You are here

Git for Administrators

This page describes how to maintain a copy of Moodle on your production server which can easily be upgraded using Git. If you have customisations of Moodle core code, you are advised to follow the instructions in the Git for developers guide.

To get the most of Git it is worth making the effort to understand its basic concepts - see the section below. It can be a bit of a steep learning curve, especially if you are used to CVS or Subversion.

Contents

Getting hold of Git (Windows, OSX, Linux and others)

Support for Git was, up until recently, mostly confined to Linux but builds are now available for most popular operating systems:

- List of downloads from Git site - http://git-scm.com/download

Once you have downloaded and installed your OS relevant git installation, the git commands in this document should work with your operating system.

Obtaining the code from Git

The command line version of Git is discussed here. Graphical clients are little more than wrappers around the command line version, so you should be able to deduce the correct parameters quite easily.

You can find the official Moodle git repository at git://git.moodle.org/moodle.git (with an official clone at git://github.com/moodle/moodle.git). To initialize your local checkout, use

```
$ cd /path/to/your/webroot
$ git clone git://git.moodle.org/moodle.git                       (1)
$ cd moodle
$ git branch -a                                                   (2)
$ git branch --track MOODLE_38_STABLE origin/MOODLE_38_STABLE     (3)
$ git checkout MOODLE_38_STABLE                                   (4)
```

- The command (1) initializes the new local repository as a clone of the 'upstream' (i.e. the remote server based) moodle.git repository. The upstream repository is called 'origin' by default. It creates a new directory named *moodle*, where it downloads all the files. This operation can take a while as it is actually getting the entire history of all Moodle versions
- The command (2) lists all available branches.
- Use the command (3) to create a new local branch called MOODLE_38_STABLE and set it to track the remote branch MOODLE_38_STABLE from the upstream repository.
- The command (4) actually switches to the newly created local branch.

Git Connection Refused Error

- If connection refused, use: $ git clone https://github.com/moodle/moodle.git

fatal: unable to connect to git.moodle.org: git.moodle.org[0: 34.210.133.53]: errno=Connection refused

Note that Git has a huge number of options for each command and it's actually possible to do the above process with a single command (left as an exercise!!).

Git from behind a firewall

Git uses a read-only protocol that may be blocked by your firewall (port 9418). If this is a problem, you can use Github's http version https://github.com/moodle/moodle.git. It's a bit slower, so use the Git protocol if you can.

Updating your installation

The Moodle development team performs integration and testing of fixed bugs every Monday and Tuesday. On Wednesday you can install all patches by updating your code. Check the shortlog to see if the official repository has been already updated or not.

To update your code to the latest version (on the MOODLE_38_STABLE branch) **all** you have to do is:

```
$ cd /path/to/your/moodle/
$ git pull
```

If this is a production site you should still consider the Upgrade instructions (e.g. take backups).

Installing a contributed extension from its Git repository

This is one way to handle adding plugins from other Git repositories into your Moodle repository. Another way is to use Git Submodules. However, at the time of writing, this is one of Git's rougher features and should be regarded as an advanced option.

For example, let us say we want to install the Certificate module from its Git repository into our Moodle 3.8.

```
$ cd /path/to/your/moodle/
$ cd mod                                                          (1)
$ git clone https://github.com/markn86/moodle-mod_certificate.git certificate     (2)
$ cd certificate
$ git checkout -b MOODLE_38_STABLE origin/MOODLE_38_STABLE        (3)
$ git branch -d master                                            (4)
```

The command (1) changes the current directory into the *mod* folder of your local Moodle clone. The command (2) creates a new subdirectory *certificate* and makes a local clone of vanilla Certificate repository. The command (3) creates a new local branch that will track the remote branch with a Certificate version for Moodle 3.8. The command (4) deletes the *master* that was created automatically by git-clone in (2) as we do not want it in this production checkout.

Note: you should check first the compatibility of a module with your Moodle branch by asking directly to the Maintainer before cloning the repo or - if you want to guess it - by issuing the command below before running the command (3), in order to verify what is available among the branches:

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/MOODLE_20_STABLE
  remotes/origin/MOODLE_21_STABLE
  remotes/origin/MOODLE_22_STABLE
  remotes/origin/MOODLE_23_STABLE
  remotes/origin/MOODLE_24_STABLE
  remotes/origin/MOODLE_25_STABLE
  remotes/origin/MOODLE_26_STABLE
  remotes/origin/MOODLE_27_STABLE
  remotes/origin/MOODLE_28_STABLE
  remotes/origin/MOODLE_29_STABLE
  remotes/origin/MOODLE_30_STABLE
 remotes/origin/MOODLE_31_STABLE
  remotes/origin/master
```

This will avoid an error message when you issue the command (3) against a nonexistent branch, e.g.:

```
§ git checkout -b MOODLE_31_STABLE origin/MOODLE_31_STABLE
fatal: git checkout: updating paths is incompatible with switching branches.
Did you intend to checkout 'origin/MOODLE_31_STABLE' which can not be resolved as commit?
```

```
Note: To fix above error, use: "git fetch origin MOODLE_31_STABLE:LOCAL_MOODLE_31_STABLE"
```

Now it is wise to add the new directory mod/certificate/ to the list of ignored files of the main Moodle clone, otherwise a status of the main clone will keep reminding you that the new code has not been checked in.

```
$ cd /path/to/your/moodle/
$ echo /mod/certificate/ >> .git/info/exclude
```

To update your Moodle installation now, you must visit both Git repositories and pull changes from upstream.

```
$ cd /path/to/your/moodle/
$ git pull
$ cd mod/certificate
$ git pull
```

Writing a shell script with these lines in the root of Moodle installation is a very good idea. Otherwise it is easy to forget what Git repositories are there within the main Moodle repository.

Installing and maintaining contributed extensions using Git submodules

As it was said in the previous section, this is for advanced users only. Therefore it is necessary, that you have some experience with Git and its commands. A step-by-step explanation will be provided, but in order to follow these steps it is helpful to understand, what these commands do.

Advanced options and commands can be found at [Git book]. If you have any questions about Git submodules, please visit the site above first.

Installing a new extension into an existing Moodle

As an example we use the Certificate module from the previous section.

```
$ cd /path/to/your/moodle
$ git submodule add https://github.com/markn86/moodle-mod_certificate.git mod/certificate
```

Note, that Git is reporting two new files in the repository:

```
$ git status
# On branch MOODLE_29_STABLE
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   .gitmodules
#       new file:   mod/certificate
#
```

The file **.gitmodules** contains the local path and url of all your submodules. It has to be committed, if you intend to clone the repository later (see the page Moodle development environment with Git submodules). Before commiting, make sure to check the configuration of the plugin's Git repository, since the automatically generated settings may be not sufficient. For future updates it is helpful, to track the remote branch, which corresponds to the Moodle version of your repository.

```
$ cd mod/certificate
$ git branch -avv
* master                          345f5b1 [origin/master] Replaced deprecated function
  remotes/origin/HEAD             -> origin/master
  remotes/origin/MOODLE_20_STABLE 1aa1040 Added option to print 'grade category' grade
  remotes/origin/MOODLE_21_STABLE 1aa1040 Added option to print 'grade category' grade
  remotes/origin/MOODLE_22_STABLE 1aa1040 Added option to print 'grade category' grade
  remotes/origin/MOODLE_23_STABLE fe047de Check that the function exists rather than relying on the Moodle version
  remotes/origin/MOODLE_24_STABLE 1051f7d CONTRIB-4892 Fixed the email to others functionality
  remotes/origin/MOODLE_25_STABLE cdb221a CONTRIB-4946: Removed character from language file breaking AMOS
  remotes/origin/MOODLE_26_STABLE 696802a Increased version
  remotes/origin/MOODLE_27_STABLE d3c0379 Increased version
  remotes/origin/MOODLE_28_STABLE fa8df83 Increased version
  remotes/origin/MOODLE_29_STABLE 3f03740 Replaced deprecated function
  remotes/origin/master           345f5b1 Replaced deprecated function
```

Git created the branch **master** which tracks **origin**/**master** automatically, because the remote repository has checked out **master**. Therefore, create a new branch, which tracks the appropriate remote branch. Of course, this is only possible, if the remote repository offers those branches.

```
$ git checkout -b MOODLE_29_STABLE origin/MOODLE_29_STABLE
Branch MOODLE_29_STABLE set up to track remote branch MOODLE_29_STABLE from origin.
Switched to a new branch 'MOODLE_29_STABLE'
$ git branch -D master
Deleted branch master (was 345f5b1).
```

It is not necessary to delete the **master** branch, but it's useless to keep it. In fact, these settings don't need to be touched afterwards.

The final step is to commit the changes to the main repository.

```
$ cd /path/to/your/moodle
$ git commit -a -m "New extension mod_certificate installed"
```

It has to be ensured, that the commit includes only the changes for the new Git submodule (since **-a** commits all non-staged changes).

Maintaining Git submodules

Maintaining a set of submodules is extremely easy. Consider a Moodle repository with several submodules installed. Keep in mind, that the extension **mod_mylittleextension** is a fake plugin, created for a test scenario in this example. It is not an official Moodle module. For updating all your submodules at once, type in:

```
$ cd /path/to/your/moodle
$ git submodule foreach git pull
Entering 'block/coursefeedback'
Already up-to-date.
Entering 'mod/certificate'
```

```
Already up-to-date.
Entering 'mod/mylittleextension'
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From /local/repositories/mle
   89d9eae..64c122d  master     -> origin/master
Updating 89d9eae..64c122d
Fast-forward
 index.html  |    9 +++++++++
 version.php |    6 +++---
 2 files changed, 12 insertions(+), 3 deletions(-)
 create mode 100644 index.html
$ git status
# On branch MOODLE_29_STABLE
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   mod/mylittleextension (new commits)
#
```

The command **git submodule foreach [another command]** walks through all submodule repositiories and executes what is specified by **[another command]**. In this case it is **git pull**. Therefore the module mylittleextension was updated and the main repository isn't clean anymore until changes are committed:

```
$ git commit -a -m "Plugin updates"
```

Maintaining plugins with Git submodules has also another application than simplifying the update process. In a greater scale it can be used to maintain a Moodle project, where multiple developers need to have an exact copy of your moodle without organizing external plugins manually. You can read more about this topic at the page Moodle development environment with Git submodules.

See also

- Windows installation using Git
- Git for Mac
- dev:Moodle versions
- For fixing a Tracker Issue (MDL) / Forking Moodle / CONTRIButing code User:Sam_Hemelryk/My_Moodle_Git_workflow
- Case study Git + Moodle from Technical University Berlin

Moodle forum discussions

- Github and Moodle deployment for production
- GIT help needed
- Clear git guide for Admins (not developers)
- Best way to use Git

External resources

- Deploying Moodle from git - Blog post from a production experience
- Git Reference
- Pro Git book

Retrieved from "https://docs.moodle.org/38/en/index.php?title=Git_for_Administrators&oldid=136000"

Category: Installation

- This page was last modified on 22 November 2019, at 12:53.
- Content is available under GNU General Public License unless otherwise noted.