Using Git

- Install Git on your computer
- Learn how to use Git
- Get the code (with the intention of contributing changes)
    - Create account on github.com
    - Check out the code to your machine
- Get the code in ready-only mode (without intention of contributing)
- Work on your task
    - Submit the code
        - Synchronizing with EGit
    - Getting feedback on a pull request and making changes to it
    - After the ticket has been closed
- Creating your own module on github
- Naming Conventions
    - For modules
    - For scripts that package a distribution
    - For variations of the OpenMRS Standalone
    - For other contributions
- Why Github?
- Also see

Looking for a quick reference to git commands? Check out the Git for SVN Users page.

We switched OpenMRS core /trunk code from svn to git. As of August 2012 we are using github.com as the hosting provider for our git projects. See the openmrs github organization.

Git is the version control system which we use to collaborate on the OpenMRS code bases.

Install Git on your computer

To get the git binary, see OS specific sections on installing git page.

This gives you a git command line. Git at the command line is easier to use in some instances than IDE Integration. But if you want
to use your IDE to interact with git (ala subclipse in eclipse), then see Git IDE Integration.

If you are a Windows user, see how to setup your user and password for the git command line here.

Learn how to use Git

You will be using Git all the time when working on OpenMRS and its also very likely that you will use it on other projects. It is therefore very useful to understand git. Don't be afraid its totally ok to start with understanding it a little bit to get started, but try to invest some more time once in a while to dig deeper into its workings. It will pay off 🙂

Here are a few free resources we recommend

- https://try.github.io - short interactive intro course to git
- https://git-scm.com/book/en/v2 - very good book! You can start with the basics and work your way through with time. The Git internals is also super interesting!

Get the code (with the intention of contributing changes)

These instructions are designated for a developer who wants to contribute to projects under https://github.com/openmrs. A private fork will be used only as a backup storage and for pull requests, but not for collaboration with other developers. If you intend to use your fork to work in a team, please make necessary adjustments to the proposed workflow by avoiding the use of rebase.

Create account on github.com

https://github.com/signup/free

Check out the code to your machine

1. On github.com, fork a project you want to work on (see tutorial: http://help.github.com/fork-a-repo).
    - We will base this example on the "openmrs-core" project (the core OpenMRS application).
    - Go to https://github.com/openmrs/openmrs-core
    - Click "Fork" button in the upper right
2. Clone the fork to your local machine (replace "yourusername"):

```
git clone https://github.com/yourusername/openmrs-core.git
```

3. Go into the folder just created and set up the "upstream" remote so you can eventually pull changes from the main repository (see "Configure Remotes" on http://help.github.com/fork-a-repo):

```
git remote add upstream https://github.com/openmrs/openmrs-core.git
```

4. Fetch and track all branches on remote repositories.

```
git fetch --all
```

5. After you cloned the repository you can list available branches:

```
git branch -a
```

The currently checked out branch is highlighted with an asterisks. The master branch is the latest development branch (trunk was its counterpart in SVN).

6. Pull changes from the upstream remote:

```
git pull --rebase upstream master
```

The '--rebase' option reverts any of your commits which are not in upstream/master, then fast forwards your local branch to upstream/master and finally applies your commits on top of that. This allows you to avoid merge commits and keep the history linear, but must not be used if you want others to work with you on a branch in your fork.

Optional pro tip: To save some key hits we recommend to setup an alias for 'pull --rebase' by running

```
git config --global alias.up "pull --rebase"
```

From now on you will be able to use 'git up upstream master' instead of 'git pull --rebase upstream master'.

7. Push changes to your fork on github:

```
git push
```

This command pushes all your commits from all your tracked branches to your fork. If your fork has commits, which are not in your local repository, the branches containing these commits will not be pushed. You will need to pull on these branches first to have them pushed.

We do not advise you to work on master or maintenance branches of openmrs-core, but create topic branches instead. This way you will be able to send us pull requests to merge your code back to the main repository. See "Create Topic Branches" below, please take a look at our pull requests tips page.

See Maven wiki page for how to build, compile, etc.

Get the code in ready-only mode (without intention of contributing)

This will check out the code for "openmrs-core" (the core OpenMRS application):

```
git clone https://github.com/openmrs/openmrs-core.git
```

1. Checkout out a new local branch based on your master and update it to the latest. The convention is to name the branch after the current ticket e.g. TRUNK-123.

Make sure you have committed any new files that are not under version control otherwise you will lose them when you run *git clean -df*

```
git checkout -b TRUNK-123 master
git clean -df
git pull --rebase upstream master
```

2. Push the branch to your fork. Treat it as a backup.

```
git push origin TRUNK-123
```

> Pro tip! Git has tab completion here!

Work on your task

NOTE: If the ticket you are working on requires you to work off a development branch say 1.9.x, you will have to use the development branch name instead of **master** for the instructions in this section. E.g. instead of **git pull --rebase upstream master** you will run **git pull --rebase upstream 1.9.x**, where 1.9.x is the development branch you are coding off.

1. Make changes in code.
2. Add changes to a commit (stage them). To see what files have changed.

```
git status
```

To stage all changed and new files.

```
git add -A
```

To pick only some files.

```
git add -i
```

You will see a summary of changed and new files. You need to choose '2' to mark files as updated. Now you need to pick files, which you want to mark as updated. You can specify them as a list 1, 2, 3, as a range 1-3, or simply * to select all. Confirm by hitting the enter key twice. If there are new (untracked) files, choose '4' and pick files the same way. Choose '7' to quit.

3. Commit changes.

```
git commit -m "TRUNK-123: Put change summary here (can be a ticket title)"
```

Please remember to specify the current ticket id in your commit message.
Optionally run mvn install -DskipTests=true before your commits so that the formatting is fixed.

4. If you need to take a break from your work and continue later, you should backup your code by committing changes and pushing to your fork.

```
git push origin TRUNK-123
```

> If you absolutely need to and you know what you are doing, add the '-f' option to force the push, which means overwriting history if necessary. You must NEVER use this option with the main repository (upstream: https://github.com/openmrs (https://github.com/openmrs\!))!

5. Before you pick up your work again, update to the latest code.

```
git pull --rebase upstream master
```

If you have uncommitted changes and git refuses to merge, you can stash your changes, perform a pull and then unstash them.

```
git stash
git pull --rebase upstream master
git stash pop
```

If you have conflicts, you need to resolve them by editing conflicting files, adding them to index, committing and then pulling again. Alternately you could try with a different merge strategy.

```
git pull --rebase -X patience upstream master
```

If you want to nuke all your commits run 'git reset --hard upstream master'. If you want to nuke only last commit run 'git reset --hard HEAD~1'.

Submit the code

1. Update your branch to the latest code.

```
git pull --rebase upstream master
```

2. If you have made many commits, we ask you to squash them into atomic units of work. Most of tickets should have one commit only, especially bug fixes, which makes them easier to back port. To squash three commits into one, do the following:

```
git checkout master
git pull --rebase upstream master
git checkout TRUNK-123
git rebase -i HEAD~3
```

In the text editor that comes up, replace the words "pick" with "squash" or more preferably "f" next to the commits you want to squash into the commit before it. Save and close the editor, and git will combine the "squash"'ed commits with the one before it. Git will concatenate the commit messages if "pick" is replaced with "squash".

Helpful hint: You can always edit your last commit message, before pushing, by using:

```
git commit --amend
```

See Git-Squashing commits.

3. Make sure all unit tests still pass

```
mvn clean package
```

4. Push changes to your fork

```
git push -f
```

5. On github.com, submit a "pull request" to the main repo. (About pull requests: http://help.github.com/send-pull-requests/)
6. On the ticket, click the "Request Review" button and add a comment linking to the url of the pull request.

Helpful Hint: its always a good Practice , that before you push code to your remote repository , first pull all the latest changes from the remote branches (upstream and origin).

This is important in such a way that , the time you spend working on your ticket, new changes are being pushed to the remote repositories by other people , hence this may lead to merge conflicts if you try to push the same changes already pushed by some one else.

1. pull all the latest changes from the upstream branch

```
git pull --rebase upstream master
```

2. Then you can now safely push .

Synchronizing with EGit

if you are using EGit, you may need to synchronize the changes you made on your local copy against either your fork or your remote master copy (upstream).

To compare with upstream:

1. Right-click on the module to compare
2. Select 'compare with'
3. Select 'Branch / tag or reference'
4. Select 'Remote tracking'
5. Select 'Upstream/master'

To compare your changes against your fork (origin),

1. Right-click on the module to compare
2. Select 'compare with'
3. Select 'Head Revision'

Getting feedback on a pull request and making changes to it

After you requested a code review you need to wait for a full committer to look at your work and either merge your changes or ask for improvements. We are doing our best so that it does no take longer than a few days for you to get feedback.

Reviews are normally done in github as comments on your pull request (see https://github.com/openmrs/openmrs-core/pull/93 for example). You can see all comments on the discussion tab. If you did follow up commits, some comments may be hidden and you just need to click 'Show outdated diff' to see them. When a reviewer has finished commenting on the pull request, the ticket will be moved to the 'Rework Needed', 'Committed Code' or 'Closed' stage. If it is 'Committed Code' or 'Closed', your work has been merged, the pull request closed and your job is done.

In order to make improvements you just need to commit to the same branch you created for the pull request and push to your fork. Github will automatically detect new commits and add them to your pull request. After you have made all improvements you need to move the ticket back to the 'Request Review' stage. The reviewer will be automatically notified to look at your code again.

Sometimes it may take a few rounds of reviews for the code to be ready so do not be discouraged. It is our common effort to keep good quality of code.

After the ticket has been closed

- After your pull request has been accepted, the ticket is closed, and you have no further need of the branch you created, you can clean things up by deleting it from your local repository and your fork:

```
git branch -D TRUNK-123
git push origin :TRUNK-123
```

Creating your own module on github

(TODO, link to place in the "Creating Modules" hierarchy instead?)
The OpenMRS organization on GitHub (http://github.com/openmrs) can be used to host community OpenMRS projects. All bundled modules managed within git are kept under this organization with the naming convention "openmrs-module-<*moduleid*>". All community-supportes modules are welcome and encouraged to be kept under the OpenMRS organization as well.

Other organizations and authors can maintain OpenMRS modules under other accounts within github but are encouraged to use the same naming convention "openmrs-module-*moduleid*" naming convention.

1. Create the remote repository, and get the URL such as: https://github.com/openmrs/openmrs-module-appointment.git
2. Locally, at the root directory of your module's source, git init
3. Locally, add and commit what you want in your initial repo (for everything, git add .)
4. `git commit -m 'adding module to repository'`
5. To attach your remote repository with the name 'origin' (like cloning would do), git remote add origin URL From Step 1, for example:
   `git remote add origin https://github.com/openmrs/openmrs-module-appointment.git`
6. `git pull origin master`
7. `git push origin master`

Naming Conventions

When using Git for your project, please use the following naming conventions:

- Use all lowercase.
- Please include "OpenMRS module", "OpenMRS Contrib", or "OpenMRS Distro" somewhere in your project's README.

For modules

- Use the name "openmrs-module-*moduleid*" (for example: `openmrs-module-htmlformentry`)

For scripts that package a distribution

- Use the name "openmrs-distro-*xyz*" (for example: `openmrs-distro-referenceapplication`)

For variations of the OpenMRS Standalone

- Use the name "openmrs-standalone-*xyz*"

For other contributions

- Use the name "openmrs-contrib-*contribname*" (for example: `openmrs-contrib-atlas`).

When creating a new module or contrib, please e-mail code@openmrs.org to request a module ID or contribution name, including your OpenMRS ID, a description of your module/contribution, and your proposed module ID or contribname. The conversation with the "code" e-mail group is used to avoid duplicates and maximize consistency of naming; you should expect a module ID to be decided within one day of your request. Eventually, we hope to create a more automated process for ensuring uniqueness of module identification & awareness of existing modules.

Why Github?

Github is currently the largest git hosting repository available. We want to join the community of developers there as well as eliminate another headache of providing our own code hosting on our servers.

Also see

- A cool interactive git cheat sheet.
- See the htmlformentry hosted on github.
- Also see the Collaborating on HTML Form Entry Using Git page
- OpenMRS University session on best practices for claiming a ticket, working on it in git, and creating a pull request: Youtube (15m) | Adobe Connect recording (1h)