CVS Policies for FMS

Amy Langenhorst <arl@gfdl.noaa.gov>

This document contains cvs rules and conventions for FMS [http://www.gfdl.noaa.gov/~fms] users and developers. For help in using CVS, a more detailed Guide to CVS [../cvsguide.html] is available. This document is probably only relevent to developers within GFDL.

Table of Contents

1. Introduction

CVS is the Concurrent Versions System, a set of tools for management of source codes with multiple users and developers distributed across a wide-area network. It is a central pillar of the open source community and is maintained by GNU. CVS features include:

- Single source repository. Different versions are stored as incremental differences from a base file.
- Access is controlled by unix file permissions on a per-directory basis. Access can be further refined by scripts that are triggered whenever files are checked in or out. Locking mechanisms are used to serialize simultaneous operations on the repository by multiple users.
- Users choose when to synchronize with the repository.
- Any prior state of the code can be recreated using the appropriate flags.

CVS documentation for FMS developers is available at the Guide to CVS [../cvsguide.html].

2. Location and Ownership of the CVS Repository

The root of the FMS repository is `/home/fms/cvs`. The user `fms` will retain prime ownership of the directory and especially the administrative files. The designated FMS code manager(s) will be the user(s) who will at any time have the `fms` password.

Each code-containing directory in the repository will belong to the `fms` user, but be writable by the `f` group.

The `fms` user will create directories as needed for an FMS component.

3. CVS Check-In Permissions

All FMS developers should be a member of the `f` group. Permissions in the repository allow anyone from the `f` group to check in new versions of any code to the CVS repository at any time, as long as they check in to a branch ending in `_$user`. Developers do not need to ask the `fms` user for permissions to check in a new version of a file.

4. Directory Structure

The main directory **$CVSROOT=/home/fms/cvs** contains directories:

- `CVSROOT/` (administrative files)
- `FMS/` (all code used in creating an executable which runs a model)
- `preprocessing/` (code for processing the input to model runs)
- `postprocessing/` (code for processing the output from model runs)
- `runscripts/` (scripts for running various models)
- `test/` (programs for testing various modules)
- `bin/` (scripts related to FMS)
- `data/` (placeholder location for external users to download input data)
- `.lockdir/` (world-writable directory for filelocking)

Directory FMS contains subdirectories:

- `coupler/`: main coupled model driver: contains file tree `main/coupler_main.f90`, `fluxes/flux_exchange.f90`, ...
- `shared/`: contains common utilities. Subdirectories `mpp/`, `exchange/`, `time_manager/`, `diagnostics/`, `advection/`, `tracers/`, ...
- `atmos/`, `ice/`, `land/`, `ocean/`: component model subdirectories

Each component model directory contains subdirectories `driver/`, `param/` and one for each dynamical core (e.g the directory `atmos` contains `bgrid/`, `spectral/`, ...

- The `driver/` directory, which may contain subdirectories `solo/` and `coupled/`. `solo/` contains a driver program to run this component by itself, e.g `atmos/driver/solo/atmos_model.f90`. `coupled/` contains a driver subroutine that acts as an interface between the dynamical cores and the main coupled model driver, e.g `atmos/driver/coupled/atmos_coupled.f90`.
- There is a directory for each dynamical core, e.g `bgrid/` and `spectral/` under `atmos/`, or `mom4/`, `mixed_layer/`, `amip/`, ... under `ocean/`.
- The `param/` directory contains all the parameterized physics. It need not exist if not appropriate for the com-

ponent model. Parameterization options for the same physical function are organized in a tree underneath this directory.

5. CVS Modules

CVS modules are organized in a 2-level hierarchy. The first level groups together files on the basis of affinity, but do not constitue a full executable set of code. The second level groups together modules from the first level to create models.

The first level controls individual directories:

- Individual dynamical cores each have a module named component_core: e.g `atmos_bgrid`, `ocean_mom3`, `ice_sis`...
- Individual param directories each have a module named *`component`*`_param`, e.g `atmos_param`, ...
- The module `shared` for the directory shared/.
- Individual directories under shared/ may have a module named subdir, e.g `mpp`, `exchange`, ...
- The module `coupled` for the directory coupled/.

On the second level, we have executable models:

- `fms_bgrid_solo`, `fms_bgrid_amip`, etc.

The modules file has also been set up with two additional features that is run for each checkout of a first-level module. One is a script called `list_paths` which creates path name lists of the checked out source and documentation files. The other is the use of the `-s` flag. The name of the principal developer is entered here. This is used by some scripts to identify the owner. Also users can type **cvs checkout -s** to get a table of available modules and their developers.

Note

Principal developer documentation is not implemented yet. Perhaps we could use this for module descriptions instead...?

Note

The modules file is controlled by the `fms` user alone. Changes to the module configuration can only be made by request to the `fms` user, and will require review by the MI team.

6. Release Schedule and Procedure

Developers are encouraged to get their code tagged with the `latest` tag as soon as possible regardless of the FMS release cycle. The longer you wait, the more merging will have to be done.

Modeling Services will propose a city release based on their estimation of code evolution since the last release, and the amount of branch code from multiple developers with the `latest` tag. An email will be sent announcing an FMS Developers Meeting about a week in advance.

The FMS Developers Meeting will be held about three weeks before the anticipated release date. Developers meet to describe, discuss, and ratify code changes for the release. This is the last chance to propose new code modifications to be included in the release.

The week after the meeting is dedicated to code merging and conflict resolution. At the end of this week, the `latest` tag should be on all code to be released.

When conflicts are resolved, Modeling Services performs regression testing on the code. The regression test suite must verify scientific correctness as well as continuity across restarts and PE counts before the release tag is applied.

When testing is complete, the code will be tagged and the release will be announced to all.

Thus a release schedule is of the form:

Table 1. Release Schedule

Time	Action
-4 weeks	Email scheduling an FMS Developers Meeting
-3 weeks	FMS Developers Meeting, code proposals finalized
-2 weeks	All conficts resolved, final testing begins
0	Release Date

7. Tags

7.1. City Release Tags

Trunk releases are tagged with city names, chosen in alphabetical order.

7.2. Branch Tags

CVS branches are used for FMS development. Branchnames are chosen by the developer, with the requirement that they end in `_$user`.

Branch Naming Conventions

How should you name your branch? Here are some guidelines you may want to keep in mind.

- If your changes all concern a specific theme and you think they will be incorporated in the next city release, use a name like *`theme_usr`*, ie, `parallel_arl`.
- If your changes concern a specific theme and you are not sure whether they will be incorporated in the release or not, or you would like to note which city release your changes correspond to, use a name like `city_`*`theme_usr`*, ie, `galway_parallel_arl`.
- If your changes concern multiple themes and you cannot put a specific label to them, you may use a name like `city_`*`usr`*, ie, `galway_arl`.

Note that all related changes that you make should go on the same branch. In other words, if a change you make in one file requires a change you've made in another file, be sure to put the same branch tag on both files.

7.3. The `latest` Tag

A tag called `latest` is used to keep track of the most recent well-tested lines of development which are intended for the next city release. At the time of a city release, the code tagged `latest` will be moved to the cvs trunk for the release. Developers need to make sure their code changes intended for the next city release have the `latest` tag.

A rule of thumb for when to have code tagged `latest`: Develop along your branch until you are ready for others within the lab to start using your code modifications. At that point, you should try to get your code tagged `latest`.

In order to have code included in the `latest` tag and the next city release, FMS developers should do the following.

1. Please try to start development from the `latest` version of the file you intend to modify. This will save time later. (ie, if a `havana` tag and a `latest` tag exist on different revisions of a file, please start making your modifications from the `latest` revision.)
2. Make sure your code has been checked into the repository on a branch with user initials.
3. Make sure your code is compatible with all of the other code which currently has the `latest` tag. If it is not, Modeling Services can help you merge your code with the `latest` code.
4. Developers are encouraged to run the FMS Specification Checker (**Controller.pm**) to identify certain conflicts with FMS specifications. Instructions are at the Spec Checker Page [http://www.gfdl.noaa.gov/~bnd/fms/spec_util.html].
5. Ensure the code is fully tested, meaning the code compiled, ran, and produced the answers you expected with the `latest` codeset.
6. Send an email to Amy.Langenhorst@noaa.gov [mailto:Amy.Langenhorst@noaa.gov] including the following information:
    a. the files affected
    b. the branch name
    c. a brief description of the change
    d. whether the new revision changes answers
    e. if answers change, whether they are machine-level (mathematically consistent changes like order of operations) or algorithm changes
7. if there are any further modifications to the code after it has been tagged `latest`, it is important that developers let the user `fms` know if the `latest` tag needs to be moved to the new version.

The user `fms` will do the following:

1. Move the `latest` tag to the new code.
2. Incorporate the relevent information onto the FMS `latest` webpage [http://www.gfdl.noaa.gov/~fms/latest.html].
3. Make sure that Modeling Services' regression tests incorporate the new `latest` code.

7.4. Additional Revision Tags

Descriptive revision tags such as `am2p9` may be applied to specific revisions of a set of files in the repository.

7.5. Intermediate Releases

Occasionally the need arises to move code revisions to the trunk between city releases. Bugfixes are an example.

7.5.1. Bugfix Releases

Bugfix releases are intermediate releases that occur when it is important to move crucial bugfixes to the trunk prior to the next city release.

8. External Access

CVS is set up to allow outside access to a repository, usually with restricted privileges, called pserver. The machine `cvs.gfdl.gov` now accepts incoming requests on port 2401. Incoming requests on that port from an external cvs client launches a cvs server. The file `$CVSROOT/CVSROOT/passwd` designates a generic user called cvs to be an external user and specifies a password. The file `$CVSROOT/CVSROOT/writers` designates external users who are allowed write access to the archive. Currently we have a nullfile of this name as an additional layer of protection preventing write access from outside.

The client follows this command sequence:

```
% setenv CVSROOT :pserver:cvs@cvs.gfdl.gov:/home/fms/cvs
% cvs login # you will be prompted for a password
% cvs checkout ...
```

A script has been written to automate this process. It will preserve your initial `CVSROOT`. Execute `/home/arl/bin/gfdlcvs` with no arguments to view a short help message.

9. Security issues

CVS, correctly used, has several levels of protection against inadvertent or malicious modification of the repository:

- New code-containing directories in the repository are created by the `fms` user with mode 775, so that they are writable by the `f` group. The `Attic` directory should be created at the same time with the same permissions.
- The `commitinfo` mechanism is used to control access to the trunk. At each cvs commit, the script `/home/fms/bin/check_ci_permissions` is called to test that developers are committing to a branch containing their user initials.
- The directory `$CVSROOT/CVSROOT` in particular must never be writable by anyone but the `fms` user. It is created with mode 755.
- Outside users can only read the CVS archive. The server permits no other action. There are no security holes when outside access is restricted to commands that cannot modify the archive.
- All checkout, commit, rtag, update and release commands are logged in the file

`$CVSROOT/CVSROOT/history`, which must be created by the `fms` user and be mode 666 (world-writable).

- The file `$CVSROOT/CVSROOT/val-tags` must exist and be mode 666 (world-writable).
- CVS provides the command **cvs admin** which provides access to the underlying RCS calls. This is strongly discouraged, since some RCS options are incompatible with CVS. We have created the unix group named `cvsadmin` with no members, effectively disabling this command.

Guide to CVS for FMS Users and Developers

See the Guide to CVS [../cvsguide.html].

City Release Procedure

See the Release Procedures page [http://cobweb.gfdl.noaa.gov/~arl/release_proc].

Warning

This is an internal-only page.