Version control using Git

Tommy S. Alstrøm

Document version 0.1

November 16, 2015

1 Introduction

This document is a draft that contains condensed information about how to use GIT.

2 Installing git

Installation instructions depend on your operating system. To download git and get instructions for your specific OS, go to
`https://git-scm.com/downloads`

For Mac, Linux and Unix, this will install git so it can be used from your terminal. The Windows installer will install a unix like eco-system on your computer, called "git bash".

2.1 For Windows users

When installing git, you should choose "Use Git from the Windows command prompt", but otherwise use default settings unless you're an expert user that knows what the other options implicate. Unfortunately, the git bash eco system does not include make, so you will have to install this separately if you want to use Makefiles. This can be done using these steps:

1. Download make from
   `http://gnuwin32.sourceforge.net/packages/make.htm`
   Choose the complete package, or use this url as direct download:
   `http://gnuwin32.sourceforge.net/downlinks/make.php`
2. Install the package
3. Open a console (cmd.exe)
4. Go to the git bash directory - e.g.
   ```
   cd C:\Users\tommysl\AppData\Local\Programs\Git\usr\local
   ```
5. in the console, type
   ```
   mklink /J bin "C:\Program Files (x86\GnuWin32\bin"
   ```

If you prefer to have a more complete unix like eco system on your computer, I suggest you look up MSYS2:
`https://msys2.github.io/`
This system has a package manager, called Pacman and you can configure this greatly. You will also be able to use git from that system instead of git bash (or you can have both). If you decide to pursue this route, you can read about pacman here:
`https://wiki.archlinux.org/index.php/pacman`

3 Getting started

To get started with your first git repository (usually denoted repo), go to a directory that contains the files you want to have under revision and type

```
# git init
Initialized empty Git repository in <path>
```

This command creates the .git directory which contains the “system files” of git. Adding files is done with

```
# git add <filename>
```

Before you’re able to commit changes, you need to identify yourself to git. Do this by these commands

```
# git config --global user.name "Tommy S. Alstrom"
# git config --global user.email "tsal@dtu.dk"
```

And you can always see the values of all global settings by typing

```
# git config --global --list
user.name=Tommy S. Alstrom
user.email=tsal@dtu.dk
```

Now, create a text file with some content, and add it to your git repo. A file in git can have four different states:

- Untracked - files that git does not track.
- Unmodified - files that are unchanged compared to last commit.
- Modified - files that are modified since last commit.
- Staged - files ready to be committed.

The command

```
# git add <file>
```

updates the state of a file to ”staged”. The command

```
# git commit -m "commit message"
```

commits staged files to the repo. The command

```
# git status
```

will show the current status of the directory, that is, how the directory is different compared to the git repo. The command

```
# git ls-tree -rt master --name-only
```

will show all files that is currently tracked by git. Now, create some text files, and get comfortable with above commands. Play with changes the status of files and move the status from unmodified -¿ modified -¿ staged -¿ committed. You will also notice that the same file can both modified and staged at the same time. To get help, type

```
# git help <cmd>
```

for any one git command, and the help will be displayed. To see what you have committed to the repo use `git log`. The most common variants of this command are

```
# git log
# git log --stat
# git log -1
# git log --since=2.weeks
```

Remember that these options can be combined.

4 Diffs

To view differences in content of the files a diff tool is used. Git has a build in diff tool, but an external can also be installed. Comparing modified content with staged content use

```
# git diff
```

You can also specify the filename if there is a specific file you're interested in. To see differences of staged file compared to commit - that is - what will be written to the index at next commit, use

```
# git diff --staged
```

To compare the current modified file to the last one committed, use

```
# git diff head
```

These changed can also be seen launching the git-gui. Play around with the diff tool and git-gui to get comfortable with these command and different states.

I recommend to install a visual diff tool. I recommend the tool called p4merge, but to get a full list of visual tools that git support, type

```
# git difftool
This message is displayed because 'diff.tool' is not configured.
See 'git difftool --tool-help' or 'git help config' for more
    details.
'git difftool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld kompare gvimdiff diffuse
    diffmerge ecmerge p4merge araxis bc codecompare emerge vimdiff
```

To use p4merge, download the tool from
`https://www.perforce.com/downloads/helix#clients` After installing the tool, set git to use this as default

```
# git config --global diff.tool "p4merge"
```

Now you can use `git difftool` the same way as you used `git diff` but now the visual tool will be launched instead.

5 Branching

Create a new branch by typing

```
# git branch <newbranch>
```

Switch to a given branch by typing

```
# git checkout <newbranch>
```

Create a couple of text files and run through the branching example in chapter 3.2 in the text book:

`https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging`
Try and force a merge with conflict. Create test branch and in the same file in two different branches, change the same line.

6 Uploading your git repo to a server

In order to upload your git repo to a server, you need an account. Once you have a git URL give, you add these with the git remote command.

```
# git remote add origin <giturl>
# git push -u origin master
```

In order to update a repo from a remote, use

```
# git pull
```

When adding a new repo using a clone command, the remotes are already set up

7 Common pitfalls and how to amend

git config –global option "value" will still work even though the option does not exists. So if git starts to act weirdly, make sure you haven't misspelled an option.

If you've added a file to the staging area, you didn't want to add, use

```
# git rm --cached FILE
```

If you want to amend last commit, use

```
# git commit --amend
```

If you committed something that you'd like to undo, use

```
# git reset --soft HEAD\string^
```

This command will put your last commit back to the staging area. Beware though, if the staging area already had content, this content will be concatenated to the last commit. E.g. if you amend one file, but have another file in the staging area, the staging area will not contain two files.

If you want to reset a file to last commit

```
# git reset -- <file>
```

8 Useful options

If git complains about line endings, use

```
#git config --global core.safecrlf false
```

If you want to skip the stage area and commit modified files directly, use

```
# git commit -a
```

If you are unfamiliar with vi (the default editor that git uses), you might want to change the editor. On windows, you might want to use notepad instead, do achieve this type

```
# git config --global core.editor "notepad"
```

9 Large repos and big files

In general, due to the fact that the .git folder must be on the local computer, large git repos (in terms of size) are very cumbersome with git. Git is simply designed for code developing in open-source environments, not for having monster repos and/or large binary files (by large, we mean gigabyte size) Linus (the creator of Git) talks about git and large files:
`http://osdir.com/ml/git/2009-05/msg00051.html`

The other thing you pay by having complete distributed version control (the .git folder) is that you must always download the entire repo. No selective downloading can be done. This is in part solved by putting websites on top of git repos. These serve as a kind of client.

On DTU compute, these issues are solved by powering git with a perforce server, and this is the very reason we advocate a hybrid system and not a pure git solution.