Version Control and git

Relevant Links

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

Notes

General information about Version Control and git

- Version Control allows you to maintain copies of your code at various stages during its evolution.
- Here are some things it allows you to do:
  - quickly revert back to a previous, perhaps more stable, version of your code
  - see the differences between where you code is now and where it was at a previous time
  - find out who in your team was the last person to change a line of code, and what happened in that change
  - seamlessly merge together changes implemented at the same time by different people
  - share changes to someone's code, that they can decide to use or not
  - see a history of what has happened when in your code
- Your code history is stored in a **repository**
- A specific version of your code is called a **revision**
- Creating a new revision based on the current state of your code is called a **commit**
- We will be using "git", which is known as a **distributed version control system**
  - Essentially this means that each person holds in their local repository the entire history of the project
  - Git achieves this very efficiently
  - Git was developed in order to maintain the Linux operating system kernel
  - DVCSs allow nonlinear development. You can work on your computer moving along and making commits, and at some later point try to sync those commits with a backup repository that others may have been using as well.

[1] http://git-scm.com/book/en/v2
[2] http://git-scm.com/docs
[3] http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

- When you commit, git takes a "snapshot" of how your entire project filesystem looks, and saves that snapshot.
- Git uses a lot of checksums. Every file and snapshot is captured by its checksum and referred to that way.
- When working with Git there are 3 main areas (see here[4])

  repository where all previous revisions are stored. This is actually a hidden folder named ".git", and nothing more.

  working directory This is your current directory the way you have it set up, with all its files and things.

  staging area This sits between the repository and the working directory. When you decide that you want to keep some of the changes you made to your repository, you will "stage" them, and then they become part of the staging area. When you are certain that you have staged all the changes that "are logically together", you can "commit" them, which moves these staged changes into the repository.

- Your files can be in 4 different states[5]:

  committed these are files whose current version is already in the repository.

  staged these files have some modifications to them, and we have already added them to the staging area.

  modified these files have some modifications to them, but have not been added to the staging area.

  untracked these files are in the directory that is being tracked by the repository, but they themselves have never been added to the repository.

Setting up git in your system

- We have already installed git for you. You do need to customize it a bit however.
- First, you want to set a name and email address for yourself. These are used every time you make a commit, to identify who made the commit. You can do this with the lines:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

- Next you will want to set your editor, so that git knows what program to use. This would be done with a line like:

```
git config --global core.editor "/usr/bin/subl -n -w"
```

  of course the path will be different depending on what editor you want to use; the above line is for sublime text. For vi you would do instead `"/usr/bin/vi"`, for emacs you would do `"/usr/bin/emacs"`

[4] http://git-scm.com/book/en/v2/Getting-Started-Git-Basics#The-Three-States
[5] http://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

- You can verify your settings with `git config --list`.