Commands to know:

* pwd- prints out the current directory- print working directory
* mkdir- make directory
* cd change directory- syntax cd {directory name}
  + Cd .. will go back one directory
  + cd ../.. Will go back 2 directories
* touch creates a file- syntax touch {file\_name}
* ls- list files in working directory
  + ls directory will list the files in a directory without making it working directory
* cat will output the contents of a file to the screen- syntax cat {file\_name}
* rm will delete a file- syntax rm {file\_name}
  + Rm -R {directory} will delete an entire directory and any sub directories
* | pipe used to pass the output of one command on a file/directory to another command
  + command {file\_name} | command
  + command {file\_name} | command > {new\_file}
  + i.e. sort newyork.txt | uniq > {file\_to\_be\_written\_to}
    - Check the room\_5 repo on github and run this command and you will see Bronx, Brookly, Queens, Manhattan, Staten Island and Long Island. If you cat newyork.txt you will see a repeating list of those names
* > writes to a file- syntax echo ‘some\_text’ > {file\_name}
  + Whatever the text is will be written to file\_name
* >> appends- syntax echo ‘some\_text’ >> {file\_name}
  + Whatever the text is will be written {file\_name}
    - Difference between > and >> is > will overwrite a file and >> will add to the end of the file without overwriting the contents of the file
* sort will alphabetize a file by lines not by words- syntax sort {file\_name}
  + Cat the english.txt file and see the order of the lines, then sort the file and see how it works. Then cat the newyork.txt file and then sort the newyork.txt file and see the results.
* unique- will remove duplicates that are adjacent to each other only
  + Cat the newyork.txt file and notice how many duplicates are next to each other but are then separated by other boroughs. Now do uniq newyork.txt and notice that the duplicates that were next to each other are gone. But the list still has duplicates. Now sort newyork.txt and see how the whole list has been put into order and all the duplicates are next to each other, but the list is still long. Now do sort newyork.txt | uniq and see the list has been broken down to only 6 unique results
* Sed- stream editor- searches a file for a pattern and will replace it with another pattern.
  + sed ‘s/pattern\_to\_look\_for/patter\_to\_replace\_with/g’ {file\_name}
    - The s means substitute and the g at the ends means global, meaning that it will replace all occurrences, without it it will only replace the first occurrence.
    - You can also use the optional -i to overwrite the file. i.e.
      * sed -i ‘s/pattern\_to\_look\_for/patter\_to\_replace\_with/g’ {file\_name}
    - Try these examples to get a hang of it. Type:
      * sed ‘s/Bronx/Houston/’ newyork.txt
        + It will put out a list of newyork.txt And the first occurrence of Bronx will read Houston. The rest will remain Bronx
      * sed ‘s/Bronx/Houston/g’ newyork.txt
        + Now it will produce the list but all Bronx will now show Houston
      * cat boroughs.txt and notice that it is a copy of newyork.txt now do:
        + sed -i ‘s/Bronx/Houston/g’ boroughs.txt NOTICE THE CHANGE IN FILE NAME

It will not output to the screen so do:

cat boroughs.txt

Now the file has Houston instead of Bronx, the file has been overwritten

Be careful of using -i if you don’t want to overwrite a file, if you want to save the info use >

sed ‘s/Bronx/Houston/g’ newyork.txt > nyc.txt

Now cat nyc.txt and see that all Bronx are changed to Houston but in a different file, cat newyork.txt and see how it is unaffected.

You can also change then sort and uniq and write to a new file with |

* + - * + sed ‘s/Bronx/Houston/g’ newyork.txt | sort | uniq > nyc.txt

cat nyc.txt and see that you have Brooklyn Houston Long Island Manhattan Queens Staten Island

* grep- global regular expression print- will find a pattern and report on the occurrences
  + grep ‘pattern\_to\_look\_for’ {file\_name or directory}
    - You have options
      * -R (recursive) is used to search directories
      * -i (insensitive) makes the search case insensitive
        + Note this is different from sed where -i overwrites a file
      * -l (list) will return just the file name with an occurrence
    - Examples
      * grep ‘Bronx’ newyork.txt
        + It will print to screen Bronx 11 times
      * grep -l ‘Bronx’ newyork.txt
        + Will print out newyork.txt because there is an occurrence

This is like asking if there is ‘Bronx’ in the newyork.txt file and it returning true

grep -l ‘Bronx’ english.txt

Will not return anything because ‘Bronx’ is not in that file

* + - * grep ‘bronx’ newyork.txt
        + Will not return anything because ‘bronx’ is lowercase and in the file it is uppercase ‘Bronx’
      * grep -i ‘bronx’ newyork.txt
        + Will print ‘Bronx’ 11 times because it looked for bronx regardless of case
      * Move out of that directory with cd ..
      * grep -R ‘Bronx’ northamerica
        + Will print out 14 times in a format such as:

northamerica/french.txt:Bronx

northamerica/newjersey.txt:Bronx

northamerica/newyork.txt:Bronx

I added ‘Bronx’ to other directories. The -R option extended the search to all files in that directory.

* + - * While still in the continent directory
      * grep -Rl ‘Bronx’ northamerica
        + Will print out:

northamerica/french.txt

northamerica/newjersey.txt

northamerica/newyork.txt

northamerica/texas.txt

Instead of every occurrence the l just tells you what files. The -R will look at each file in the directory

* + - * grep -Ril ‘Bronx’ northamerica
        + Will print out

northamerica/english.txt

northamerica/french.txt

northamerica/newjersey.txt

northamerica/newyork.txt

northamerica/texas.txt

This is because there a bronx in english.txt which was counted now because of the -i

* + - * grep -Ril ‘Bronx’ northamerica > northamerica/boogie\_down.txt
        + This will take the results of the grep command and write it to a new file boogie\_down.txt that will be found in the northamerica directory

Shortcuts

* ~ = home file echo ~ will print your home directory
* . = current directory or hidden file
* .. = parent directory

command line

absolute path= /home/user/userName/long/way

relative path = if you are already in the userName directory you just need long/way

nano= git bash file editor

arguments and parameters

navigating file system= the use of absolute path/relative path cd {directory\_name} cd .. cd ../..

remote repository= when working with github the repo on github is the remote repo and the clone of that repo on your computer is the local repo

What is ~/.bash\_profile? This is a file that is stored in your home directory that contains information that is loaded every time you open git bash

* i.e . alias or export ps1=’>>’ which will appear on each line
  + Remember to source ~/.bash\_profile to properly update the system with the changes to the bash\_profile

What is Github?

* An online remote version control system

Git commands

* Init
  + Used to create an empty repo
* Add
  + Used to added files to a staging area in a git repo
    - Syntax git add {file\_name}
    - Can also use git add .
      * That will add all files that have been changed
* Commit
  + This takes the files in the staged area (those files that have been added) and commit the changes to the git log
* Pull
  + This will fetch AND download content from a remote repo and immediately update the local repo
* Push
  + This will take the changes you committed and PUSH them up to the remote repo
* Checkout
  + Used change between branches
    - If you have two branches, main and test\_branch and you are on the main branch(which can be checked by git branch) git checkout test\_branch will switch from main to test\_branch
  + Can be used to revert back to a previous HEAD
    - A HEAD is formed when a commit is created. The HEAD is the furthest save point of a branch
* reset
  + This is a way to reset to a previous HEAD
  + Can be used to remove a file from the staging area
    - Git reset HEAD {file\_name}
  + Can be used to reset to a previous commit using SHA
    - Git reset {first 7 numbers of the SHA}
      * Once you reset to that SHA the changes that were made since that point will remain, all the commits will be erased, in order to erase the changes to the file you will have to:
        + Git checkout HEAD

This will revert back to the most recent HEAD, since you erased all the other commits the HEAD is done before all the changes.

* Branch
  + A way to separate from the main repo without making system wide changes.
    - Example
      * All the files for a website are on the main branch, you have to change some pictures and text. You make a branch for you to work on the html and css files without disrupting the website. After you have made the changes, added them to the staging area, and committed them, you can push them to the main branch, then if no merge conflicts, can merge it to the main branch allowing the changes you made on your branch, then pushed and merged to the main, to take place instantly on the website.
* Clone
  + This is how you copy all the files of a remote repo to your remote/local repo
    - Example
      * You are working with a several other students on a website. One person has set up a repo on github with some useful files such as images, sounds and video. You can fork that repo so you have a copy of the files on your github account. This will be a branch off the main. You can then clone your fork, this will copy the files from your remote repo to your computer and create a link between the two so files that are created and worked on can be pushed back to your remote (github) repo to then create a pull request to have it merged with the main branch
* Diff
  + This is a way to see the difference between the HEAD version and the current version of a file in a repo
    - Cat diff.txt
      * You will see that there is nothing in the file
        + echo ‘See the difference’ >> diff.txt

cat diff.txt

You will see the statement ‘See the difference’

git diff diff.txt

Will print +see the difference

That is the difference between the HEAD version of diff.txt which had nothing in it and the current working file of diff.txt which has the line ‘See the difference’