You are here

Git for developers

This document is for helping you get started on Moodle development with Git. For further details of Git, see Category:Git.

Contents

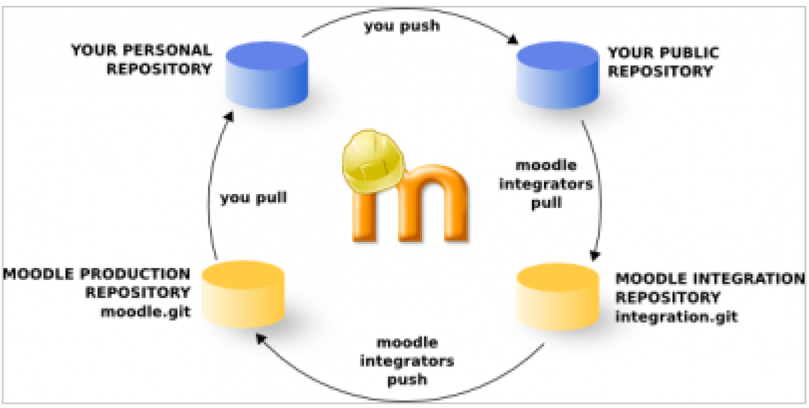
General workflow

A reasonable knowledge of the Git basics is a good idea before you start to use it for development. If you are new to Git, you are encouraged to go to 'See also' for some more general reading.

Detailed explanation of the workflow can be found in the Process page. In short, the Moodle development with Git looks like this:

- You as the contributor commit changes into your personal repository at your computer
- You push the changes into your public repository and publish links to your changes in the Moodle Tracker
- You request a peer review of your code from another developer
- When peer reviewer is happy they submit issue for integration
- Moodle integrators pull the changes from your public repository and if they like them, they put them into Moodle integration repository
- The integrated change is tested and finally pushed into Moodle production repository
- You update your local repository with all changes from the production repository and the next cycle may start again

This workflow runs in roughly weekly cycles. The integration happens on Monday and Tuesday and the testing on Wednesday. On Thursday (or Friday if testing takes too long), the production repository moodle.git is usually updated with changes from the last development week.



Moodle development workflow with Git

Most Moodle developers have their public repositories hosted at Github. Alternatively you may want to try Gitorious or the legendary repo.or.cz. In the examples in this guide we assume you'll set up your public repository at Github.

When you first register on tracker you can not assign issues to yourself or send them for peer review. You will be added to the developers group after your first bug fix is integrated. Before that just comment on the issue with a link to your branch and component lead or another developer will send issue for peer review for you.

Installing Git on your computer

Install Git on your computer. Most Linux distributions have Git available as a package to install. On Debian/Ubuntu, type '**sudo apt-get install git'** on the terminal. If you are on Mac, git-osx-installer installs it in a few clicks.

Immediately after the installation, set your name and contact e-mail. The name and e-mail will become part of your commits and they can't be changed later once your commits are accepted into the Moodle code. Therefore we ask contributors to use their real names written in capital letters, eg "John Smith" and not "john smith" or even "john5677".

```
git config --global user.name "Your Name"
git config --global user.email yourmail@domain.tld
```

Unless you are the repository maintainer, it is wise to set your Git to not push changes in file permissions:

```
git config --global core.filemode false
```

Also, it's recommended to verify that the your git installation is not performing any transformation between LFs and CRLFs. All Moodle **uses only LFs** and you should **fetch/edit and push** it that way (may need to configure your editor/IDE too). Note that having any "magic" enabled is known to cause problems with unit tests execution. So we recommend you to set:

```
git config --global core.autocrlf false
```

Setting-up the public repository

1. Go to Github and create an account.

2. Go to the official Moodle Github repository and click on the Fork button. You now have your own Github Moodle repository.

3. Now you need to set up your SSH public key, so you can push to your Github Moodle repository from your local Moodle repository. On Mac you can go on this Github help page. If you are on another system, go to your Github administration page, to the section SSH Public Keys, and you should see a link to a help page. Done? Good! That was the most difficult part!

Setting-up the local repository at your computer

Create a local clone repository of your Github repository. In a terminal:

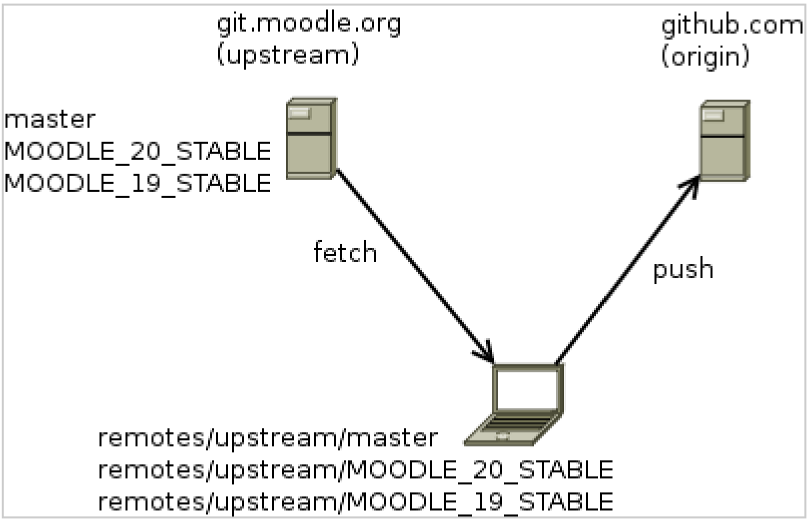
```
git clone git://github.com/YOUR_GITHUB_USERNAME/moodle.git LOCALDIR
```

```
(or:  git clone git@github.com:YOUR_GITHUB_USERNAME/moodle.git LOCALDIR)
```

This command does several jobs for you. It creates a new folder, initializes an empty Git repository in it, sets your Github repository as the remote repository called 'origin' and makes a local checkout of the branch 'master' from it. The important point to remember now is that your Github repository is aliased as 'origin' for your local clone.

Note that the format of the URL here is important. In the first example, the URL starts "git://github.com" and this will give read-only access to the repository at github.com. If you use this URL, the "git push origin" command that appears later in this document will not work. Therefore, if you want to be able to update the "origin" repository, you should use the URL that starts "git@github.com", i.e. the second of the two "git clone" commands given above. This will give you read and write access to the repository on github.com.

Keeping your public repository up-to-date



Fetching changes from upstream and pushing them to github

Your fork at Github is not updated automatically. To keep it synced with the upstream Moodle repository, you have to fetch the recent changes from the official moodle.git and push them to your public repository. To avoid problems with this it is strongly recommended that you never modify the standard Moodle branches. *Remember: never commit directly into master and MOODLE_xx_STABLE branches.* In other words, always create topic branches to work on. In Gitspeak, the master branch and MOODLE_xx_STABLE branches should be always fast-forwardable.

To keep your public repository up-to-date, we will register remote repository git://git.moodle.org/moodle.git under 'upstream' alias. Then we create a script to be run regularly that fetches changes from the upstream repository and pushes them to your public repository. Note that this procedure will not affect your local working directory.

To register the upstream remote:

```
cd moodle
git remote add upstream git://git.moodle.org/moodle.git
```

The following commands can be used to keep the standard Moodle branches at your Github repository synced with the upstream repository. You may wish to store them in a script so that you can run it every week after the upstream repository is updated.

```
#!/bin/bash
git fetch upstream
for BRANCH in MOODLE_{19..38}_STABLE master; do
    git push origin refs/remotes/upstream/$BRANCH:refs/heads/$BRANCH
done
```

How it works

The git-fetch command does not modify your current working dir (your checkout). It just downloads all recent changes from a remote repository and stores them into so called remote-tracking branches. The git-push command takes these remote-tracking branches from upstream and pushes them to Github under the same name. Understanding this fully requires a bit knowledge of Git internals - see gitrevisions(7) man page.

Note there is no need to switch the local branch during this. You can even execute this via cron at your machine. Just note that the upstream repository updates typically just once a week.

New branches

Occasionally, moodle.org will create a new branch that does not exist in your public (e.g. Github.com) repository. If you try to push this new branch, you will see an error such as the following:

```
error: unable to push to unqualified destination: MOODLE_99_STABLE
The destination refspec neither matches an existing ref on the remote
nor begins with refs/, and we are unable to guess a prefix based on the source ref.
error: failed to push some refs to 'git@github.com:YOUR_GITHUB_USERNAME/moodle.git'
```

In the above example, "MOODLE_99_STABLE", is the name of the new branch that does not exist in your public repository. To fix the error, you need to create the new branch on your public repository, using the following commands, replacing "MOODLE_99_STABLE" with the name of the new branch you wish to create:

```
git checkout MOODLE_99_STABLE
git push origin MOODLE_99_STABLE:MOODLE_99_STABLE
```

The above code will create a new copy of the "MOODLE_99_STABLE" branch in your local repository. If you do not need to keep a local copy of the new branch - and probably you do not need it, you then can remove it from your local repository as follows:

```
git checkout master
git branch -D MOODLE_99_STABLE
```

Preparing a patch

As said earlier at this page, you never work on standard Moodle branches directly. Every time you are going to edit something, switch to a local branch. Fork the local branch off the standard branch you think it should be merged to. So if you are working on a patch for 1.9 or 2.0, fork the branch off MOODLE_19_STABLE or MOODLE_20_STABLE, respectively. Patches for the next major version should be based on the master branch.

```
git checkout -b MDL-xxxxx-master_brief_name origin/master
```

Note that if you forget to specify the starting point, the branch is based on the currently checked-out branch. It may not be what you want. It is recommended to always specify the starting point.

To check the current branch, run

```
git branch
```

The current branch is highlighted.

Now go and fix the issue with your favorite IDE. Check the status of the files, view the change to be committed and finally commit the change:

```
vim filename.php
git status
git diff
git commit -a
```

For the commit message, please do respect the guidelines given on in the article Coding_style.

Note that this is safe as the commit is recorded just locally, nothing is sent to any server yet (as it would in CVS). To see history of the commits, use

```
git log
```

Once your local branch contains the change (note that it may consists of several patches) and you are happy with it, publish the branch at your public repository:

```
git push origin MDL-xxxxx-master_brief_name
```

Now as your branch is published, you can ask Moodle core developers to review it and eventually integrate it into the standard Moodle repository.

Changing commit message, reordering and squashing commits

It often happens that you made a mistake in your patch or in the commit message and helpful CiBot pointed it out for you. You can "rewrite the history" and change the existing commits.

Option 1. Reset all the changes in the branch and commit again.

```
git reset --mixed origin/master
```

Now all your changes are still present but all commits on top of "master" branch are gone. You can create a new commit

Option 2. Discover **git rebase --interactive** - this is a powerful tool to change the sequence of commit, change the commit messages, squash commits, etc. We will not cover it here, there are many articles in the Internet about it, for example: https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History.

Whatever option you chose, you have "rewritten the history" and you can not simply push the changes to github again because they would need to overwrite the commits that were already pushed. If you try "git push MDL-xxxxx-master_brief_name" you will get an error message suggesting you to force push. To force push the changed commits use:

```
git push -f origin MDL-xxxxx-master_brief_name
```

If an error occurs because you are still using the git protocol (read only), use this command :

```
git remote set-url origin https://github.com/<user.name>/moodle.git
```

A prompt will ask for your credentials, if you previously setup your SSH public key you can also use this one :

```
git remote set-url origin git@github.com:<user.name>/moodle.git
```

Checking if a branch has already been merged

After some time contributing to Moodle you would have a lot of branches both in your local repository and in your public repository. To prune their list and delete those that were accepted by upstream, use the following

```
git fetch upstream                                    (1)
git branch --merged upstream/master                   (2)
git branch --merged upstream/MOODLE_20_STABLE         (3)
```

The command (1) fetches the changes from your upstream repository at git.moodle.org (remember that git-fetch does not modify your working dir so it is safe to run it whenever). Command (2) and (3) print all branches that are merged into the upstream master branch and MOODLE_20_STABLE branch, respectively. To delete these local branches, use

```
git branch -d MDL-xxxxx-master_accepted_branch
```

The similar approach can be used to check the branches published at your origin repository at github.com

```
git fetch origin                                      (1)
git fetch upstream
git branch -r --merged upstream/master                (2)
git branch -r --merged upstream/MOODLE_20_STABLE      (3)
```

The command (1) makes sure that you have all your branches from github.com recorded as the remote tracking branch locally. Commands (2) and (3) work the same as in the previous example but they list remote tracking branches only (see -r param). To delete a branch at github.com, use

```
git push origin :MDL-xxxxx-master_branch_to_delete
```

This syntax may look weird to you. However it is pretty logical. The general syntax of the git-push command is

```
git push <repository> <source ref>:<target ref>
```

so deleting a remote branch can be understood as pushing an "empty (null) reference" to it.

Peer-reviewing someone else's code

To review a branch that someone else pushed into their public repository, you do not need to register a new remote (unless you work with such repository frequently, of course). Let us imagine your friend Alice pushed a work-in-progress branch called 'wip-feature' into her Github repository and asked you to review it. You need to know the read-only address of the repository and the name of the branch.

```
git fetch git://github.com/alice/moodle.git wip-feature
```

This will download all required data and will keep the pointer to the tip of the wip-feature branch in a local symbolic reference FETCH_HEAD. To see what's there on that branch, use

```
git log -p FETCH_HEAD
```

To see how a particular file looks at Alice's branch

```
git show FETCH_HEAD:admin/blocks.php
```

To create a new local branch called 'alice-wip-feature' containing the work by Alice, use

```
git checkout -b alice-wip-feature FETCH_HEAD
```

To merge Alice's work into your current branch:

```
git merge FETCH_HEAD
```

To see what would be merged into the current branch without actually modifying anything:

```
git diff ...FETCH_HEAD
```

Once you are all set and reviewing code, this checklist should prove to be useful.

Rebasing a branch

Rebasing is a process when you cut off the branch from its current start point and transplant it to another point. Let us assume the following history exists:

```
      A---B---C topic
     /
D---E---F---G master
```

From this point, the result of the command:

```
git rebase master topic
```

would be:

```
              A'--B'--C' topic
             /
D---E---F---G master
```

and would end with 'topic' being your current branch.

You may be asked to rebase your branch submitted for the integration if the submitted branch was based on an outdated commit. The typical case is if you create a new branch as a fork off the upstream master branch on Tuesday. Then on Wednesday, the upstream master branch grows as all changes from the last integration cycle are merged to it. To make diff easy on Github for next weekly pull request review, you want to rebase your branch against the updated master.

```
git rebase master MDL-xxxxx-master_topic_branch
```

Note that rebasing effectively rewrites the history of the branch. **Do not rebase the branch if there is a chance that somebody has already forked it and based their own branch on it.** For this reason, many Git tutorials discourage from rebasing any branch that has been published. However in Moodle, all branches submitted for integration are potential subject of rebase (even though we try to not to do it often) and you should not base your own branches on them.

Conflicts during rebase

During the rebase procedure, conflicts may appear. git-status commands reports the conflicted files. Explore them carefully and fix them in your editor (like you would do with CVS). Then add the files with 'git add' command and continue.

```
vim conflicted.php
git add conflicted.php
git rebase --continue
```

Applying changes from one branch to another

Most bugs are fixed at a stable branch (like MOODLE_20_STABLE) and the fix must be prepared for other branches, too (like MOODLE_21_STABLE and the main development branch - master). In Moodle, we do not merge stable branches into the master one. So usually the contributor prepares at least two branches - with the fix for the stable branch(es) and with the fix for the master branch.

If you have a patch prepared on a local branch (let us say MDL-xxxxx-20_brief_name), it is possible to re-apply it to another branch.

Cherry-picking a single commit

Let us have two local Git repositories ~/public_html/moodle21 containing local installation of Moodle 2.1 and ~/public_html/moodledev with the local installation of most recent development version of Moodle. They both use your public repository at github.com as the origin. You have a branch in moodle21 called MDL-xxxxx-21_topic that was forked off MOODLE_21_STABLE. It contains one commit. Now you want to re-apply this commit to a branch MDL-xxxxx-master_topic in moodledev.

```
cd ~/public_html/moodledev
git checkout -b MDL-xxxxx-master_topic origin/master    (1)
git fetch ../moodle21 MDL-xxxxx-21_topic                (2)
git cherry-pick FETCH_HEAD                              (3)
```

The command (1) creates new local branch forked off the master. The command (2) fetches all data needed to re-apply the topic branch and stores the pointer to the tip of that branch to FETCH_HEAD symbolic reference. The command (3) picks the tip of the branch (the top-most commit on it) and tries to apply it on the current branch.

There is also a variant of the cherry-pick command that supports multiple commits, shortly (see its man page for details):

```
$ git cherry-pick A^..B
```

if you want to include from A - see ^ - to B, A should be older than B. We will use another approach for cherry-picking multiple commits.

Applying a set of patches

If the branch MDL-xxxxx-21_topic from the previous example consists of several commits, it may be easier to use git-format-patch and git-am combo to re-apply the whole set of patches (aka patchset). Firstly you will export all commits from the topic branch to files.

```
cd ~/public_html/moodle21
mkdir .patches
git format-patch -o .patches MOODLE_21_STABLE..MDL-xxxxx-21_topic         (1)
```

The command (1) takes all commits from the topic branch that are not in MOODLE_21_STABLE and exports them one by one to the output directory .patches. Look at the generated files. They contain the patch itself (in diff format) and additional information about the commit. You could eg send these files by email to a friend of yours for peer-review. We will use them in another repository.

```
cd ~/public_html/moodledev
git checkout -b MDL-xxxxx-master_topic origin/master
git am -3 ../moodle21/.patches/*                        (1)
```

The command (1) applies all the files from the .patches directory. When a patch does not apply cleanly, the command tries fall back on 3-way merge (see the -3 parameter). If conflicts occur during the procedure, you can either deal with them and then use `git am --continue` or abort the whole procedure with `git am --abort`.

See also

- Git tips

Moodle forum discussions

- GIT help needed
- Best way to manage CONTRIB code with GIT
- Handy Git tip for tracking 3rd-party modules and plugins
- Moodle Git repositories
- Git help!! I don't understand rebase enough...
- add MOODLE_24_STABLE to github.com repository

External resources

- Everyday GIT With 20 Commands Or So
- 'Pro Git' complete book
- Getting git by Scott Chacon - an recording of an excellent 1-hour presentation that introducing git, including a simple introduction to what is going on under the hood.
- Tim Hunt's blog: Fixing a bug in Moodle core: the mechanics
- Flight rules for Git

Retrieved from "https://docs.moodle.org/dev/index.php?title=Git_for_developers&oldid=56959"

Category: Git

- This page was last modified on 27 February 2020, at 17:42.
- Content is available under GNU General Public License unless otherwise noted.