Subversion Code of Conduct

Our Subversion server is deprecated in favor of using Git and GitHub to store code going forward. This page remains here for legacy projects and historical accuracy.

See GitHub Conventions for the updated version

Why have this document?

Our Subversion repository (view) is wonderful in many ways. To keep it wonderful, committers should follow a few simple guidelines.

Compiling / Building

Code in the **trunk** should always *build* but does not always have to run smoothly.

The code in **other branches** does not have to compile or run. It is all up to the branch creator/maintainer(s).

The code in **module** branches does not have to compile or run. It is all up to the module source creator/maintainer(s).

The code in **contrib** branches does not have to compile or run. It is all up to the specific contributor(s).

The code in **custom-builds** is up to the specific contributor, but and is not for developing, see custom-builds

See also: Subversion Repository Layout

Commit Access

There are three types of commit access: *admin*, *full*, and *partial*. Developers without commit access can submit code through patches (via the developers list, attached to a ticket, or via a committer).

- **Admin Committers**
  - Able to commit to the /trunk/tags folder
  - Only used during a release cycle. The role of the admin(s) at this time is to tag the trunk code, generate the war file, update the install file, and post (release) the files.
  - The current admins are anyone who has served as a release manager: Ben Wolfe, Darius Jazayeri, and Wyclif Luyima
- **Full Committers** (mostly core devs)
  - May commit to any part of the subversion tree
  - May commit patches to trunk by outside developers
  - May create a branch for themselves or partial committers (see #Creating a branch for commit comment examples)
  - Can request a partial committer be promoted to a full committer.
- **Partial Committers** (any volunteer/student/intern or module writer)
  - Should commit only within their area(s) of expertise (their module folder or branch)
  - Are welcome to make straightforward, obvious fixes to the trunk.
  - May create a branch for themselves for a ticket or with approval from a full committer (see #Creating a branch for commit comment examples)

How to Become a Partial Committer

Anyone can become a partial committer. Email code at openmrs.org and request access for a module or branch. See committer subversion access for details on how to request repository access and what to include in your request.

How to Become a Full Committer

After contributing a few non-trivial patches or enhancements to the system, a partial committer can be nominated for full commit access by a full committer . Decisions on granting full commit access are based on the consensus of existing full committers. Email code at openmrs.org to request. As described by the Subversion group, *"the primary criterion for full commit access is good judgment."*

Committing to Trunk

Full committers can commit fixes and patches to trunk. A code review should be opened in crucible with at least (and probably at most) 2 devs on it: http://source.openmrs.org

Partial committers should not commit to trunk, but are allowed to commit **obvious and straightforward bug fixes, or minor self-contained enhancements to existing features** directly into trunk.

All other developers may submit bug-fix patches by attaching a patch file to a jira ticket. Any full committer may apply these patches to trunk. Full attribution should be given in the commit message (see commit comment conventions for formatting attribution below).

Any major feature additions or code overhauls should be submitted to a branch. Branching is quick, easy, and if done right, low-bandwidth – don't be afraid of it (see Subversion Branching and Merging Techniques).

New Features

All new features should be developed in a branch. This allows the developer to adhere to the policy of committing **early and often** much more easily than if the feature were developed directly on trunk (see Subversion Branching and Merging Techniques).

All developers are expected to watch source code that is committed (e.g., via rss) and give the one who is committing feedback. The committing developer *must* be open to suggestions and constructive criticism.

When a feature has been fully developed and tested, it can be merged back into trunk by a full committer. Merging "permission" should be acquired from the other full-access developers prior to the merge (see Subversion Branching and Merging Techniques).

We welcome outside developers to become partial access committers. For outside developers that would like to contribute something larger than a bug fix, a branch can be created and read/write access given to that developer. If you would like to have a branch created for you see Developers Guide

Modules

All module code will be placed under the /openmrs-modules folder in the repository. The name of the folder for the source code will be the name of the module id. The code stored in the repository must be open source, but does not have to be under the OPL. See also Module Licensing

Module developers are encouraged to read our Module Conventions page and follow Subversion's suggested directory layout:

```
/openmrs-modules/
    yourmoduleid/
        trunk/
        branches/
        tags/
```

The latest code goes under the trunk folder, the branches folder is used for trying out changes separate from trunk, and tags are copies of trunk at the time of a particular release. If you are not planning on a release cycle at first, we still encourage you to put your code into a "trunk" folder, since many modules eventually evolve to needing this structure. If you need to change to a branches/tags/trunk structure, see Module Branching.

Module authors are encouraged to mimic the OpenMRS versioning style: major.minor.maintenance – e.g., 1.5.2.

Modules that work with a specific version of openmrs are encouraged to use 1.5.x-compatible or 1.4.x-compatible as the branch name inside their module.

To request space for a new module see requesting a module/project of the Code Repository page.

Source code for forgotten, no longer supported, and/or abandoned modules may be moved to the /openmrs-modules/abandoned/ folder.

Custom Builds

A space for custom builds is provided within the repository to facilitate the process of combining specific releases with features from subsequent releases and/or unreleased patches. While applying patches to a specific release may often require some small tweaks to the code to accommodate interval changes, we do NOT want people doing development within these custom builds. All development should occur within patches or branches. Developers who run off and start using a custom build to commit all their new code risk being publicly castigated and/or losing their commit rights.

Each custom build should have a README file in the root folder describing what changes are in that build. This allows anyone else that might want to use those same features to quickly know how that build is compiled.

Each custom build is suggested to have a set of patches in the repository that have been applied. This helps when upgrading and is another tool that gives other developers insight into what that custom build contains.

All new code that is committed to custom-builds should get a ticket that either points at the changeset or gets the patch attached to it. This will help to make sure that a fix isn't going into a custom-build and then forgotten about. The rest of the community suffers when this happens.

Cleaning up a module folder, branch, or contrib

1. Review the modules (or write a script) that shows all modules not touched within the last two years.
2. Get list of modules downloaded or "checked for updates" in the module repo in the last 1 year.
3. Delete from the repo all modules that are in the first list but not in the second
4. Add a note to the wiki pages for each module (that we can find) that was deleted about its automatic pruning and where to find the last working svn code.

Commit Comments

This section outlines the different scenarios and sections in which commits occur. Find the one that best matches yours and follow that.

Note that code commit comments and ticket comments are linked in jira via key characters:

- Link to a specific revision with the 'rev' string: rev:12343
- Link to a specific ticket number with the ticket key and id: TRUNK-123 or ECLP-36

Commits to trunk

- Note that these should be smaller bug fixes. See above section for when to use branches.

  You do*not have to include the word "trunk" in your message:

- The ticket id is optional, as there could be no jira ticket associated with the fix.

```
Fixing feature XXX - TRUNK-123
```

where "Fixing feature XXX" is a descriptive sentence about the error/function being corrected and TRUNK-123 is the jira ticket id.

Commits to a branch

- You do not need to include branch name in the message.
- Be as descriptive as possible

```
Fixing bug that caused X if Y was done - TRUNK-123
```

Adding functionality X

Commits to a module

- You do not need to include the module id in the message
- Be as descriptive as possible

```
Adding functionality X - MODULE-123
```

Merges from trunk to a branch

- Aka: updating a branch to the latest trunk code
- Include the revision numbers that are the start and end of the merge. (The ending merge number is *very* important when re-merging later)

```
Merging trunk to dictionary_import rev:123 - rev:234
```

Where 123 and 234 are the revision start and end numbers for the merge.

Merges from a branch to trunk

```
Merging dictionary-import to trunk rev:123 - rev:234
```

Where 123 and 234 are the revision start and end numbers for that merge.

Applying patches to trunk and other branches

- You do not need to include the name of the branch
- Be sure to give attribution to the username of the person that wrote the patch

```
Adding feature XXX . see ticket TRUNK-123
Author: bwolfe, hhornblower
```

Where 123 is the ticket the patch came from and bwolfe is the author's openmrs/svn username.
Be sure to attribute the patch author(s) in the comment so we can track and properly attribute contributions.

To allow patch authors to be properly credited in an automated fashion, attribution should be on a separate line starting with "author:" followed by the OpenMRS ID of the contributor(s) separated by commas.

For information about submitting patches to OpenMRS, see our pages on patches.

Creating a branch

- Be as specific as possible in your initial commit comment. Commits creating new branches are sent to the developers mailing list.
- Be sure to include the branch name and if applicable, the username of its intended user
- For partial committers, include the username of the approving full committer
- For branches based on tickets, put ticket number in branch name

See requesting subversion repository access for how to request a new branch.

```
Creating dictionary-import branch to add web services throughout the api to allow for concepts and forms to be passed from a parent server to a child server.
```

Creating foo-bar branch to do XX features so that YY functionality. Approved by mseaton

```
Creating ticket974-liquibase branch to add liquibase into openmrs to replace the .sql files and the manually update-to-latest-mysqldiff.sql file.
```

Creating a module

- Be sure to include the module's id and the username of the intended user

  ```
  Creating module formentry for bwolfe
  ```