Class 6: 8/29/22 (Git)

* Git
  + A “revision control system”. It is very difficult for a group of people to work on a software at the same time.
  + Version control systems keep a history of a project
    - Can see in version control why you changed a version of your code
  + A tool which allows you to keep a history of your software project as a series of “checkpoints”. It also allows you or another party or parties to maintain other versions of the software projects, in which case git provides extensive utilities to merge or otherwise combine or modify multiple different “branches” or “forks” of the software
    - Ie. if you have a brilliant idea towards the end of your project, create a branch for your project (you can eventually merge them)
  + Working area, working copy, working history
  + Github
    - A website that hosts git repositories
    - Github copilot is one of the new language model (train AI model on all the code it has on github)
  + Commands
    - git init
      * initialized empty Git repository in /tmp/example/.git
    - ls – lath
    - git status
      * get status of working area, working copy, working history
    - git add READ.ME
      * add a file
* Review of Docker
  + A container inside your computer
  + Docker is a convenient place to run stuff (ex. grab RStudio environment from somewhere else)
  + Workflow:
    - Create a Docker file
    - Use Linux commands therein to install software and libraries
    - *docker built . -t 611-example*
      * Telling docker to build a docker container called “611-example”
      * Docker has an environment that maps a name to a container
      * “.” = this directory
    - *docker run -p 8787:8787 -v $(pwd):/home/rstudio/work -e PASSWORD=pwd -it 611-example* 
      * Need to create a port on a computer (inside on the right, outside on the left)
      * Inside the container, docker always runs on 8787
      * “-v” mapping the volumes from outside the container to inside the container
      * “-e”: environment variable called password
    - Go to web-browser and *localhost:8787*
    - Every time you install something inside the container, add a line to your docker file

Class 5: 8/24/22 (Docker)

* The idea behind Docker
  + Distribute Docker container along with the project and we have already set up all the programs (like a tiny computer that lives inside the computer)
  + Running a version of Linux/Ubuntu
  + Usually using the rocker images (preloaded RStudio)
    - The Rocker Project (Docker Container for R Studio)
    - We’re using rocker/verse
    - “docker run -it rocker/verse /bin/bash”
* Commands within Docker
  + docker run --rm -ti -e PASSWORD=yourpassword -p 8787:8787 -v "$(pwd)":/home/rstudio/work rocker/verse
    - “-p”: port
    - “-v”: directory
    - “-e”: password
    - 16767 is the outside of the container
    - 8787 is inside of the container
    - Map 16767 to inside container 8787
    - Users have a particular place on the computer to put their files, located at /home/sophieshan
    - Rocker container is set up so there is a pre-existing user called rstudio, access files in directory called work
  + docker kill 7345
    - Kills the port/docker container 7344
  + Then go to localhost:8787
    - And log in via rstudio, your password
  + docker build . -t 611-example
    - build a container called 611-example which had an additional library installed
* Project
  + Think of project as the source code
  + Have perfect faith that our source will perfectly reproduce the figures we have
  + Periodically kill your R analysis and rerun it to make sure it’s reproducible
  + Create a Docker file (text file: just representing text) in your image directory (tell machine to download the packages)
    - FROM rocker/verse (container you want to build the container on)
    - RUN R -e “install.packages(\”matlab”\)”
    - RUN apt update && DEBIAN\_FRONTEND=noninteractive apt install -y python3-pip
  + docker

Class 4: 8/23/22 (Lab)

* Can redirect a standard output to a file on the disk
  + echo hello > /tmp/example
  + cat /tmp/example
* Commands from “Compute Resources”
  + cut -d ‘ ‘ -f1, 6 (splitting by spaces, and getting the 6th element)
  + md5sum is a hash function
* How to utilize the git repository
  + How to access files
    - .html (open via google chrome)
  + How to clone into repository
    - git clone <https://github.com/Vincent-Toups/datasci611.git>
  + Refresh depository
    - cd datasci611/
    - git pull
* VCL vs Longleaf vs Local Computer
  + VCL
    - base\_datasci611 is a custom environment pre-loaded with software needed for this class (e.g. Docker)
    - No long-term storage
  + Longleaf
    - Computing cluster available to UNC users
    - Provides long-term storage
    - Not pre-installed with Docker
  + Local computer
    - If you can run Docker from your computer
* Docker: Container vs Image
  + A Docker container is a run-time environment where users can isolate applications from the underlying system. These containers are compact, portable units in which you can start up an application quickly and easily.
  + A Docker image is a file that contains the source code, libraries, dependencies, tools that are needed for an application to run.
  + Start docker instance
    - docker run -e PASSWORD=hello -v $(pwd):/home/rstudio/work -p 8998:8787 rocker/verse

Class 3: 8/22/22 (Unix and Docker)

* Docker
  + Solves the problem of distributing a complex file which contains your analysis
  + If someone doesn’t want to use Docker, they are able to see the packages they need to install
  + We will create a docker file associated with our project
  + Hosting containers that other people have built
    - Start with container that already exists: FROM rocker/verse
    - Add a bunch of commands on top of it: ARG …
  + Most Docker containers have a command line in the last couple lines of the file
  + Commands
    - docker run hello-world
      * run is a subcommand which pulls the container hello-world
      * hello-world container just prints out the ‘Hello from Docker’ message
    - docker run -it rocker/verse /bin/bash
      * we want it to run the command inside the container /bin/bash (which is inside the container)
      * can install a different text editor instead of RStudio
    - apt update
      * updates the software
    - exit or CONTROL + D (will send the end of file token)
      * exits the bash
* Commands that professor used in class
  + docker run -e PASSWORD=pw -i rocker/verse /bin/bash
  + apt update && apt-get install -y nodejs
  + find . -iname “\*.R” | wc
    - Find the number of files that end with .R
    - wc: word count
  + head power\_gender\_data.csv | grep -v females
    - grep: reads the standard input and filters it so that you can output the standard output (only shows strings in the file that has females in it)
  + cat power\_gender\_data.csv | cut -d’,’ -f3 |sort |uniq -c
    - |: pipe send the output of the first commands
    - cut: tokenizing the string by the commas
    - sort: sorting the genders
    - uniq: tells you just in this dataset what the unique gender terms are
    - uniq -c: gives you the counts
  + find . -iname “\*.R” |xargs grep power\_gender\_data
    - Figure out all the places in a project that you’ve used a file
  + find . -iname “\*.R” |xargs grep function
    - Find every line of every R file which have the word function in it
  + git status | grep modified
    - Get every file that has been modified
  + Git log | grep scraping
    - Get every line that has to do with scraping
  + apropos lisp
    - returns a list of commands that might be related
* Installing software
  + Package manager is used to install software
  + Linux distribution
    - Linux is just a kernel (core of the operating system) and all of the standard commands are provided by someone
    - On top of that people build package environments (called linux distributions)
    - Ubuntu’s package manager is called Apt
* Shell/Bash Evaluation Rules
  + Token the input string by breaking on spaces (but not within double quotes)
    - Token is a chunk that bash can understand (ex. man grep; break them up into a list of tokens); start with a string and convert into a list of tokens
    - man grep -> [‘man’, ‘grep’]
  + Examine the first token to see if it is a command
    - Types “which” before the command
    - What species of object is a path is? Path is an environment variable (a list of paths to search for commands)
      * Why when we type echo $PATH, we get a list
      * /home/sophieshan/.local/bin: is a environment variable
    - Environment variables maps string to strings (just a function that maps from strings to strings)
  + After tokenization, substitute the environment variables (where they are proceeded by $)
    - If an environment variable doesn’t exist substitute the empty string
  + Runs $<(shell code)> and replaces the output into $<(shell code)>
  + Globs are expanded
    - \* -> all the files in the directory (head.\* -> all files starting with head)
  + Runs the command by connecting standard input (is a file that it’s reading data from) and standard output (the file that’s outputted) to the command and passing in the arguments
    - Taking the rest of the tokens on the line and passed to the command as arguments (command will be able to access the arguments)
    - Ex. echo $PATH -> [‘echo’ ‘$PATH’] and then shell replaces PATH with the string it is linked to
* Helpful commands:
  + man: gives you helpful information
  + which: look it up in the path and show whether it is a command (print the location on our file system where our file is)
  + “ – double quotations (will prevent bash from tokenizing)
  + ‘ – single quotations
  + ` - back tick (`` will have the effect of running a subshell on a command)
    - Ex. echo $(ls) is the same as echo`ls`
  + cat: prints the input of the files (opens the standard input and prints it out to the standard output)
    - cat unix.\* (act if all the files that start with unix. are here)
  + head: shows you the first couple of lines of the file

Class 2: 8/17/22 (How the bash shell works)

* Unix
  + Why Learn Unix
    - Most encountered as the underlying idiom of OSX important systems, powers the world
    - A large proportion of computers that are making the internet run are using Unix
    - Unix philosophy is that programs should be small and do one small thing
      * Pass input to a program in two ways:
        + Via the command line
        + The standard input (printing to the standard output: show up on our terminal)

Pipe output of one unix command into another

* + What happens when we open up a shell?
    - A shell is an interactive text space for talking to computers
    - Can also access terminal from RStudio browser
  + Text scripters
    - Emacs
    - Nano
    - VI
      * Quit VI: “:q!”
  + Linux commands
    - When you type in one word with, it’s interpreted by shell as a command
      * How does shell know how to look for commands? B/c there is a path (echo $PATH)
      * Looks in order of the files and executes the first one that it finds
      * Splits up what you type via spaces (“tokenization” occurs on spaces)
    - $[text]: find variable in the environment (“shell substitution”)
      * Where are environment variables located?
    - env: print out all the environment variables bound in the current session
      * environment maps names to strings (ex. USER=toups)
    - echo: just print out what you tell it to
    - which [command]: tell you where in the hard drive you’re finding this command
    - emacs [rules.txt]: opens the text editor called “rules.txt”
    - export [MYVAR=hello]: create a environment variable
      * export is shell built-in
    - echo $USER | rev: reverses the output of echo $USER via pipe
    - ls: lists all of the files
    - find . -type f -iname “\*.csv\*” | wc: gives you the count of how many csv files
    - ls -1: makes the list of things in a straight line
      * Anything after “-“ is a switch
    - Creating your own command
      * bash say-hello.sh (running the hello shell)
      * chmod u+x say-hello.sh
      * ./say-hello.sh
      * Make it run from everywhere (put it on the path)
        + echo $PATH
        + cp say-hello.sh /home/toups/.local/bin/
        + say-hello.sh (without the dot, will run it from everywhere)
      * cd ~/ (go to home directory)
      * mv ~/.local/bin/say-hello.sh ~/.local/bin/say-hello
    - Rules:
      * Tokenize by breaking strings on spaces, until you see a pipe “|”
        + After the pipe repeats these rules
      * Replace anything like $<TXT> with the environment variable
        + Environment variables that don’t exist are replaced with nothing
        + Replace anything like $(<shell code>) with the result of running that code
      * Look up the first token to find the command or lookup and run the “build-in” (of which export is an example)
      * Run that command with the rest of the tokens (after applying 2)
* Login to VCL
  + Vcl.unc.edu -> New Reservations
  + Put in ssh sophshan@IPaddress (IP address is unique identifier)
* Linux commands
  + sudo: I would like you to run this as the administrator account (root)
  + man…: tell us more information about a command (look at the synopsis)
  + sshfs: ssh (secure shell to connect to VCL); fs (file system); use ssh to simulate file system on local machine (could technically ssh to longleaf but want this one machine)
  + $USER <- shell variable (use $ to expand the variable)
  + echo: will print stuff back to you (ex. echo $USER -> sophshan)
  + scp: I want to use secure shell to copy something
    - ex. scp -r bios611 sophshan@IPAddress:storage/bios611 (-r means copy everything in directory; it is recursive)
* Get RStudio window running in Linux
  + docker run -e PASSWORD=hello -v $(pwd):/home/rstudio/work -p 8998:8787 rocker/verse
    - command is run
    - set password to hello
    - run within our directory ($(pwd):/home/rstudio/work)
    - on the left of the colon is outside the docker container (so we want to connect our current working directory to directory inside docker container)
    - port outside the docker container: 8998
    - port inside the docker container: 8787
    - name the image we want to run (rocker/verse; image built to help us run Rstudio easily)
    - created a folder “work” and connect to the directory from Linux
* How to access lectures
  + (extra lectures) Github -> X-02-vcl
    - RPres (clone repository can download and look at it)
* Longleaf provides students with persistent storage
  + Tell Linux machine to connect to precise file store

Lab 1: 8/16/22

* Virtual Computing Lab
  + vcl.unc.edu
  + Reservations -> New Reservations
  + Go into shell: ssh sophshan@IPaddress
  + Test Docker is running correctly: docker run hello-world
  + Using Docker to interactively with rstudio: docker run -e PASSWORD=pwd -p 8787:8787 rocker/verse
    - Image that has RStudio running in it
    - Told it that we want the password to be pwd
  + Make a directory called work in RStudio
    - pwd
  + Link to RStudio Server:
    - Go to Google Chrome: 152.19.196.20:8787
    - Username: rstudio
    - Password: pwd
  + Kill RStudio:
    - Control + C
* Shell:
  + Working directory: A location where files will be (*pwd* or
    - *which pwd*: which folder *pwd* is located in (it’s a command and a folder)
  + Root directory: /Users/SophieShan
  + Printing out files in the directory: *ls*
    - Subdirectories are colored as blue
    - Files coded as green
  + If you have a question regarding commands: *man ls* (*q* to escape the man page)
  + More information regarding files: *ls -l* (permission, user groups)
  + Copy: Shift + Control + V
* IP: Numerical addresses that locate individual addresses
  + sophshan@IPaddress
* Docker are virtual computers that are temporary
  + Can configure them myself
  + Only install software on Docker container

Class 1: 8/15/22

* Data Scientist vs Statistician
  + A statistician takes data, typically from a designed experiment (often the statistician is involved in the design) and produced particular sorts of answers
  + A data scientist takes “raw” data
* Tasks data scientists do:
  + Visualization (make lots of plots!)
  + Exploratory Data Analysis (quickly understand the dataset, how do I identify outliers, etc.)
  + ETL (Extract, Transform, Load)
    - Ex. Data from water meters, ask questions
  + Application Development (ex. show users the real-time status of water in a large city)
  + Operations (setting up pipelines between database and models)
  + Orchestrating data collection standards (ex. Modifying the practices of where the data is coming from)
* Data Science Project Lifetime
  + Collect
  + Tidy
  + Analyze
  + Report
  + Publish
* Version Control
  + Git – Track how your code changes over time
  + Unix skills – tie everything together
  + Programming skills – R and Python and Shell
  + Docker – Reproducible development environments
  + Make – reproducible builds (build system)
* Why shouldn’t we use jupyter/rmarkdown?
  + They obscure dependencies
  + They maintain a lot of global state (variables that hang around even after you finish using them)
  + They discourage “factoring”
  + They pose a modest technical lock in
  + They don’t play well with git
  + They interleave reporting with processing and these are fundamentally disjoint tasks
* What this class will cover in analysis
  + Exploratory data analysis in R and Python
  + Processing, Joining, and Cleaning Data (ex. SQL, dplyr)
  + Visualization (ggplot2)
  + Modeling (clustering, classification)
* Project example
  + README – first thing you have on your git repository
    - Project Organization