**Terminal Notes**

* **Acronyms**
  + GUI - Graphical User Interface
  + CLI - Command Line Interface
* **Notes**
  + Unix is a family of operating systems. Includes:
    - Linux
    - Android
    - iOS and macOS
  + Microsoft Windows is not part of Unix
  + Typical Elements of a command
    - Ex: [projects]$ rm -f foo.txt
    - Prompt
      * Automatically supplied by the terminal, do not type it
      * Typically ends with $
    - Command
    - Option
    - Argument
  + Shell – the window in Terminal
  + Use up/down arrows to find previous commands
  + Some options are compacted but can be uncompacted and re-ordered
    - ex: [$ ls -rtl] = [$ ls -r -t -l] = [$ ls -trl]
  + Files starting with a period (.) are hidden
  + Operations can be combined
    - ex: $ head sonnets.txt > sonnets\_head.txt
    - This will output the first 10 lines of sonnets.txt into sonnets\_head.txt
  + Superusers with special powers are known as root
  + Contents in ~/.bash\_profile will execute whenever terminal is opened
  + Storing an alias in ~/.bash\_profile will make it available in terminal
  + Element Variables are settings in the current shell?
* **Git / GitHub**
  + Invented 2005 by Linus Torvalds (guy who created Linux), a free, open source, distributed tool for collaboration
    - Distributed means decentralized – no single authority, everyone has a copy of everything, if GitHub crashes, the data is still on machines
      * Distributed often has Version Control issues
    - Centralized is like a library, anyone can check a file out, but it has to be checked back in, and if the library burns down everything is lost
    - Comes with Version Control (database of versions) and Backup Control (able to download old versions even if not on your computer)
  + Git Workflow
    - Create and edit files
    - Add the files to the staging area
    - Commit the staging area



* + GitHub Workflow:
    - Create a new Repository
      * Ensure directory is initialized (git init)
      * Ensure commit status is clear (git status)
      * Go to GitHub and:
        + Create a new repository
        + Follow terminal commands for existing files on computer
    - Work on a new branch
      * Create an issue (GitHub GUI)
      * Checkout a new branch ( ‘git checkout -b nameTheBranchHere’ )
      * Make your changes on your computer
      * Add, Commit
      * Push the branch to GitHub ( ‘git push -u origin nameTheBranchHere’ )
      * Find your branch on GitHub
      * Select ‘New Pull Request’
        + Add text
        + ‘closes #issueNumber’ - in a Pull Request, typing this in the ‘write’ section will auto-close issue number ‘issuesNumber’ if the Pull Request is approved
  + Git Commands – available on Terminal (commands preface with ‘git’)
    - **add** <filePath> – add a file to be staged for commit
      * Use wildcards to add multiple files, ex: git add ‘\*.html’
    - **commit** -m “message” – commits all staged files with a short descriptive message
      * Also ‘git commit folderName -m “message” ‘ will commit everything in folderName without having to first do the ‘add’ command
    - **checkout** (option) <path>
      * ‘git checkout <branch>’ - lets you switch between branches
      * ‘git checkout <commit#> - lets you look at a previous commit without affecting HEAD
        + Use ‘git checkout master’ to get back to real-time
      * To grab a file from a previous commit, do this: ‘git checkout <commit#> <fileName>’
      * ‘ -b ‘ checks out a new branch, omit -b if it’s an existing branch
    - **clone** <url> - download the repository from GitHub at the specified URL
    - **config** – change config info
      * git config --global user.name “Your Name”
      * git config --global user.email [yourEmail@example.com](mailto:yourEmail@example.com)
      * To fix the identity of a file after it has been committed, use:
        + git commit --amend --reset-author --no-edit
    - **diff** - shows visually the changes of a file
    - **fetch** - checks GitHub if your origin matches the online master(need to ‘git status’ after for info about origin/master)
      * fetch --all - also checks all branches
    - **git** – displays a menu of available commands with a brief description
    - **help** <command> – similar to ‘man’ for bash, displays info about <command>
    - **init** – initializes the current directory to be tracked by Git
    - **log** - outputs a list of all Git commits currently being tracked
    - **pull** origin master - downloads the ‘master’ branch to the origin (your computer)
    - **push** (how, the option) (where) (what)
      * push -u origin master - pushes the origin files upstream (-u) to the branch ‘master’ the first time
        + Must be in a clear state to push (no staged files?)
        + Use the first syntax, without ‘-u’, to upload a revision
      * push -u origin branchPathName - pushes the files upstream (-u) to the branch ‘branchPathName’
    - **remote** -v - displays all currently active remote connections
    - **reset** (option) <commit#> - sends you back to an old version
      * option --hard - deletes all files/changes created after the commit
      * option --soft - takes you back to the staging area after the commit (but doesn’t delete the changes)
      * reset HEAD <filePath> - same thing as ‘rm --cached' from your perspective, but should be used if prompted by Git instead of ‘rm --cached'
    - **revert** <commit#> - used to add new commits to reverse the effect of some earlier commits - MAKES NO SENSE, SEEMS TO JUST DELETE FILES AND NOT RESET THEM
      * -n flag makes the revert staged for commit instead of auto-commit
    - **rm** <filePath> - remove the staged file AND the original file
      * rm --cached <filePath> – remove a file that is staged for commit
        + Use wildcards to remove multiple files, ex: git rm ‘\*.js’
    - **status** – tells you if the current directory is or is not tracked by Git
      * Alternatively, .git will be in the directory of a Git tracked directory
  + Git Branches
    - Small changes are highly preferred over large changes, as they are more manageable and easy to reverse
    - Manage collaborator settings in GitHub before mass populating with ‘Branch Protection Rules’
    - ‘Pull Request’ are when changes in a branch are being requested to be ‘pulled’ into the master (to become part of the master)
      * Good idea to require pull requests to be reviewed by Code Owners
      * Code Owners are contained in the file CODEOWNERS file
      * Syntax:
        + Own entire repository:

\*@gitHubID

* + - * + Own a particular file:

file/path @gitHubID

* + - Watch out for Collisions - when two people are working in the same branch on top of each other
    - Git Branches do not push to origin master, do this instead: ‘git push -u origin branchPathName’
    - **NEVER DO WORK ON THE MASTER BRANCH!!!!**
* **Bash Scripting**
  + Beg of script file should start with #!/bin/bash
  + Place commonly used scripts in ~/bin/
  + Needs to have the “execute” permission to allow them to run
    - ex for filename “script.sh”: chmod +x script.sh
  + To ensure scripts in ~/bin/ are available to the terminal, you must add the path to the ~/.bash\_profile
    - ex: PATH=~/bin:$PATH
  + Create your own variables with variablename=”string”
  + Reading external data
    - read <variableName> - can be used to extract info from the user
      * ex: echo “Guess a number” ; read number ; echo “You guessed $number”
    - files=<path> - can be used to access external files
      * You can iterate through each file to do something
        + ex, to print: for file in $files

do

echo $file

done

* + - Have the user add input arguments when you run the script
      * Arguments are entered after the script name separated by spaces
        + ex: saycolors red green blue
      * Within the script, arguments are accessed using $1, $2, etc.
      * Arguments are 1 indexed (not 0)
      * $@ syntax is used to accept an indefinite number of input arguments
        + ex using above: for color in “$@”

do

echo $color

done

* + Conditionals in bash scripting
    - Start with if
    - Statements are contained in [ ]
    - End with fi (if backwards, or short for “finish”)
    - Ex: if [ $index -lt 5 ]

then

echo $index

else

echo 5

fi

* + Operators for bash scripts
    - -eq – equal
    - -ne – not equal
    - -le – less than or equal
    - -lt – less than
    - -ge – greater than or equal
    - -gt – greater than
    - -z – is null
  + Operators for comparing strings
    - == - equal
    - != - not equal
    - ex: if [“$foo” == “$bar”]
  + Loops in bash scripts
    - for – condition follows
      * ex: for variable1 in $variable2
    - while – condition goes in brackets
      * ex: while [ $index -lt 5 ]

do echo $index

index=$((index + 1))

done

* + - until – condition goes in brackets
      * ex: until [ $index -eq 5 ]

do echo $index

index=$((index + 1))

done

* **Element Variables**
  + USER – the person owning the session
  + PS1 – style of the command prompt (i.e. the $ line)
  + HOME – path for ~ ???
  + PATH – list of directories that contain scripts
* **Shortcuts**
  + \* - short for wildcard or star, let’s you filter results
    - ex: $ ls \*.txt
  + ~ - same as the home directory
    - ex: ( /Users/DanB/ruby/projects ) = ( ~/ruby/projects )
  + 🡪| - short for tab completion, auto fills the remainder of a file name if only 1 file starts with the same input
  + . – (dot) short for current directory
    - ex: cp ~/text\_files/sonnets.txt . – will copy sonnets.txt to . (current directory)
  + . . – (dot dot) short for one directory up
  + $ - used for returning values of element variables
    - ex: echo $USER
  + command K – clears terminal, also works with command ‘clear’, ^L, Cmnd L
  + command L – clears terminal, also works with command ‘clear’, ^L, Cmnd K
  + command N – new terminal window
  + command T – new terminal tab
  + control (^) A – takes you to the beg of the command line
  + control (^) C – cancel, if this fails use esc
  + control (^) D – ends the terminal session
  + control (^) E – takes you to the end of the command line
  + control (^) L – clear the terminal screen
  + control (^) R – short for reverse i-search, lets you search for a previous command
  + control (^) U – clears to the beginning of the command line (delete)
  + down arrow / space bar – how to see more info when : is present
  + option (click) – allows you to select a specific spot in terminal with mouse
  + q – short for quit, gets you out of man (argument)
* **Commands**
  + > – short for redirect, puts a thing in a thing
    - ex: $ echo “Text for my text file.” > textFile.txt
    - < redirect right to left, ex: $ cat < fileName.txt
  + >> – short for append, adds a thing to the end of a thing
    - ex: $ echo “The next line of text.” > textFile.txt
  + <command> ; <command> - runs two commands back to back
    - always runs even if previous command fails
  + <command> && <command> - runs two commands back to back
    - doesn’t run the next command if the previous fails
  + | - short for pipe, used to transfer an output to another program
    - ex: $ head sonnets.txt | wc
  + ! – short for bang, retrieves previous commands
    - $ !! repeats the previous command
    - $ !echo repeats the last echo command
    - $ !n executes the ‘n’th number command in the command history
  + alias <string1>=”<string2>” – creates a shortcut of <string2> with <string1>
    - useful in ~/.bash\_profile to make globally available in terminal
      * ex: alias desiredName=’./aliasPath.sh’
    - Can also add standard input arguments to the alias
      * ex, if we want “green” to be included as the first input to saycolors:

alias saycolors=’./saycolors.sh “green” ‘

* + atom <filePath> - opens the pathed file in Atom
  + brew <command> – runs specific homebrew commands
    - brew update – updates homebrew
    - brew doctor – checks homebrew directories for correct pathing
  + cat – short for concentrate, displays file contents on the screen
    - ex: $ cat textFile.txt
  + cd <directoryName> - short for change directory, to the destination
    - $ cd ~ - takes you back to the main directory
    - $ cd . . – (c, dot, dot) changes directory to one level up
    - $ cd – (no directory name) takes you back to the main directory
    - $ cd - - (cd dash, only 1) returns you to the previous directory
    - $ cd ../../<location> - takes you up 2 folders, then to a new sub-folder
  + chmod <permissions number> <file name> - changes file modes (permissions)
  + clear (^L) – clear the terminal screen
  + cp <string> <string> - short for copy, will make a copy
    - ex: cp originalFile.txt copiedFile.txt
    - copying a directory WITHOUT trailing slash (/) copies the folder + file
    - copying a directory WITH a trailing slash (/) copies the files only
      * same as $ cp <copyLocation>/\* <desiredLocation>
    - copy multiple files to a location
      * $ cp <filePath1> <filePath2> <destination>
    - copy all files (not directories) of a current path to a new path
      * $ cp \* <destinationLocation>
  + curl <URL> – short for cURL, interact (download) a file from a URL
    - ex: $curl -OL cdn.learnenough.com/sonnets.txt
    - option -L handles redirects appropriately
  + diff – short for difference, displays differences in a file
    - ex: $ diff textFile1.txt textFile2.txt
  + echo <string> – prints the string, use ‘ or “ to wrap text
    - ex: $ echo ‘Hello World’
    - ex2: $ echo | curl www.somedomain.com
  + env – short for environment, prints a list of environmental variables
    - env | grep <elementVariable> - returns values of <elementVariable>
  + exit (^D) – ends the terminal session
  + export <elementVariable>=”<string>” – makes <elementVariable> available to all child sessions with the value <string>
  + files=<path> - allows access to external files
    - ex: files=/some/directory/\*
  + find ? ? – used to find files meeting a search criteria
    - ex: $ find . -name ‘\*.txt’
      * Finds files with .txt in name in the . (current) and sub directories
  + grep <string> <file> – short for “Globally search a Regular Expression and Print”, prints all lines containing the string in the file
    - grep is case-sensitive by default
    - -i – option to ignore case sensitivity
    - grep can also search for words that start/end with certain characters
      * ex: $grep ‘ ro[a-z]\*s ‘ sonnets.txt
      * This will find all words that start with ro and end with s (lowercase)
      * excluding the ‘ space and space ‘ will find any word with ro…s
    - $ grep -r <string> <directoryLocation> - finds the files in <directoryLocation> and sub-directories containing the <string>
    - -l – ignores the results and displays file names only
  + head <string> - views the first 10 lines of a file
  + history | less – displays a list of previous commands in a particular terminal shell
    - history | grep <searchCriteria> - lets you search commands in your history and filter results to your <searchCriteria>
  + kill -<level> <pid> - Kill a process
    - ex: $ kill -15 24601
    - pkill -<level> -f <name> - Can also kill a process matching a name
      * ex: $ pkill -15 -f spring
  + less <string> - viewing mode for a file, enables tools like space, ^B, arrows
    - /<string> lets you search for the string
      * n navigates to the next match
      * N navigates to the previous match
    - ^B to go back a page
    - ^F or spacebar to go forward a page
    - G moves to the end of the file
    - 1G moves to the start of the file
  + ls – short for list, displays files and directories
    - $ ls -l – long form, displays date/time last modified with ls output
    - $ ls -h – short for human readable, displays bytes is kb (use with -l)
    - $ ls -rtl – short for Reversed Time of modification (Long form), same as ls -l but in reverse order
    - $ ls -a – short for all, also displays hidden files and directories
    - $ ls -ld <directoryName> – useful for showing directory info without the contents, -d removes the contents to show directory only
  + nano <fileLocation> - opens <fileLocation> in the nano text editor
  + node <filePath>.js – executes the JavaScript code in the destination file
  + man <commandName> – short for manual, displays info about the command
    - ex: $ man echo
    - use q to quit out of a man
  + mkdir <string> - short for make directory, makes a folder
    - option -p will make subdirectories as required
    - ex: $ mkdir -p ~/foo/bar
  + more <string> - viewing mode, an older/weaker version of less
  + mv <string> <string> - short for move, a way to rename a file
    - WILL override if destination is same as another file
    - ex: $ mv originalFile.txt newFile.txt
    - Can use wild cards to move multiple files
    - ex: $ mv \*.txt text\_files/
    - Move multiple files to a location
      * $ mv <filePath1> <filePath2> <destination>
  + nslookup <domain> – returns the IP address of the argument
  + open <file> - opens its argument using the default program
    - ex: $ open .
      * Opens the . (current) directory in the default program
  + ping <URL> (string) - pings a URL every second (and logs it in the string file)
    - control + C – stops ping from running forever
  + ps aux – short for process status, shows all processes
    - to filter results by program name, pipe the results of ps through grep
    - ex: $ ps aux | grep <programName>
  + pwd – short for print working directory, shows the path of current directory
  + read <variableName> – makes data available from external places (the user)
  + rm <string> - short for remove, deletes a file
    - ex: $ rm fileToDelete.txt
    - $ rm -i – will force a confirmation before deleting
    - $ rm -f – short for force, overrides -i, preventing the “are you sure?” prompt
    - $ use wildcards to delete multiple files with a certain string
      * ex: $ rm -f \*.txt
        + will delete all files containing .txt
    - $ rm -rf <directoryLocation> - short for remove recursive force, removes a directory, it’s files, and any subdirectories (without confirmation)
  + rmdir <string> - short for remove directory, will not work if directory contains files
    - -r flag means ‘recursively’
  + sed ‘s/<string1>/<string2>’ <fileLocation> - stands for stream editor, similar function to ‘find and replace’, replaces string2 for instances of string1 once/line
    - ‘s’ stands for substitution
    - add /g to the end of <string2> to make the command global (changes all instances)
  + sort <fileLocation> - sorts the file and prints the output
  + source <filePath> - makes changes available now (like in .bash\_profile)
  + sudo <command> - short for superuser do, executes the command as root
  + tail <string> - view the last 10 lines of a file
    - tail -f is used to view a file that is actively changing
  + top – shows the processes consuming the most resources
  + touch <string> – creates an empty file with the string name provided
    - ex: $ touch folderName
  + traceroute <domain> - returns the ‘stops’ to get to the argument domain
  + uniq <fileLocation> - filters out duplicate & adjacent lines
    - to filter all duplicates, use $ sort <fileLocation> | uniq
  + wc <string> - short for wordcount, outputs the number of lines, words, and bytes of a file
  + which <program> - checks if a given program is available at the command line
    - ex: $ which curl
  + whoami – prints the user who is currently logged in