**Basic setup**

If you’d like, you can go ahead and save your git username and email so that you won’t have to enter them in again for future git commands.

git config --global user.name "User Name"

git config --global user.email "email"

## Colours

A neat trick that some people often miss is that you can enable some extra colouring to git, so that you can read the output of the commands easier!

git config --global color.ui true

## Initialising git

Now we can start versioning our project. Go ahead and navigate to the directory you want to setup version control for in the terminal using the standard “cd” command. Now you can initialise a git repository like this:

git init

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.

## Adding and committing

To start version-controlling existing files you should start by tracking those files and do an initial commit. To accomplish that, you start by adding the files to git that you would like to be attached to your git project.

git add <file>  
git commit -m 'first commit'

### Adding files one by one  
git add filename  
  
### Adding all files in the current directory  
git add -A  
  
### Adding all files changes in the current directory  
git add .  
  
### Choosing what changes to add (this will got through all your ### changes and you can 'Y' or 'N' the changes)  
git add -p

The working directory is simply, your current local directory that you are working on. e.g if you have **master, dev and yourname-dev** as your remote branches, if you **checkout** from **dev** to **yourname-dev**, **yourname-dev** is now your working directory if you checkout from this (**yourname-dev**) working directory to another say **dev**, **dev** is now your new working directory.

## Branching : Here’s all of the things you need to create and work on a branch:

## Create a local branch to work on  
git checkout -b branchname  
  
### Switching between 2 branches   
git checkout branch\_1  
git checkout branch\_2

### Pushing your new local branch to remote as backup  
git push -u origin branch\_2  
  
### Deleting a local branch - this won't let you delete a branch ### that hasn't been merged yet  
git branch -d branch\_2  
  
### Deleting a local branch - this WILL delete a branch even if it ### hasn't been merged yet!  
git branch -D branch\_2  
  
### Viewing all current branches for the repository, including both ### local and remote branches. Great to see if you already have a ### branch for a particular feature addition, especially on bigger ### projects  
git branch -a  
  
### Viewing all branches that have been merged into your current ### branch, including local and remote. Great for seeing where all ### your code has come from!  
git branch -a --merged  
  
### Viewing all branches that haven't been merged into your current ### branch, including local and remote  
git branch -a --no-merged  
  
### Viewing all local branches  
git branch  
  
### Viewing all remote branches  
git branch -r  
  
# Rebase master branch into local branch  
$ git rebase origin/master

# Pushing local branch after rebasing master into local branch  
$ git push origin +branchname

## Remote backup

Great! You’ve now started versioning your GitHub project **locally**. If you would like to save and backup your project **remotely,**you’ll need to create a remote repository on GitHub (it’s free!). So first head on over to [github.com](https://github.com/) and create a repository. Then, use the link of the repository to add it as the origin of your local git project i.e where that code will be stored.

**Git** terms, a "**checkout**" is the act of switching between different versions of a target entity. The **git checkout** command operates upon three distinct entities: files, commits, and branches.

### General example  
git remote add origin \  
<https://github.com/user/repo.git> ### An example with a repository of mine  
git remote add origin \  
<https://github.com/GeorgeSeif/Semantic-Segmentation-Suite>.git

Then you can go ahead and push your code to GitHub… viola! You’ve backed up your code!

git push origin master

