Git Tutorial

Ali Farhat, Shakthi Visagan, Adam Weiner

Preliminaries on BASH.

Some simple BASH, or **B**ourne **a**gain **sh**ell, commands before we begin (only for Linux shells):

Format	*Example*	*Definition*
`pwd`		**p**rint **w**orking **d**irectory
`ls`		**l**i**s**t files in working directory
`mkdir <new_folder_name>`	`mkdir git_tutorial`	**m**a**k**e a new **dir**ectory
`cd <folder_location>`	`cd git_tutorial`	**c**hange **d**irectories
`cp <src_fl> <trmnl_fl>`	`cp ali.pub ali.txt`	**c**o**p**y a file to another location
`touch <new_filename>`	`touch ali.txt`	create a new empty file
`echo"text" >> <file>`	`echo "<3 ALI!" >> ali.txt`	append text to the end of a file

Example shell.

Aretha Shell

`shakthi@aretha:~$ pwd`
`/home/shakthi`
`shakthi@aretha:~$ ls`
`Documents Spongebob_Memes Ali_Christmas_WILD_Party_Pic.png`
`shakthi@aretha:~$ mkdir git_tutorial`
`shakthi@aretha:~$ ls`
`Documents Spongebob_Memes Ali_Christmas_WILD_Party_Pic.png git_tutorial`
`shakthi@aretha:~$ cd git_tutorial`
`shakthi@aretha:~/git_tutorial$ pwd`
`/home/shakthi/git_tutorial`

Notes.

`~` This symbol almost always implies your home directory. The shell will remind you of your current location in Aretha using the space listed before the prompt, `$`. When using `ls`, folders will appear in blue while files will appear in white. You can also use `pwd` to remind you of your current location. More advanced commands and shell scripting will come in a later tutorial. It is good practice to avoid using strange characters and spaces in your folder names as it is easier to `cd` into them.

Preliminaries on SSH Keys and RSA Authentication.

SSH, or **s**ecure **sh**ell, is an encrypted protocol used to administer and communicate with servers, especially Linux servers. If this is your first time pushing code to Github using Aretha and Jenkins, then this will be useful for you. Read the tutorial from Github if you are unfamiliar with this process. Some of the steps will be reproduced here with less explanation. Most of this should be set up already, however.

First check if you have any existing SSH keys.

Format
`ls -al ~/.ssh`

Check the directory listing to see if you already have a public SSH key. If you see an existing public and private key pair listed (for example `id_rsa.pub` and `id_rsa`) that you would like to use to connect to GitHub, you can add your SSH key to the ssh-agent.

If you don't have an existing public and private key pair, or don't wish to use any that are available to connect to GitHub, then generate a new SSH key. If you receive an error that `~/.ssh` doesn't exist, don't worry! We'll create it when we generate a new SSH key.

Format
`ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`

When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location. At the next step, it is optional to add a passphrase, so feel to just press Enter to pass through it.

Add your SSH private key to the ssh-agent. :

Format
`ssh-add ~/.ssh/id_rsa`

To configure your GitHub account to use your new (or existing) SSH key, you'll also need to add it to your GitHub account. Follow the instructions here to do so. Aretha doesn't have `clip` installed so you'll have to find creative ways to copy the contents of the `id_rsa.pub` file to your clipboard. For example, using the BASH command **c**o**p**y to create a new file that you can access through text can be one way:

Format
`cp id_rsa.pub id_rsa_read.txt`

Introduction.

For future reference, this is a good resource. But for now, we will try our own cases.

We begin by cloning a repository from github by typing in the following in our JupyterLab terminal:

Format
`git clone git://host.xz[:port]/path/to/repo.git/`
Example
`git clone git@github.com:meyer-lab/tutorial-notebooks.git`

Next, it is always a good habit to type in the following command in your terminal:

Format
`git status`

That should show you that you are on branch `master`. We need to branch off from master to create a private branch for each one of us:

Format	***Example***
`git checkout -b <new_branch_name>`	`git checkout -b Ali_progress`

… or whatever you desire to name your branch. If you'd like to check out a pre-existing branch, that is, checkout a branch already created by you or someone else:

Format	***Example***
`git checkout <branch>`	`git checkout master`

Type in `git status` to make sure it worked if you want to, but a confirmation will be shown. Use `git checkout Ali_progress` to return back to the branch we were working on.

Format
`git push --set-upstream origin <new_branch_name>`
Example
`git push --set-upstream origin Ali_progress`

Using Git to track changes in code.

The next step is to make some changes on the branch and try to push it. Make sure that you are ***NOT*** on branch `master` before you make changes to the code. You can check this with `git status` and then use `git checkout <branch>` to checkout the branch you want to work on.

So, go into the file [inser_file_here] and make an edit, that is, write some code, text, etc.:

Format	***Example***
`git add <filename>`	`git add [inser_file_here]`

You can also use `git add -A` to track all the changes made by you on files tracked by git:

Example
`git add -A`

Then, add it to your commit:

Format	***Example***
`git commit -m <"reason for commit">`	`git commit -m "my first commit"`

Alternatively, you can stage all changes and commit in one line:

Example
`git commit -a -m "my first commit"`

Then push your changes to your local branch.

Format
`git push`

Note that the push commit won't work exactly like that if it is your first time pushing from that branch. So just type in the command recommended in front of you on the terminal.

Now, let’s move to Github and try out a Pull Request…
By now, we have merged the changes you made on your branches with master. Now, let’s switch back to master and pull these changes using:

Format
`git pull`

`git pull` is just pulling whatever is on the github server onto your local account. A "pull request" merges changes from your current branch to the master branch (or some other specified branch).

We can also use this pull on our personal branches if several people are working on the same branch…

Now that we have played around with the simple cases, let’s try and do some edge cases that you might encounter.
First, instead of branching off from master, let’s commit directly to it. Usually, people forget to switch onto a new branch. So it would be good for us to see how we might deal with that.

-Do you not care about these commits? Delete the using:

Format
`git reset --hard origin/master`

Note about the above line. This will reset everything with what is on master.... If you're on branch "example" and want to reset your local to whatever is the most recent version of that branch on github then you you'd replace "master" with "example" in the command.

-Do you need these commits? Then, first, you would need to switch to a new branch from the master you committed to, push those changes for that branch. Then switch back to master using `git checkout master`.
Note you can only switch between branches after you have committed changes on your local branch. Then, to delete those changes from master, you can delete those changes using `git reset --hard origin/master` or:

Format
`git pull --rebase origin/master`

-A hardcore way would be to download the file on your local desktop, then revert the commits by removing all these changes or by going back in time (removes commits to master) using
`git reset HEAD~12 --hard` Then, pull master, make a new branch, bring back the file from your desktop and act as if its a commit to be made on your new branch.