Git Usage Guide

April 21, 2019

Contents

1 When in Doubt

The go-to command to see what is going on in a git repository is

```
git status
```

This will display things such as changes to any tracked files, the presence of untracked files, changes that have been staged for committing, and commit differences between the local and remote repositories. It will also display help text on how to perform common actions on these objects.

1.1 Quick Reference

Common actions and their corresponding commands are listed here.

- Save a new snapshot: `git add <files>`, then `git commit`
- Revert file to last snapshot (warning, no undo): `git checkout <file>`
- Unstage a file: `git reset HEAD <file>`
- Upload new commits to remote repository: `git push <remote-name> <branch-name>`
- Download new commits from remote repository: `git pull <remote-name> <branch-name>`
- Download existing repository from server: `git clone user@server:<path-to-repo>`
- Temporarily store any changes and revert to last commit: `git stash`, retrieve changes with `git stash pop`
- Change branch: `git checkout <branch-name>`, add `-b` flag to create a new one

2 First time configuration

If you have not used git on your current computer before, you will need to configure three things:

- `user.name`
- `user.email`
- `core.editor`

The first two options, `user.name` and `user.email`, are what your commits will be attributed under. The third option, `core.editor`, is what will be used when git requires you to enter text (typically for a commit message).

To configure an option, use

```
git config --global <key> '<value>'
```

For example, you might use

```
git config --global core.editor 'emacs -nw'
```

to use terminal-mode emacs as the default editor.

3 Creating a Repository

3.1 Local Repositories

To create a new git repository on your local machine, create a new directory, `cd` into it, and use

```
git init
```

If you have a pre-existing project folder, you can just run the same command from within it.

3.2 Remote Repositories

To create a new git repository on the server (via ssh), create a new directory, `cd` into it, and use

```
git --bare init
```

The naming convention for the directories created this way is `<project-name>.git`. You should not use any other git commands on the server directly. Any further management of the server-side repository is done from your local repository (see section 6).

3.3 Initial Commit

The initialisation command will setup the current directory as a blank git repository. This means it will have no past history, and will not track any files. To start tracking project files, go to your *local* repository and use

```
git add <file>
```

to mark the files you wish to be version controlled. Note that globbing works as expected, and adding a subdirectory will automatically add everything within it recursively. Once you have staged all the files you wish to be tracked, use

```
git commit
```

to permanently save the added files as a snapshot that can be checked-out later. You will be prompted to enter a commit message under your default editor. Note that this will store the state of the files as they were when you used the `add` command on it, not their state when you call `commit`.

4 Updating a Local Repository

You have complete control over when git will save snapshots of your project files, as it will only do so when you call `git add` and `git commit`. To save a snapshot, first specify the changes you want to be saved by calling

```
git add <files>
```

on the files of interest. This will save the state of the files at the time of execution, and stage them for commit. If you wish to remove a staged change, use

```
git reset HEAD <file>
```

If you wish to remove a file from being tracked in the repository, use

```
git rm <file>
```

Once you are happy with the changes you have staged, call

```
git commit
```

to finalise the creation of the new snapshot. This will prompt you to enter a commit message in the default editor to describe the changes. The convention for commit messages is to enter a short summary of the changes on the first line, optionally followed by more detailed descriptions and/or context starting from the third line if needed. For example:

```
1 Update makefile
2
3 Changed compiler to gcc from cc, and added warning/optimisation flags. Still
4 need to update some dependencies.
```

By default, the editor will also show a commented list of changed, staged and untracked files in the commit template to help you remember what you've done. You can also have git show file diffs in the commented section by adding the `-v` flag. If you have a simple commit message and don't wish to use a full editor, you can specify your message in the `git commit` command by adding it as a quoted string after the `-m` flag, eg:

```
git commit -m 'Initial commit'
```

5 Branches

Branches allow you to have multiple parallel and independent streams of development. By default, git repositories are created with only one branch called `master`. To switch to another branch, use

```
git checkout <branch-name>
```

If that branch does not exist yet (ie to create a new branch), add the `-b` flag. You can see a list of existing branches with

```
git branch
```

You can also check your currently checked-out branch with

```
git status
```

5.1 Merging

To integrate changes from one branch into another, use

```
git merge <incoming-branch>
```

This will merge in changes from the incoming-branch into your currently checked-out branch, and leave the incoming-branch unchanged.

If there is divergent history between the two branches, you will need to manually resolve the merge. First, find which files have merge conflicts using `git status`. Open the offending files with your preferred text editor and search for the markers "`<<<<<<<`", "`=======`", and "`>>>>>>>`" - these will delineate the conflicting changes from the two branches. Edit these areas to reflect what should be in the final state (making sure to remove any added markers that should not be in the file), and stage the files. Once this has been done for all conflicts, you can commit the merge with the usual `git commit`.

6 Syncing with a Remote Repository

Commits (and branches) you create on a local repository will only exist on that local repository until you explicitly upload them to a remote repository.

6.1 Remotes

To start working with remote repositories, you should setup a name association for the remote url to make later commands easier. You can create a named remote with

```
git remote add <remote-name> <remote-url>
```

Typically, the remote name `origin` is used. The url is given in the scp-like syntax of `user@host:<path>` (we are using the ssh protocol for git). For example, this might look like

```
git remote add origin git@192.168.1.51:test.git
```

The path after the colon is relative to the home directory of the specified user, however you may specify absolute paths if you choose. Depending on the permissions set on the directories, you may be able to access another user's repositories by supplying a path such as `/albert/my-repo.git` (some handy symlinks have been added to facilitate this). The full command would then look like

```
git remote add origin git@192.168.1.51:/albert/my-repo.git
```

You can print out a list of remotes and their urls using

```
git remote -vv
```

This might be useful to verify your setup has worked as intended.

6.2 Syncing

To upload the commits on one of your branches, use

```
git push <remote-name> <branch-name>
```

Here remote-name is the name you gave for the remote you added in the previous step (`origin`), and branch-name is the name of the branch which has your commits (probably `master`).

To download new commits from the remote (if someone else has been working on the same repository) and checkout the latest changes, use

```
git pull <remote-name> <branch-name>
```

The parameters here parallel those from `push`, although branch-name refers to the branch on the remote repository now instead of your local branch. If the currently checked-out branch does not match the remote branch you are pulling, it will attempt to being a merge.

If you want to download the new commits but not immediately check them out, use

```
git fetch <remote-name>
```

instead. You can then checkout the changes manually with

```
git checkout <branch-name>
```

If you wish to copy an existing repository on the server to your local machine, use

```
git clone <url>
```

The url has an identical form to what was used to specify a remote previously. In fact, using the `clone` command will automatically set up a remote named `origin` with the same url that you passed to `clone`.