Basics of working with Git

Relevant Links

- Pro git book[1]
- Git command reference[2]
- Read this[3] Chapter 2 (at least skim through it)

Notes

Starting a repository

- The most common way to start work is by cloning an existing repository, and we will learn about that later. For now, we will learn about starting a fresh repository.
- Start by going to the directory that you want to have become a repository.
- We can type `git init` to start maintaining the current directory as a repository. It is OK if there is stuff already there, it will not be deleted or anything like that. But for now, do it in a new directory.

Recording changes

Look at this picture[4] for a good idea of the different stages that files can be in.

The main command used to get information about the status of your files is `git status`. If we try this on our empty directory, we get a nice clean message that there's nothing to do. Let's get something in there. Open the folder in Sublime Text. You can do it by typing "`subl .`"

Let's create a new HTML file, call it `testFile.html`. Try these contents:

```
<!DOCTYPE html>
<html>
    <body>
        <p>Hello world!</p>
    </body>
</html>
```

Save the file, and back on the terminal type `git status`. It should tell you that you now have one untracked file.

To start tracking this file, you need to add it:

```
git add testFile.html
git status
```

[1] http://git-scm.com/book/en/v2
[2] http://git-scm.com/docs
[3] http://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository
[4] http://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

You should see that the file now is staged to be committed when we make the next commit. Git also tells you what to do if you want to unstage it. Let's try it:

```
git rm --cached testFile.html
git status
git add testFile.html
```

Let us go ahead and commit this file:

```
git commit
```

This should open a window in your chosen editor, allowing you to type in a comment to go with the commit. Type in a comment, save and close the window. Git will get back to you with an somewhat informative message. Try `git status` again.

Let us go back and edit our file:

```
<!DOCTYPE html>
<html>
   <body>
      <p>Goodbye cruel world!</p>
   </body>
</html>
```

Try `git status` again. It will now tell you that there is a modified file, but that it is not staged for commit. To get more information, we can use :

```
git diff
```

Try that now. It should show you that there is one line removed, and one line inserted. Go ahead and do an "git add" to stage that change.

It is typical that you might want to review the changes that you are about to commit before you do so. you can do this as follows:

```
git diff --staged
```

Now do a "commit". If you want to skip the "message in the editor" part, you can instead do this:

```
git commit -m "Hello␣becomes␣goodbye"
```

Congratulations, you have now made your first two commits! We will see a bit later how to view previous commits and other cool stuff.

And these are the basics of the commit system. Here are some other useful commands:

```
git reset HEAD filename     # unstages the filename. Useful if you did not mean to add that fi
git commit --amend          # amends the previous commit instead of creating a new one
git checkout -- filename    # discards modifications to the filename, restoring it to its prev
```

Managing commits and their history

All these commits wouldn't do us any good if we couldn't go back and interact with previous commits. Let's see how we can do that. The basic command is:

```
git log
```

try it out now. You should see two commits, each with a hash number next to the word "commit". This hash number is in fact the commit's "name", it's how the system refers to that commit. It also shows you the name of the committer, as well as the message.

Some times you want a shorter view:

```
git log --pretty=oneline
```

There are lots of other options. Note that these "hashlike" commit names may look weird, but are in many ways awesome. For now, know that it is usually enough to copy/paste just the first 6 numbers/letters, they are usually enough to uniquely determine the commit.

Here are some other useful versions of `git log`:

```
git log -p -2
git log --stat
```

The first one will show you, along with each commit, a diff of the changes that commit introduced. The `-2` limits it to the last two commits. The second will show you some statistics of each commit.