Git Tutorial

Version control with Git

Version control systems such as Git are vital to good software development. They allow programmers to maintain a detailed history of all the changes made to the codebase, and allow groups of people to work concurrently on different pieces of code without interfering with each other. It allows to manage source code changes, and maintain multiple branches that represent different versions of the project.

At a high level, a Git *repository* consists of a group of files (including directories) for which changes should be tracked. When a modification is made to any of those files, Git keeps track of what was added and deleted, and the programmer can group modifications into a *commit*. When the commit is made, those modifications are placed in the repository history as a "snapshot," allowing the programmer to return to the state of the repository at that commit if they wish. Since Git only tracks modifications to files, as long as two people do not modify the exact same lines in a file, two concurrent commits can be merged without conflict.

Fun fact: Linus Torvalds is the creator of Linux and Git (in fact Git was created for managing the Linux kernel, because existing version control systems were not good enough). Torvalds said that he named both projects after himself (git is British slang for an idiot).

Initializing a repository

In this tutorial, we will use the command-line version of Git. It should be installed by default in your virtual machine. The `$` is used to indicate the shell prompt, and should not actually be entered into the command.

To initialize a Git repository, run

```
$ git init
```

This will initialize a new Git repository in the current directory. Git will store all of its internal information in a folder called `.git/`. To remove the repository simply remove the `.git/` folder. At this point all files in the directory are untracked.

It is also possible to download Git repositories that exist on websites onto your computer. To do so, use the command

```
$ git clone <repository_name>
```

This will create a directory called `repository_name` with all the files that are tracked in the Git repository.

Checking the status

Git provides a command to check the status of the repository. This will list the current changes to the repository since the last commit, and the changes that are staged for the next commit. To check the status run

```
$ git status
```

Tracking files and staging modifications

If a file is currently untracked, it can be added to the repository with the command

```
$ git add <file>
```

In addition, if a file is already being tracked, but has unstaged changes since the last commit, those changes can also be staged with

```
$ git add <file>
```

Multiple files can be passed to the add command. For example

```
$ git add .
```

will add any untracked files in the current directory to the Git repository, and will stage the changes of any tracked files in the repository. Try running `git status` after making changes to your repository. You should see that changes have become staged.

To avoid tracking files, the files can be added to a special file named `.gitignore` and placed in the root of the repository (in the same place as the `.git/` folder). File name globs can be used in the `.gitignore` as well. For example, if you never want to track `.pdf` files, or files called `test.txt`, you could use a `.gitignore` consisting of the two lines

```
*.pdf
test.txt
```

Viewing changes

Git can show you the modifications you have made since the last commit. To see the unstaged changes, run

```
$ git diff
```

This will display every line that was modified. Lines with additions are displayed in green and lines with deletions are displayed in red.

To close the viewer press `q`.

To view the changes that you have already staged, run

```
$ git diff --cached
```

Committing changes

Once you have staged the files which contain the changes you want to commit, you can commit the changes to create a snapshot of the current state of the repository. Run

```
$ git commit -m "<message>"
```

to create a commit with the given message. It is recommended that messages be written in the present tense, rather than the past tense, such as:

```
$ git commit -m "Add git tutorial"
```

(that will be my commit message after I finish writing this guide).

If you run `git commit` without a message, Git will open an editor for you to write a longer message.

Viewing Git history

To view the history of a Git repository, run the command

```
$ git log
```

This will display all the commits that have been made to the repository along with their author, message, commit date, and unique hash identifying the commit. To close the viewer press `q`. For a more abbreviated version, run

```
$ git log --oneline
```

Syncing Git repositories

One of the main features of Git is the ability to synchronize a repository and development to a codebase across multiple computers. So far, all the commands have only modified the local repository that is stored on your computer (the information is in the `.git` directory). A Git repository can be associated with a URL which will allow you to push your commits to the internet, and pull updates that other people have pushed from the internet. Github is a website which provides hosting for Git repositories, as well as the ability to browse the repository from the internet, among other features.

If you don't already have a Github account, it is a good idea to create one now. To associate a local Git repository with a Github URL, first create a repository on Github by clicking the `+` symbol in the upper right and selecting repository. Give it a name, and make sure to make it private if you are storing your course projects in it. Once you have created the repository, Github will show you instructions for pushing from an existing Git repository (or instructions for creating a new one which just involves running `git init`, making a commit, and then doing the push instructions). The important commands are

```
$ git remote add origin <URL>
```

which adds the URL as the "remote" URL this repository is associated with.

Next run

```
$ git push -u origin master
```

This pushes all your commits online and associates them with the master branch of the origin remote (which you just added as the Github URL).

From now on, when you make a commit and want to push your changes online, you can run

```
$ git push
```

and to update from the online repository, run

```
$ git pull
```

More features

This was only an introduction to Git, and there are many more features to be covered. For more in-depth guides, please see the following resources:

- CS61 Git tutorial: https://cs61.seas.harvard.edu/site/ref/git/
- SEAS Git tutorial: https://wiki.harvard.edu/confluence/display/USERDOCS/Introduction+To+GIT
- Git from the Bottom Up (very extensive): https://jwiegley.github.io/git-from-the-bottom-up/ (pdf link)