    group 15-2
    kpp 15-2, 15-3
    library 15-2
    obtain manual 15-3
    sequential development 15-2
    transaction log 15-2
    uvma 15-2
    uvmbld 15-2

uvmi 15-2
umask 6-18
Unattended automatic mail incorporation
.maildelivery file F-6
AFS file permissions F-9
customize F-6
fixmailperms F-9
Universal Resource Locator 3-5
UNIX cluster 2-6
UNIX Code Management 15-1
UNIX commands
VMS equivalents D-1
UNIX flavors 1-8
UNIX flavors supported 1-9
UNIX password 2-6
UNIX password, standard 2-6
UNIX process, environment 8-1
UNIX Product Distribution 1-9, 9-4
UNIX Product Distribution platform 9-1
UNIX Product Support 1-9, 9-1
UNIX prompt 2-3, 4-2
UNIX Reference Desk 3-6
UNIX Resources WWW page 1-10
UNIX shell 8-1
UNIX support 1-9
UNIX system mailbox 11-22
UNIX system, log in 2-1
UNIXHelp 3-6
unlog D-3
unset 8-1, 8-2
unsetup ups product 9-3
upd 1-9, 9-4
Update file's last modified date D-2
Updating files 14-1
ups 1-9
ups list 9-3
ups product directories 9-2
ups product versions 9-3
ups products 9-1, 9-2
man pages 3-1
URL 3-5
Fermilab pages 1-10
Usenet News 3-6
User id 2-1
user variable 8-2
Users
display D-3
phone D-4
talk D-4
Users, information on 3-7, 3-8
Using **ups** products 9-2
uvma 15-2
uvma create element 15-3
uvma create library 15-3
uvma fetch 15-3
uvma group 15-3
uvma replace 15-3
uvma reserve 15-3
uvma tag 15-3
uvmbld 15-2
CHECKIN 15-3
uvmi 15-2
information commands 15-3

V

Variable name replacement 5-2
Variable types 13-17
Variables
Bourne shell family 8-2
C shell family 8-1
define in Bourne shell 8-2
delete 8-1
display value 8-1
environment 8-1, 8-2
shell 4-3, 8-1
unset 8-1
Vendor shells 4-3
Vendor-provided online help 3-4
Version 6-4
Version control system 15-1
Versions of files 6-4
vi 5-6, 5-14, 7-4, 10-1, 11-7, 11-15, 11-16
commands 10-3
pros and cons 10-2
VMS backup save-sets, unpack and convert D-5
VMS commands
UNIX equivalents D-1
VMS labeled tapes 16-7
VMS source code management
CMS 15-1
VMS tcp/ip, MultiNet 12-3
VMS, converting from H-1
vmsbackup D-5

W

wc 6-11, 6-13
Web Browsers 3-4
Web site 3-5
What Now? 11-12
whatis 3-3
where 13-19
which 6-3
White space 5-4
in command interpretation 5-2
who 3-8, 5-8, 5-9, 5-10, 6-13, D-3
who am i 3-8
Wildcard replacement 5-2
Wildcards 2-5, 5-13
Working directory 6-1
World Wide Web 3-4
Write permission 6-7, 6-10, 6-16
directory 6-19
file 6-16
Write to screen D-3
WWW 3-4
addresses 3-5
browser commands 3-7
browsers 3-4
directories 3-7
Fermilab at Work 3-6
Fermilab Home Page 3-6
Fermilab pages 1-10
file 3-5
ftp 3-5

gopher 3-5
HTML files 3-4, 3-6
http addresses 3-5
links 3-5, 3-6
mailto 3-5
news 3-5
newsgroups 3-6
product databases 9-1, 9-2
protocol 3-5
search engines 3-6
UNIX Reference Desk 3-6
UNIXHelp 3-6
URL 3-5, 3-6
www 3-5, 8-8
WWW Browsers 3-4
lynx 3-5
mosaic 3-5
netscape 3-5
www 3-5
WWW browsers 8-8
WWW directories 3-7

X

X display
DISPLAY variable 10-3
X windows applications
DISPLAY variable 8-3
X11 toolkit
Tcl/Tk 13-3
X-based browser commands 3-7
xfontsel 11-10
xlf 13-4, 13-14
xlsfonts 11-10
xocs 16-6
xrn 3-7

Y

yacc 13-3
Yellow pages 2-6
ypmatch 4-2
yppasswd 2-6, D-3

Z

zftp 12-2
zsh 4-3