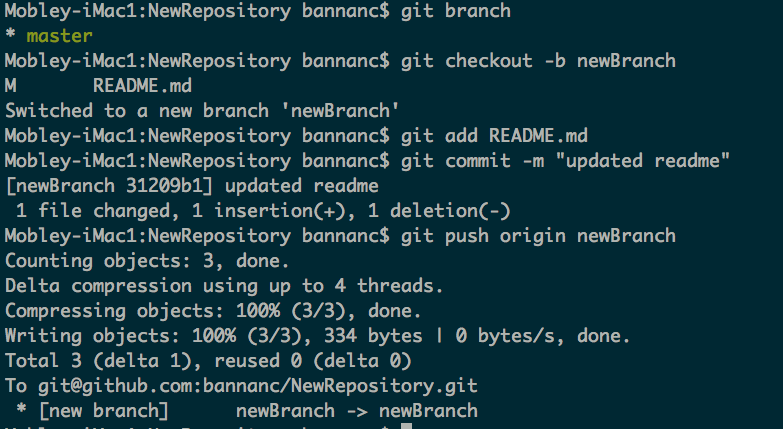
# Branching

Branching is one way you can make and save changes without disrupting the overall project. On a group project the “master” branch is often protected so people can’t make changes directly to it. This is the most basic way to avoid clashing edits on a shared repository

Even if it’s not a shared project, I suggest different branches for your different machines (If you’re going to work at home on a laptop) so that if you forget to push changes at work or home you don’t have conflicting master branches.

* Here is how to do this from the command line:
  + See current branches “git branch"
  + move to another branch “git checkout [branch name]"
  + create a new branch “git checkout -b [branch name]”
  + now you should make changes on your branch and when you’re ready to add those changes push them to your branch instead of master.
    - Remember to first add and commit changes then:
    - “git push origin [branch name]”
* Here’s is an example:



* Now instead of pushing to the master branch, when you’re ready to add your changes to the master branch you should go to the github website and create a [**pull request**](https://docs.google.com/document/d/1L3kE17JCd0TeBhmoE7qSsfohiG1C7A8-X0Yi7PiYFAs/edit?usp=sharing)
  + This requests that the master branch pull your changes

# Updating Your Remote Branch

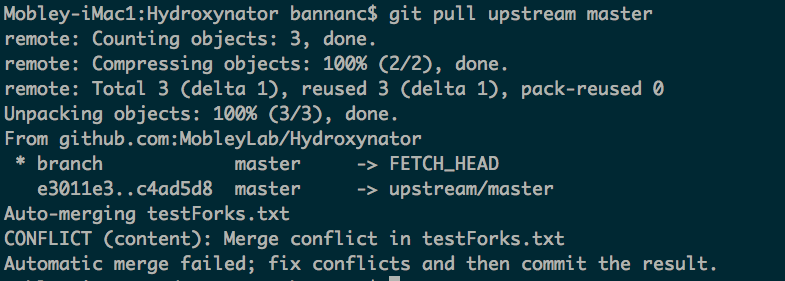
* Add your script to your directory (copy from where you were writing before for example)
* Before you make any changes on your local (desktop) repository, you should pull from the the remote repository (the one on github) with “git pull origin master"
  + Note if there are conflicts with your local repository this will raise an error
* Use “git status” to see how your local changes compare with the remote repository
* To get your file ready to add type “git add [file name]"
  + You can also do “git add \*" to add everything that was changed in this directory
  + or “git add -v \*” which will show everything that is being added
* Commit your changes with “git commit -m [type an explanation of your changes here]"
  + If you forget the -m part it will ask for a comment
  + Use brief, informative comments that explain what you changed
  + This stores your changes on your local copy of the repository, but makes no changes on github yet
* Push your changes to your remote repository (on github) with “git push origin master"
  + This will push to the master branch of the originating repository (origin) on github
  + Differences for forks and branches are explained below
  + Generally, DO NOT push to the master branch on a project which is shared
* If you want to move or remove files already added to the repository you have to use git in front of your command for example:
  + git mv [original file name] [new file name]
  + git rm [file name]
  + Using normal linux mv or rm commands will not have any affect aside from changing your local copies of the files

# Update your Local Repository

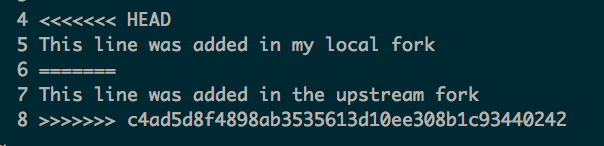
Below are options for how to get changes from your remote directory (the one on github) to your computer. The most common scenario for this is a collaborator has merged a pull request and you need to move those changes into your local repository before you can merge your own pull request.

## Pulling:

* Pulling (as distinct from a "pull request") is used to get changes from a remote repository onto your computer and into the branch you currently have checked out
* If you have no changes in your local fork then you can simply pull from the upstream fork with “git pull [fork name] [branch name]”
  + for example here “git pull origin master”
  + Note, origin is the default github name for the fork you cloned from
* If you have changes in your local fork then pulling may not be ideal; it can result in this kind of output:



* When I open the conflicting file, this is what the conflicting lines look like:

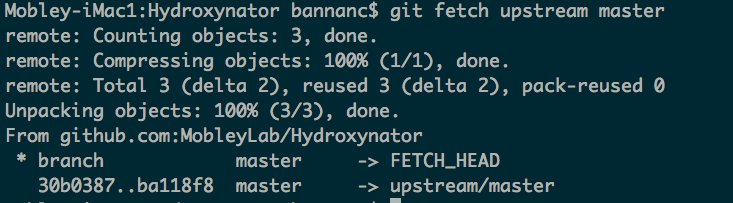
 the === marks the difference between the two files

* You have to resolve these differences before you can do any other merging or pulling.
  + Resolution can be done manually such as in a text editor
  + There are also editors designed specifically for dealing with these types of conflicts (difftool and others) which can be particularly helpful if the conflicts are complicated rather than just content that needs to be moved from one place to another

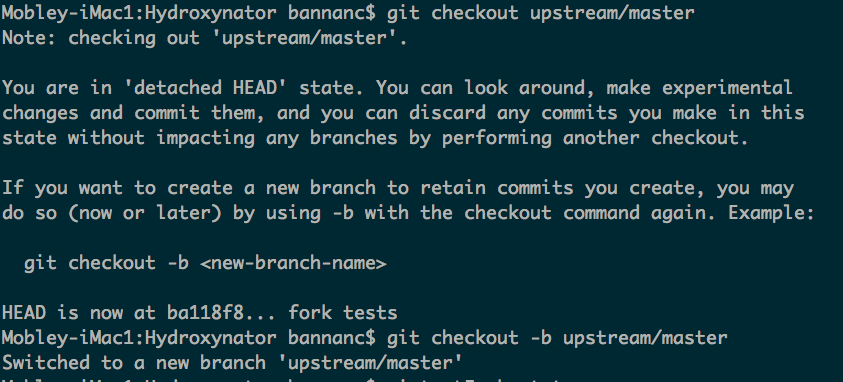
## Fetching

Note that these examples were written for a repository with multiple forks. If your repository only has one fork then in most of these cases you’ll want to use origin instead of upstream

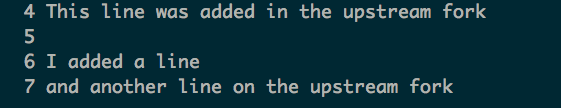
* + Fetching, as opposed to pulling, allows for retrieving changes from a remote repository without actually bringing them into your current branch yet
  + you can fetch changes with "git fetch [fork name] [branch name]” (for a branch) or "git fetch [fork name]" for the full set of branches



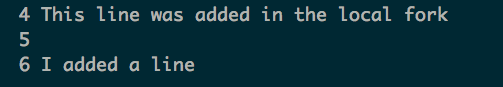
* + You can see above that there is a new branch called upstream master, it isn’t visible in “git branch”, but fetch gives you temporary access to look at it.
    - Also your current master is stored in a temporary branch “FETCH\_HEAD"
  + You can see the current changes in upstream master by:
    - “git checkout upstream/master”



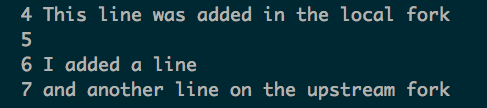
* + - * Note - you don’t want to make it a new branch (with -b) because it will be ambiguous if you fetch again later and want to check out the fetched branch.
* You can look at files that may have changed
  + example lines:



* You can do the same thing with your current local master my looking the master branch:
  + “git checkout master”
  + example lines in the changed file:



* When you are ready you can merge these changes:
  + “git merge upstream/master” (this is for a repository which is a fork)
  + Here are the lines in the merged file



# Generally Helpful Commands

* git status - tells you what the state of your local directory is relative to the remote branch
* git branch - gives a list of branches for this repository
* git remote -v - gives a list of remote forks and their names (you can point to other people’s forks of a project, but it’s not necessarily recommended)
* git checkout origin/[branch name] [filename] - reverts the [filename] to the version in origin/[branch name]
  + For example, you can use this to revert changes of a committed file you could call "git checkout origin/master filename