GitHub Conventions

- Why have this document?
- Compiling / Building
- Push/Pull/Write Access
    - Committing to openmrs-core
- New Features
- Modules
    - Best Practices for module development
    - Hosting a module under the OpenMRS organization
- README conventions for repositories in GitHub
- Commit Comments
    - Commits to a branch (same as trunk)
    - Commits to a module
    - Merges from a branch to openmrs-core
    - Applying patches to openmrs-core and other branches
    - Creating a module

Why have this document?

Our GitHub repository is wonderful in many ways. To keep it wonderful, committers should follow a few simple guidelines.

Compiling / Building

Code in the **openmrs-core** "master" branch should always *build* but does not always have to run smoothly.

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

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

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

Push/Pull/Write Access

Access to community repositories is based on Developer stages. Anyone, including /dev/null community members, may submit code through pull requests.

- **/dev/5 and maintainers of individual repositories**
    - May administrate repositories
- **/dev/4 and /dev/5**
    - Able to push to any repo in the OpenMRS org in GitHub
    - May commit patches or merge pull requests to openmrs-core by outside developers
- **/dev/3**
    - Able to push to any repo in the OpenMRS org in GitHub except for openmrs-core
    - Should commit only within their area(s) of expertise (their specific projects)

Committing to openmrs-core

/dev/4 and /dev/5 developers can push fixes to openmrs-core or merge pull requests. The associated ticket should be marked as "Committed Code" so as to be in the "Code Review (post commit)" state.

All others cannot push to openmrs-core, but are encouraged to open pull requests instead.

Any major feature additions or code overhauls should be submitted to a branch or fork. Forking and branching is quick and easy: don't be afraid of it.

New Features

All new features should be developed in a fork or 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 openmrs-core.

All developers are expected to watch source code that is committed (e.g., via the "watch" feature shown on the openmrs-core repo) 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 openmrs-core by a /dev/4 or /dev/5. Merging "permission" should be acquired from the other full-access developers prior to the merge (see Git Branching and Merging Techniques).

Modules

You can create a module at any time without your personal GitHub account. The name of the folder for the source code will be "openmrs-module-" + the module id.

Module developers are encouraged to read our Module Conventions page.

See also Module Licensing.

> When choosing a module ID, we recommend reaching out to the developer community or to code@openmrs.org with a brief description of your module and your proposed module ID. This will help avoid conflicts with other module authors, ensure that you find out about any related work going on, and ensure that you are following best practices with your module ID.

Best Practices for module development

- Latest code is always committed to master branch.
- Features are developed in branches. If using a JIRA project, the branch name should be the issue name (e.g., MYMOD-123).
- For major & minor releases, create a "major.minor.x" branch (e.g., branch named "1.5.x") branch to use for backporting future maintenance changes as you tag your release (e.g., tag named "1.5.0").
- Follow the OpenMRS versioning style: major.minor.maintenance[-modifier] – e.g., 1.5.2 – where minor releases are backwards compatible and maintenance releases are bug fixes.

Hosting a module under the OpenMRS organization

Community Modules can be hosted under the OpenMRS organization within GitHub (github.com/openmrs). They should meet the following requirements:

- **Community-developed or community-owned.** Serving the needs of more than one organization.
- **Designed to be used by multiple organizations.** Not designed for one specific location.
- **Open source.** Licensed under the OpenMRS Public License 1.0, MPL 2.0 + Health Disclaimer, or a OSI-approved license that is compatible with MPL 2.0 + Health Disclaimer.

To request that your module repository be migrated to the OpenMRS organization in GitHub, please ensure that it meets the above criteria, then:

Post to OpenMRS Talk > Repo Management and include:

1. Your GitHub username.
2. A link to your module repository.
3. A short (1-3 sentence) description of your module and why you believe it belongs under https://github.com/openmrs.
4. CC any /dev/4 or /dev/5 working with you on this effort (e.g., students should CC their mentor). You can CC people in OpenMRS Talk simply by adding "`/cc @username1, @username2, ...`" to the end of your post.

README conventions for repositories in GitHub

Each repository should contain a README.md in the root folder. The `.md` extension for a README refers to markdown syntax.

Your README should contain at least two sections: description and installation. For example:

```
== Description ==

This should be the description of your module. Write your description in a way that will help
someone who knows nothing about your module or your projects to understand the purpose of your
module and what feature(s) it provides. If there are additional resources/documentation that
describe your moule, then direct the reader to those resources in your description.
```

Commit Comments

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

Commits to openmrs-core

- Note that these should be smaller bug fixes. See above section for when to use branches.
- Every commit really SHOULD have an associated jira ticket with the fix.

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

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 (same as trunk)

- Be as descriptive as possible

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

Adding functionality X
```

Commits to a module

- Be as descriptive as possible

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

Merges from a branch to openmrs-core

```
Merging dictionary-import to which provides functionality X, Y, and Z
```

Applying patches to openmrs-core 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 in the "author" field in git

```
TRUNK-123: Adding feature XXX or fixing bug YYY
```

Where 123 is the ticket the patch came from.

Creating a module

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

  ```
  Creating module formentry for bwolfe
  ```