Git workflow/cheatsheet

The basics

- Start a new project in a new directory with sensible name for that directory.
- Write some code.
- Create a git repository in that directory.

```
> pwd
> /home/user/myCodeProjects/logicallyNamedProjectFolder/
> ls
> newFile1.py  newFile2.py
> git init
```

- Add files to be staged

```
> git add newFile1.py
```

- You can check what files are ready for a commit, which tracked files have been modified, and which files aren't being tracked with a git status

```
> git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#        modified:   newFile1.py
#
# Changes not staged for commit:
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#        newFile2.py
                              .........................
```

- Commit staged file with a useful comment.

```
> git commit -m 'first commit of newFile2.py'
```

NOTE:

You should make these comments as detailed as possible when you are making changes to the code. If you are making changes to different files that are not related, save them in different commits.

Info to give in commit message:
- Names of files being committed.
- Reasons for changes.
- Specific technical details of the change.

Backing up to a remote repository

- Open account on Bitbucket or gitHub.
- Bitbucket gives you unlimited free private repositories and gives you easy step by step instructions of how to use it so I'm not going to put them all here. It was so easy to set up, even I could do it!
- GiHub is mostly for open access code, you have to pay for private repositories!

NOTE:

This is one of the best ways to back up your code - it's offsite, you can clone the repositories onto different machines making it really easy to transport your projects and maintain them on more than one machine.

<u>JUST DO IT!</u>

Some of the clever stuff - branches

- When you have a version of your coding project that works, keep that version as your master branch and make changes in new branches.

```
› git checkout –b testingMod1
Switched to a new branch ‘testingMod1’
```

- The -b means you’re creating a new branch.
- If you’ve only committed changes to your branch, have tested them and are happy for them to be added to your master branch, a merge is very simple.

```
› git checkout master
› git merge testingMod1
```

- Using checkout to switch back to the master branch.
- If you have also made changes to master in the meantime it is possible that there are merge conflicts, but that only happens if you have modified the same section of code in each draft. Checkout the documentation for how to deal with this - it’s not difficult.
- Git is set up to handle most mergers elegantly.

Some more of the clever stuff - getting back to versions of your code that actually worked!

- You have been working on a file but you realise you don't want to keep the changes.
- Use git checkout to replace the file with the one in the previous commit.

```
> git checkout -- filename
```

- OR, you want to recover a file from a few commits back.
- Find the big long alphanumeric string of the commit from git log.

```
>  git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Emma Curtis Lake <email@gee-mail.com>
Date:   Tues Mar 17 21:52:11 2010 -1100

   changed axis labels in newFile1.py

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Emma Curtis Lake <email@gee-mail.com>
Date:   Mon Mar 15 16:40:33 2010 -1100

   Added plot as output to newFile1.py

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Emma Curtis Lake <email@gee-mail.com>
Date:   Mon Mar 15 10:31:28 2010 -1000

   first commit of newFile1.py
> git checkout a11bef06a3f659402fe7563abf99ad00de2209e6
filename
```