**Key points**

* Recap: there are four stages: working directory, staging area, local repository, and upstream repository
* Clone an existing upstream repository (copy repo url from clone button, and type "git clone <url>"), and all three local stages are the same as upstream remote.
* The working directory is the same as the working directory in Rstudio. When we edit files we only change the files in this place.
* git status: tells how the files in the working directory are related to the files in other stages
* edits in the staging area are not tracked by the version control system by default - we add a file to the staging area by git add command
* git commit: to commit files from the staging area to local repository, we need to add a message stating what we are doing by git commit -m "something"
* git log: keeps track of all the changes we have made to the local repository
* git push: allows moving from the local repository to upstream repository, only if you have the permission (e.g. if it is yours)
* git fetch: update local repository to  be like the upstream repository, from upstream to local
* git merge: make the updated local sync with the working directory and staging area
* To change everything in one shot (from upstream to working dir), use git pull (equivalent to combining git fetch+ git merge)

**Code**

pwd

mkdir git-example

cd git-example

git clone https://github.com/rairizarry/murders.git

cd murders

ls

git status

echo "test" >> new-file.txt

echo "temporary" >> tmp.txt

git add new-file.txt

git status

git commit -m "adding a new file"

git status

echo "adding a second line" >> new-file.txt

git commit -m "minor change to new-file" new-file.txt

git status

git add

git log new-file.txt

git push

git fetch

git merge

Push up stream:

cd ~/projects/murders

git init

git add README.txt

git commit -m "First commit. Adding README.txt file just to get started"

git remote add origin <https://github.com/rairizarry/murders.git>

OR

git remote set-url origin

<https://github.com/jarektan/murders.git>

git push # you may need to add these arguments the first time: --set-upstream origin master

More advanced Unix commands

* Arguments typically are defined using a dash (-) or two dashes (--) followed by a letter of a word.
* r: recursive. For example, rm -r <directory-name>: remove all files, subdirectories, files in subdirectories, subdirectories in subdirectories, etc.
* Combine arguments: rm -rf directory-name
* ls -a: Shows all files in the directories including hidden files (e.g. .git file when initializing using git init) (a for all).
* ls -l: Returns more information about the files (i.e. l for long).
* ls -t: Shows files in chronological order.
* ls -r: Reverses the order of how files are shown.
* ls -lart: Shows more information for all files in reverse chronological order. You can combine any valid arguments
* **Getting Help:** Use man + command name to get help (e.g. man ls). Note that it is not available for Git Bash. For Git Bash, you can use command -- help (e.g. ls --help).
* **Pipes:**Pipes the results of a command to the command after the pipe. Similar to the pipe %>% in R. For example, man ls | less (and its equivalent in Git Bash: ls --help | less). Also useful when listing files with many files (e.g ls -lart | less).
* \* means any number of any combination of characters. Specifically, to list all html files: ls \*.html and to remove all html files in a directory: rm \*.html.
* ? means any single character. For example, to erase all files in the  form file-001.html with the numbers going from 1 to 999: rm file-???.html.
* Combined wild cards: rm file-001.\* to remove all files of the name file-001 regardless of suffix.
* **Warning: Combining rm with the \* wild card can be dangerous. There are combinations of these commands that will erase your entire file system without asking you for confirmation. Make sure you understand how it works before using this wild card with the rm command.**

SHELL explaination

* In Unix, variables are distinguished from other entities by adding a $ in front. For example, the home directory is stored in $HOME.
* See home directory: echo $HOME
* See them all: env
* See what shell is being used: echo $SHELL (most common shell is bash)
* Change environmental variables: (*Don’t actually run this command though!*) export PATH = /usr/bin/
* In Unix, all programs are files. They are called executables. So, ls, mv, and git are all files.
* To find where these program files are, use which. For example, which git would return /usr/bin/git.
* Type ls /usr/bin to see several executable files. There are other directories that hold program files (e.g. Application directory for Mac or Program Files directory in Windows).
* Type echo $PATH to see a list of directories separated by ":".
* Type the full path to run the user-created executables (e.g ./my-ls).
* Regular file -, directory d, executable x.
* This string also indicates the permission of the file: is it readable? writable? executable? Can other users on the system read the file? Can other users on the system edit the file? Can other users execute if the file is executable?
* Be aware of common commands and know what they do.
* open/start - On the mac open filename tries to figure out the right application of the filename and open it with that application. This is a very useful command. On Git Bash, you can try start filename. Try opening an R or Rmd file with open or start: it should open with RStudio.
* nano - A bare-bones text editor.
* ln - create a symbolic link. We do not recommend its use, but you should be familiar with it.
* tar - archive files and subdirectories of a directory into one file.
* ssh - connect to another computer.
* grep - search for patterns in a file.
* awk/sed - These are two very powerful commands that permit you to find specific strings in files and change them.