Welcome to The Software Carpentry at MSU Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

----

Useful links:

- Workshop webpage: https://msu-icer.github.io/2025-03-03-msu/
- Post-workshop survey: https://carpentries.typeform.com/to/UgVdRQ?slug=2025-03-03-msu

----

Day 2

Sign-in:

Write down your name, your email address (@msu.edu preferred), and explain a project you recently worked on in just three words. Then share your answer with your tablemates.

- Kristen Mapes, kmapes@msu.edu, computational manuscript analysis
- Elizabeth Gormley Liffley, gormle11@msu.edu, Disability Mental-Health Mixed-Methods
- Tyler Smith, smit3564@msu.edu, tiktok labor unions
- Justin Wadland, wadlandj@msu, haiku translation python
- Natali Gonzalez, gonza829@msu.edu, Drought Nutrient Managment
- Nicholas Panchy, panchyni@msu.edu, AI Cancer Data
- Allison Eden, edenalli@msu.edu - movies, morality, memory
- Salih Cevik, ceviksal@msu.edu - teacher performance growth
- Pat Bills billspat@msu.edu global heatwave modeling (in R!)
- Jonathan Barber barber85@msu.edu extracting data for LLM
- Craig Gross, grosscra@msu.edu - Teaching programming R
- Navya Malik, malikna1@msu.edu - Learning programming R
- Noor Alomari,noor.f.alomari@wmich.edu - copmiting simulations and analysis

Version Control with Git

- Instructor: Sruthin Gaddam

Notes

to check if you have git installed, you can try either
  git help
  man git
  (if you open a 'man' (aka manual) page, press the q key to quit)

- Version control system
    - Helps you keep track of revisions to files
    - Makes it easy to collaborate with others
    - See who made what change
    - Revert changes and go back in time
    - Other version control systems: CVS, SVN, but most people use Git with GitHub and GitLab now
- Check that git is installed
    - On Mac
        - Run "man git"
        - Press the "q" key to exit
    - On Windows
        - Open Git Bash
        - Run "git --help"
- Don't keep multiple revisions of the same file, but work from a single file and keep track of the changes you make in multiple revisions
- Set some one-time configuration
    - git config --global user.name "INSERT YOUR FIRST AND LAST NAME HERE"
    - git config --global user.email "INSERT YOUR EMAIL HERE"
    - On Mac
        - git config --global core.autocrlf input
    - On Windows
        - git config --global core.autocrlf true
    - git config --global core.editor "nano -w"
    - git config --global init.defaultBranch main
- To edit your configuration at any time
    - git config --global --edit
- Get help on different configuration options
    - git config --help
- git init
    - Turn your current working directory into a "git repository"
    - Creates a .git directory (hidden because it starts with a dot)
        - Can see by using "ls -a"
- git status
    - Tells you the status of your current project
    - Shows you:
        - the branch you are on
        - if you have any "commits"

        - what changes it detects
        - some suggestions
- Don't need to track subdirectories using their own "git init". All subdirectories are tracked if the parent directory is.
- git add <insert-filename-here>
    - Tells Git you want it to "track" the file
    - Can add everything with "git add ."
        - USE WITH CAUTION: Convenient, but it might pick up things you weren't ready to commit to your history yet
- git commit -m "INSERT MESSAGE HERE"
    - Message is usually less than 50 characters
    - Should (succintly) explain what changes were made
    - You can run "git commit --amend" to change your last commit message
- Main workflow
    - Make a change (e.g., add a line of text to a file with nano)
    - **Stage** the change: Tell git you want it to know about that change (git add)
    - **Commit** the change: Tell git you want it to remember that change in the history of the repository (git commit)
    - (Optional) **Push** the change: Copy those changes to an online version of the repository
    - See visual here: https://swcarpentry.github.io/git-novice/fig/git-staging-area.svg
- git log
    - Show you all past commits (or changes) made to your repository
    - Each commit has:
        - A unique identifier or "hash" or "sha" that is a long string of letters and numbers
            - If you need to reference the commit, you can either use the entire hash or just the first 7 characters (referred to as the "short sha")
        - The author of that commit
        - The date/time the commit was made
        - The message associated with that commit
    - Use, e.g., "git log -1" or "git log -3" to see only the last 1 or three commits
    - Use "git log --oneline" to see the short versions of each commit
    - Use "git log --graph" to show links between the commits (more interesting when there are more branches)
- git diff
    - Show all changes you made since your last commit
    - Use "git diff --staged" to show differences between last commit and staged files
- git diff HEAD filename.ext
    - Compare the changes of filename.ext with the "head" or most recent commit
- git diff HEAD~1 filename.ext
    - Compare the changes of filename.ext with the commit before the most recent commit (the "head commit minus 1")
    - Can use any number instead of 1 to go further back
- git show HEAD~2
    - Show the change that happened in the commit two before the most recent one
    - Can replace "HEAD~2" with a reference to any commit (e.g., "HEAD", "HEAD~1", the full commit hash, or the short sha)
    - Can also add a filename after the commit reference to see what happened to that specific file
- git restore filename.ext

    - Will restore filename.ext to the version in the most recent commit
    - Can also reference a commit to restore to the version of the file in that commit
        - Example: git restore HEAD~2 filename.ext
- .gitignore
    - This is a file in your git repository that you can edit with "nano .gitignore"
    - Hidden since it starts with a dot, so you need "ls -a" to see it
    - Add filenames in this file to tell Git you don't want it to track those files
    - Can also use wildcards to ignore lots of files (e.g., "*.png" to tell Git you want it to ignore all png files)
    - To ignore a directory, add the path to the directory (e.g., "receipts/data")
    - Can start a line with "!" to undo previous ignores:
        - Example

        - *.png
        - !test.png

        - This keeps only the "test.png" file but ignores all other .png files
- Git will not track empty directories by default
    - If you want to do that, add an empty file called .gitkeep to that directory with "touch .gitkeep"
- Use GitHub to share your repositories or save them on online
- Create a new Repository on GitHub
    - Go to GitHub https://github.com/
    - Click the plus in the top right corner
    - Choose "New repository"
    - Add a repository name, e.g., "recipes"
    - Click "Create Repository"
- Linking a Git repository to its online version on GitHub
    - git remote add origin <insert-url-of-github-repository-here>
    - "origin" refers to the "remote" copy of the repository on GitHub
- git remote -v
    - Show all of the links to the remote copies of your repository
- git branch
    - Show which branch of the repository on
    - By default this is "main"
- git push -u origin main
    - "git push" - copy the changes to to a remote version of the repository
    - "-u origin main" - specify *which* remote version to push to and which branch
- Logging into your GitHub account from your computer
    - Create a new SSH keypair
        - Analogy: Lock and key is to public and private keys
        - ssh-keygen -t ed25519 -C "INSERT EMAIL HERE"
    - Show your public key
        - cat ~/.ssh/id_ed25519.pub
        - Copy your public key
    - Put it into your GitHub SSH keys
        - Go to your GitHub SSH keys: https://github.com/settings/keys
        - Click "New SSH key"
        - Paste your public key into the "Key" box at the bottom
        - Add a title, e.g., "Sruthin's Macbook key"

        - Click "Add SSH key"
- Collaborating with a friend
    - Give your friend access to your GitHub repository
        - In your repository, click the "Settings" tab
        - Click the "Collaboration" section
        - Click the "Add people" button
        - Enter your friend's GitHub account name
    - In the email your friend sent you, accept their request, and go to their repository
    - On the main page, click the Code button and copy the SSH link
    - On your computer, create a new directory to put their repository in, e.g., "mkdir collaboration"
    - Change to that directory, e.g. "cd collaboration"
    - Copy their repository
        - git clone <friends-repository-url-here>
    - Change to their repository with "cd recipes"
    - Make a change and send it back
        - Make a change to their file
        - git add <file>
        - git commit -m "Make a change"
        - git push -u origin main
    - Check your repository on GitHub to see your friend's change
    - Go back to your repository on your computer
        - cd ~/Desktop/recipes
    - Pull down your friend's change
        - git pull

Building Programs with R (part 2)

- Instructor: Pat Bills
- the scripts created for this portion of the workshop are available on github in a public repository for your reference: https://github.com/billspat/swcarpentry_msu_r_day_2

Notes

- Git repositories and R
    - You can access a terminal from RStudio (next to the console tab)
    - When you create a project, you can specify to create a new git repository along with it (which we did yesterday)
    - You can run git commands in this terminal to track your R work with git
    - Create a Repository in GitHub using steps from previous lesson
    - Make your first commit in the Terminal using the steps from previous lesson
    - Then follow steps on GitHub to push your local repository to the remote one
    - You can also use the Git pane in RStudio to do the Git workflow ("git add", "git commit", "git push")
- Use broom in the environment pane to remove any variables that are hanging around from

yesterday
- Create a new dataframe with just the rows that are part of the Americas
    - americas <- gapminder[gapminder$continent == "Americas",]
- Create a plot of these countries, each on their own panel, with x labels rotated by 45 degrees
    - ggplot(americas, mapping = aes(x = year, y = lifeExp, color = continent)) +
    -   geom_line() +
    -   facet_wrap(~ country) +
    -   theme(axis.text.x = element_text(angle = 45)) +
    -   labs(title = "Life Expectancy by Year", x = "Year", y = "Life Expectancy", color = "Continent")
- Functions
    - Collection of statements to turn an input into an output
    - Lets you do the same thing multiple times
- Writing a function
    - Assign a function to a variable using the "function" command
    - Put arguments in parentheses after "function"
    - Put all of the commands in curly braces
    - Output a variable using "return"
    - Example:
        - fahr_to_kelvin <- function(temperature_f) {
            - kelvin <- (temperature_f - 32) * (5/9) + 273.15
            - return(kelvin)
        - }
    - To use this function:
        - fahr_to_kelvin(32)
    - Can also use with vectors:
        - fahr_to_kelvin(c(0, 32, 100))
- You can save function definitions in scripts, and later click the "Source" button in RStudio to get access to the functions
    - You can also run the "source" command in R to bring in all of the functions in a script
    - E.g., "source("scripts/converters.R")"
- You can put the output of one function as the input of another function
    - E.g., kelvin_to_celsius(fahr_to_kelvin(32))
    - "Function composition"
- Using dplyr
    - Swiss army knife for data manipulation
    - Package that we installed when we installed ggplot2
    - To use it, run "library(dplyr)"
    - Example to select only a few columns from the dataframe:
        - year_country_gdp <- select(gapminder, year, country, gdpPercap)
    - **select** - takes columns from the dataframe
    - Example to filter out only certain countries:
        - year_ghana_gdp <- filter(year_country_gdp, country == "Ghana")
    - **filter** - takes rows from the dataframe matching a certain condition
    - **rename** - renames a column
        - Example: gapminder %>% rename(gdp_per_capita = gdpPercap)

        - **count** - count the number of entries in a column separated by the different values
            - Example: Find how many countries are in each continent (and sort them decreasing)
                - gapminder %>%
                - filter(year == 2002) %>% count(continent, sort = TRUE)
        - **n** - gives the number of rows that it's working on in another expression (e.g., when summarizing, it will give the number of rows it's summarizing)
            - Example: Compute the standard error in the mean for all continents
                - gapminder %>%
                - group_by(continent) %>%
                - summarize(se_le = sd(lifeExp) / sqrt(n()))
        - **mutate** - create new columns from existing ones
            - Example: Compute the total GDP by multiplying the population and per capita GDP
                - gapminder %>%
                - mutate(gdp = pop * gdpPercap)
- Piping
    - The pipe operator in R is %>%
    - The pipe takes the output of the last command and uses it as the first argument for the next function
    - Same example as above to get year_ghana_gdp:
        - year_ghana_gdp <-
        - gapminder %>%
        - select(year, country, gdpPercap) %>%
        - filter(country == "Ghana")
    - Extending this example to rename the gdpPercap column
        - year_ghana_gdp <-
        - gapminder %>%
        - select(year, country, gdpPercap) %>%
        - filter(country == "Ghana") %>%
        - rename(gdp_per_capita = gdpPercap)
    - You can pipe a dataframe into the "View()" function to open a tab in RStudio to scroll through the data like a spreadsheet
        - Example:
            - gapminder %>% View()
- Grouping
    - You can split a dataframe into others using the **group_by** function from dplyr
    - Example to create new dataframes for each continent:
        - gapminder %>%
        - group_by(continent)
    - You can use the **summarize** function to summarize each group into one number
    - Example to take the mean GDP per capita for each continent:
        - gapminder %>%

        - group_by(continent) %>%
        - summarize(mean_gdpPercap = mean(gdpPercap))

    - The previous example creates a new dataframe with one row for each continent and the new "mean_gdpPercap" column showing the mean GDP per capita for each continent
    - All other columns go away after summarizing
- To find rows that satisfy multiple conditions, you can use the | symbol to combine them
    - Example to find maximum or minimum GDP percapita

        - gapminder %>%
        -   group_by(continent) %>%
        -   summarize(mean_gdpPercap = mean(gdpPercap)) %>%
        -   filter(mean_gdpPercap == min(mean_gdpPercap) | mean_gdpPercap == max(mean_gdpPercap))

- The %>% pipe comes from the "magrittr" package, but a similar pipe |> has been built into R
    - Both are used, but |> is becoming more popular
    - They are slightly different, but mostly work the same
- You can explicitly say what package a function comes from by using "::"
    - Example: dplyr::filter
    - This lets you use a function even if you haven't loaded it with the library function
- **Plotting code from previous lesson:**

- ggplot(americas, mapping = aes(
-     x = year,
-     y = lifeExp,
-     color = continent)) +
-   geom_line() +
-   facet_wrap(~ country) +
-   theme(axis.text.x = element_text(angle = 45, size = 8)) +
-   labs(title = "Life expectancy by Year",
-          x = "Year",
-          y = "Life Expectancy",
-          color = "Continent")

- Combining with dplyr and pipes to plot just countries from Asia:

- gapminder %>% filter(continent == "Asia") %>%
-   ggplot(americas, mapping = aes(
-       x = year,
-       y = lifeExp,
-       color = continent)) +
-     geom_line() +
-     facet_wrap(~ country) +
-     theme(axis.text.x = element_text(angle = 45, size = 8)) +
-     labs(title = "Life expectancy by Year",
-            x = "Year",
-            y = "Life Expectancy",
-            color = "Continent")

- Converting this into a function to choose any continent:

- gapminder_by_continent <- function(df, Continent) {
-   df %>% filter(continent == Continent) %>%

- ggplot(americas, mapping = aes(
-   x = year,
-   y = lifeExp,
-   color = continent)) +
- geom_line() +
- facet_wrap(~ country) +
- theme(axis.text.x = element_text(angle = 45, size = 8)) +
- labs(title = "Life expectancy by Year",
-      x = "Year",
-      y = "Life Expectancy",
-      color = "Continent")
- }

- Example function call:
    - gapminder %>% gapminder_by_continent("Asia")
- Creating reports with RMarkdown and knitr
    - Create a new R Markdown file (with extension .Rmd) from RStudio's new file menu
    - You can write Markdown, but add R commands that will add their output to the file
    - Similar to a Jupyter notebook for Python code
    - To add R code, create a code block starting with ```{r} and ending with ```
    - Example:
        - ```{r}
        -   plot(cars)

        -
          ```
    - Can run code blocks by clicking the play button in the top right of the block
        - Code blocks are run from the directory where the notebook gets saved
    - The "Knit" button in RStudio will render it as a file that you can share

---------------------------------------------------------------------------

## Day 1

### Sign-in:

Write down your name, your email address (@msu.edu preferred), and your favorite class that you ever took. Then share your answer with your tablemates.

- Sruthin Gaddam, gaddamsr@msu.edu, Interaction Design
- Navya Malik, malikna1@msu.edu,  Improvisation
- Tyler Smith, smit3564@msu.edu, Social Inequality
- Nicholas Panchy, panchyni@msu.edu, Mathmatical Modeling
          ```

- Justin Wadland, wadlandj@msu.edu, One of the creative writing classes during MFA
- Natali Gonzalez, gonza829@msu.edu, Sustainable Agriculture Food Systems Field Tour
- Allison Eden, edenalli@msu.edu, entertainment media
- Joseph Ulasi, ulasijos@msu.edu, Advanced Plant Breeding
- Elizabeth Gormley Liffley, gormle11@msu.edu, art
- Windasari Aliarosa, aliarosa@msu.edu
- Kristen Mapes, kmapes@msu.edu, on-site early medieval art and architecture
- Morgan Gaston, gastonm1@msu.edu, Cognitive Psychology
- Eleanor Carr, carrele1@msu.edu; Linear Algebra
- Dennis Boone <drb@msu.edu>
- Salih Cevik, ceviksal@msu.edu, intro to instructional methods
- Craig Gross, grosscra@msu.edu, Architectural art history
- Yosia Mugume, mugumeyo@msu.edu Molecular Genetics
- Laura Dillon, ldillon@msu.edu, Computer Science & Engineering
- Jeff Kuure, kuure@msu.edu, introductry sculpture
- Pat Bills billspat@msu.edu Spatial Statistics
- Yan Wang wangya81@msu.edu Communication

Automating tasks with the Unix Shell

- Instructor: Nicholas Panchy

Notes:

- Shell Lesson Data Link: https://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip
- Open a terminal/shell
    - On Windows: open start menu and type "git bash"
    - On Mac: open Terminal app
- What is a shell?
    - A way to interact with a computer
    - Similar to a "GUI" (graphical user interface) where you can click to do tasks
        - Accessible but not very easy to automate
    - The shell allows you to interact with the computer using text only
        - Automation is much easier to achieve
- Using the "bash" shell - the language we're using to type
- bash - "Bourne Again Shell" successor to the "Bourne Shell"
    - note for Mac users: the most recent version of MacOS comes with the "z-shell" or zsh which is similar enough to bash for most commands from bash to work
- $ prompts you to type text
- "ls" - lists the files and directories in a directory
    - Same files as you can see in your file browser
    - "-F":
        - Display a slash (‘/’) immediately after each pathname that is a directory, an asterisk

        (‘*’) after each that is executable, an at sign (‘@’) after each symbolic link, an equals sign (‘=’) after each socket, a percent sign (‘%’) after each whiteout, and a vertical bar (‘|’) after each that is a FIFO.
    - "--help"
        - Shows help page
        - Doesn't work on Mac
        - on other terminals use "man" - which shows the *manual* for the command
            - example: "man ls" (to exit the 'man' screen, hit the q key to quit)
    - "-l"
        - Lower case ell, not upper case i
        - Shows "long listing" with extra information about each file or directory
    - "-h"
        - When using the "-l" option, prints sizes in a human readable form
    - "-t"
        - Sorts by timestamp from most recently changed to oldest change
    - "-r"
        - Reverse order that files are listed (useful with "-t" to show oldest to newest)
    - Can add a directory location to list the contents in a directory
        - E.g. "ls Desktop" to show all files in the desktop
    - "-a"
        - Show "hidden" files that start with a "."
    - "-R"
        - "Recursive"
        - Show the contents of the directory and every directory inside that directory (all the way down)
- Can combine options together
    - "ls -l -h" is the same as "ls -lh"
- Shell is case-sensitive and commands need to be spelled correctly
- "pwd" - **p**rint **w**orking **d**irectory
- Locations (or "paths") start at "/" - the root directory
    - Subdirectories are separated by "/"
    - E.g., /c/Users/ICER_Guest
        - c is in the root directory
        - Users is in the c directory
        - ICER_Guest is in the Users directory
        - On MacOS the path will look like /Users/ICER_Guest or /Users/username
            - also on MacOS and Linux you may see a "~" which is used as shortcut for this folder
- The arrangement of the file system is represented as a tree.
    - "root" directory is at the top
- "clear" - will clear the screen
- Can use up and down arrow keys to show previous commands you typed
- Options can be added to commands and start with a "-" or "--"
    - Changes the way the command works
    - Also called "switch" or "flag"
- "cd" - **C**hange working **D**irectory
    - use "*pwd*" command to check the path pf the directory
    - If you don't provide a directory name after, it will move you back to your ~ or "home" directory

    - Enter the path of the dir using the cd cmd, example, *cd /path/to/dir*
    - **Common shortcuts**
        - Use *"cd ../"* to go back a directory. The "../" represents a previous directyory. You can also chain it, example, *cd ../../*
        - Use "*cd -*" takes you back to where you just were.
        - "*cd .*" is where we are
        - "*cd ~/*" home directory
        - see more in the Lesson materials https://swcarpentry.github.io/shell-novice/02-filedir.html#absolute-vs-relative-paths
- Use the "tab" key to autocomplete directory locations
    - E.g., Type "Desk" then press tab and the shell will fill in "Desktop"
    - Press tab twice to show all your options when there are multiple possibilities
- "." is a shortcut for your current directory
- ".." is a shortcut for one directory up from your current directory
- You can write paths in two ways
    - Absolute - Always starts from the root directory and navigates exactly where you want to go
        - E.g., "/c/Users/ICER_Guest/Desktop/shell-lesson-data/exercise-data"
        - Longer, but always gets you to the exact location
    - Relative - From your current working directory
        - E.g., if you are currently in the Desktop directory you can access the same path above with "shell-lesson-data/exercises-data"
        - Shorter but changes relative to where you currently are
- Arguments are added onto commands like the directory you want to list the contents of with "ls"
- "mkdir" - Make a new directory
    - "-p": Create intermediate directories as required. example mkdir -p "dir1/dir2/dir3"
- If you absolutely need to, use quotes to create or reference filenames with spaces
    - E.g., mkdir thesis/"da ta"
    - This is not recommended!
        - It breaks tab autocompletion in the shell
            - (in some shells like on Mac you may see spaces are represented with a back-slash, like this: "da ta" will be shown as da\ ta without the quotes)
        - You need to add quotes every time you need to access that file
        - It will break with other programs
- "rm" - Remove files. Deleteing is permanent, so be careful
    - BE SUPER CAREFUL WITH THIS COMMAND: "rm -r": Recursively removes everything in the directory
    - "-i" - Will prompt you for confirmation. Type "y" for yes, and "n" for no
- "rmdir" - Remove Directory. You can only remove a directory if it's empty
- "nano" - Open the nano text editor in the command line
    - E.g., "nano draft.txt" will create a new file called "draft.txt" that you can type text in
    - Press ctrl+x to exit
        - You will be prompted to save the file with the filename you started in
- "cat" - Show the contents of a file
    - E.g. "cat draft.txt"
- Some commands will take your prompt away if they're waiting for you to enter something (e.g., "cat" with no file after it)
    - Use ctrl+c to go back to your prompt
- "mv" - Move or rename files

    - E.g., "mv draft.txt quotes.txt" to rename the draft.txt file to quotes.txt
    - "mv quotes.txt .." will move the file to the directory one level up
    - CAREFUL: Moving one file to an already existing file will delete that existing file and overwrite it with the first one
- "touch" - Creates an empty file
    - E.g. "touch quotes.txt" will make an empty file called quotes.txt
- "cp" - Copy files
    - Use similar to "mv", but will make a new copy of the file
    - CAREFUL: This can still overwrite the second file if it already exists
    - "-r": Recursively copy files. Use this when using directories, example, "cp -r src/dir dest/dir"
- "*" - wildcard matches anything
    - example: ls the* lists all the files and dirs starting with "the"
- "head": Prints the first 10 lines of a file
    - Use the -n flag to specify the number of lines. Example: *head -n 10 filename.txt*
- "tail": Gives the bottom lines of a file
- "wc"- Prints the word count in a file
    - Shows number of lines, words, and characters
    - Using wildcard with wc: *"wc *.pdb"*
    - "-l" - Shows only number of lines
- ">" : Redirect takes takes the output and moving it to a file
    - example: wc -l *.pdb > lengths.txt
- "sort": Sort numerically
    - example: *sort lengths.txt*
    - "-r": Sorts in reverse
- "I": A pipe takes the output of one command and make it an input to another
    - Note: this is the character that shares the \ key on the keyboard, not an i or an L
    - wc -l *.pdb | sort -n > sorted_lengths2.txt
- For loops
    - Let you repeat the same command on multiple things
    - Format:
        - for thing in collection
        - do
        -     INSERT COMMANDS HERE
        - done
    - Example
        - for filename in *.dat
        - do
        -     echo $filename
        -     head -n 2 $filename | tail -n 1
        - done
- "echo" - Repeat the argument
- We can type our commands inside a file ending with ".sh" to save them in a "script"
    - can start a new script using nano and the script name like nano myscript.sh
    - we use the .sh extension to remind us it's a shell script, but it's just a text file
- Can run them with "bash script_file.sh"
- ">>" - Add output to the end of a file instead of overwriting the file

    - Will create the file if it doesn't exist
- Use # to add a comment to a script file
    - Anything after the # will be ignored by the shell
    - Commonly used to temporarily ignore a command

To give any input to a generic script, can use dollar-sign to indicate arguments
Inside your script file, use $1 to mean the first argument after the name of the command
bash my_script.sh something -> $1 will have the value "something"
 Script to print a list of things, print_things.sh
 for input in $1; do echo $input; done

 this will expand each item in the parameter sent to the command, but the items must be in quotes
print_things.sh one and a two and a
one

```
 print_things.sh "one and a two and a"
 one
 and
 a
 two
 and
 a
```

modify the script to show the second line of a file to use $1 instead of a hard-coded filename, which would then work on

We wrote a number of custome scripts in the folder:
~/Desktop/shell-lesson-data/exercise-data/creatures

If you would like to play with these scripts yourself, they were:

```
# classify_creatures.sh
# This script writes the 2nd line (CLASSIFICTION) or all *.dat files in the folder to an output file
for filename in *.dat
do
   echo $filename
   # The line below will print each command to the screen
   #echo "head -n 2 $filename | tail -n 1 >> classification.txt"
   # The line below will run the actual commands
   head -n 2 $filename | tail -n 1 > ${filename}_class.txt
done
```

```
# print_input.sh
# The scrit print the content of the input argument to the shell, on word at a time
for input in $1
do
   echo $input
done
```

```
# classify_whatever.sh
# This script takes a list of files and write the 2nd line of each to an output file
for filename in $1
do
   echo $filename
   # The line below will print each command to the screen
   #echo "head -n 2 $filename | tail -n 1 >> classification.txt"
   # The line below will run the actual commands
   head -n 2 $filename | tail -n 1 > ${filename}_class.txt
done
```

Other useful links for bash:
Online manual of bash/unix functions organized by category (https://www.gnu.org/manual/manual.en.html)
Notepad++, a simple text editor with color coding for script in a lot of languages (https://notepad-plus-plus.org/)
Configuring bash to color code nano for scripting (https://askubuntu.com/questions/90013/how-do-i-enable-syntax-highlighting-in-nano)

Building programs with R, Part 1

- **Instructor: Craig Gross**

Why R?

- Written or Exploratory Statistical Analysis
- Open source, free, and widely used. Important for reproducibility
- R is great for automating tasks

Goals: open R, analyze data and generate a plot

About Rstudio:

- R is the programming language, and Rstudio is a way to interface with R. R can be run from the command line, or just use the R interface (aka 'base R') without Rstudio.
- Lower Left pane: Rstudio 'console', which is similar to the console used this morning to enter commands. The prompt in the R console is ">" (instead of "$"for the shell)
- Lower Right pane: list of files in the folder we are working
- Upper Right pane, IF a script is open, is the script source.

Multiple ways to enter R commands: directly in the console, or can start with a script.

Starting with a script, use the File menu, file, new, R script and blank script opens

start a script that's a calculator: type 1 +100 in the script, and click the "run" button at the top. The output shows up in the console at the bottom. To run can also put the cursor in a line of the script and press Ctrl+Enter you can type the command 1+100 directly in the console and hit enter. If you don't finish the command, R will wait for more input and change the prompt to a "+" meaning you adding more the command (the shell doesn't alway do this... but it sometimes it kind of does.

basic arithmetic   3* (5+2).   2^8 (2 to the 8th power)

R can do functions.   For example trigonometry like sin(3)   [sin means sine]

Can put arithmetic inside a function like sin(3.14159/2)

R can do comparison and can put these in your script   1 > 2   or 1 != 2 (which mean 1 is not equal to 2).
The console will print "TRUE" or "FALSE"
To save and re-use calculations, we can put those into what R (and other programming languages) call a variable.   The 'assignment' operator is <- for example
x <- 1/40   - when you run this, nothing shows up. You have to put just the variable name alone (x) on the script or console to see it.
x<-1/40 # running this assigns the value
x   # running this shows the value
Now can use value of a variable in a function   like   log(x)
R shows all the variables defined in the upper right pane called "environment"
Spaces are not allowed in variables.
"_" and "." are allowed, example, "max_height" or "max.height"\
"1:5" prints a list of numbers from 1 to 5 and is called a vector
"2^(1:5)" prints *2 4 8 16 32.* this is called vectorization
'c()': Combine function, example, *c(1,3,5),* makes a vector of 1,3,5
"ls()": Lists all the variables

- *note to list all files, the function is list.files()*

'?' can be prepended to a function to get the manual of the fuction, example, *?rm*

Extending R with 'packages' (bundles of code that includes a bunch of functions).
 - Packages are available via CRAN (The Comprehensive R Archive Network).
 - can be installed using R function install.packages('ggplot2')   <- the package is a name, not a variable, so must be in quotes

- run install.packages('ggplot2') for use in the next part

to use the package in R, need to attach it using the 'library()' function

- library(ggplot2) <- not in quotes for the library command, only for installation

 - can also install using Rstudio in the bottom right pane "packages" tab.   Type in the package name.
For example dplyr. can install via the "Packages" tab and
 can load it by checking the box next to the name of the package in the "Packages" tab as well as with the library() function, but including the library(dplyr) function at the top of your script is better because it ensures the package will be loaded every time you need it,

Challenge link: https://swcarpentry.github.io/r-novice-gapminder/01-rstudio-intro.html#challenge-2

if you have typed the commands into a script file, save it just so you can look back over it.   R script files end in letter R (often upper case but lower is ok, examplecommands.R, can put on your desktop or wherever

Rstudio Projects

we want to create a folder to put all of our stuff, more scripts and some data.
File - new project -> R project, select a folder call it what you like (my_project) , check the "create git repository"

Create folder 'scripts' for R and 'data' for -- you guessed

Challenge 3: download data https://swcarpentry.github.io/r-novice-gapminder/02-project-intro.html#challenge-3

LINK TO DATA: https://swcarpentry.github.io/r-novice-gapminder/data/gapminder_data.csv
right click and use "save link as" to put this file in the data folder where your project

cheat for advanced users: use the 'curl' library - no browser required.

- install.packages('curl')
- library(curl)
- curl_download('https://swcarpentry.github.io/r-novice-gapminder/data/gapminder_data.csv', destfile = 'data/gapminder_data.csv')

data.frame is tabular like a spreadsheet
can create data frame in a script using the data.frame() function, name each column

data.frame(name = c(stuff...), name2 = (stuff,...) ,...) see https://swcarpentry.github.io/r-novice-gapminder/04-data-structures-part1.html for how to enter the cats data

```
cats <- data.frame(coat = c("calico", "black", "tabby"),
```

- weight = c(2.1, 5.0, 3.2),
- likes_catnip = c(1, 0, 1))

can specify a data frame column using dollar sign like this cats$weight is just the vector or values in the weight column
since it's a vector, can do things to all the numbers, like cats$weight +100 and since it's a variable can use inside a function like mean(cats$weight) (average weight).

Using Gapminder

Variables have a Data types. All computer programs have data types, and R had a specific name for data types. Numbers are 'numerics' (which can be a couple of different types)
things with words, using quotes are Character (called 'string' like a string of characters in other languages).
R doesn't talk much about types, but you can tell the type R uses for a variable using the function 'typeof'

In R you usually can add two different types, especially numerics and characters (eg. what is 1 + "calico" ?)
Understanding numbers types is important if you care about numeric precision.

but data types

Important R data type is "logical" = TRUE or FALSE. The output from comparison is Logical type 2 > 1 -> TRUE.   can be used to store equivalant of yes/no in data columns

values in cats$coat are what type?
cat$likes_catnip are zeros and ones, and seems like we want it to be yes/no or TRUE/FALSE, but since it's numeric, R defaults to 'double' data type

how do we convert this numeric tuype to a T/F logical type?   R has functions as.logical() as.characters() etc

as.logical(cats$likes_catnip)   conversts 0/1 to F/T NOTE this works because R, like other languages see zeros as FALSE and non-zero as TRUE.

R vectors (made with c()) have values that are all the same data type. Can't mix character and numeric in single vector.   NOPE: c(1, "calico", FALSE)

R lists collect different types. A list is a collection of any number of things. Items in lists can have names but also have an 'index' (first item in a list is index 1).   Other programming languages like are dumb and start counting at zero.   (the index is an 'offset' from the first item in other languages)

Dataframe is a list of vectors

can use indexing to pick things out of a data frame: cats[1,2] = item in cats occurring in first row of column 2
cats$weight = all items in

other ways to index: https://swcarpentry.github.io/r-novice-gapminder/04-data-structures-part1.html#challenge-5

recommended way to get column (as a vector of values) : cats$weight (<= vector of values in this column)
recommended way to get a row (as a column of values): cats[1,]   (<= list of items in row one)
Aside: list indexing list[1] vs list[[1]] can be confusing.   x <- list[1] is a new list with one item.

A list is a group of train cars, and you get one train car using [], if you want the stuff inside the train car (the vector) use [[]]

Challeng to help understand lists vs vectors: how to pull out just rows 1 and 3 from cats data frame?
cats[1,] does row 1 and cats[3,] does row 3
what data type is a row of a data frame? why is it NOT a vector (hint can a number and character be in a vector)?
the number in cats[X,] is called the index and you can use vectors for indexing.   e.g. c(1,3)

Filtering data using data.frames and indexing.

how to list only cats over a certain weight?   can use a logical vector to index (cats[c(TRUE, FALSE, TRUE),] will pull out the rows that match true (1 and 3) what does cats$weight < 4 output?   logical vector

- cats[cats$weight < 4,] will pull rows that match the logical vector,

reading files (tabular data in 'csv' format)

read.csv() function takes a file path as the first parameter and saves in a data frame.
the file path uses a "slash" like in our bash lesson.
our pathfinder file in in the data folder, so the file path is "data/gapminder_data.csv"
File paths must be sent to functions as character, so they must be quoted

gapminder <- read.csv("data/gapminder_data.csv")
str() = "structure" function, list types of columns
summary() = gives basic stats for numeric columns and number of rows
nrow() number of rows
ncol() number of cols
dim() dimension, used primarily for matrix objects
colnames()
head() and tail()

(skipped ahead to plotting , started a new script)
Goal of script - read in data and plot it

load all packages you need at the top of each script using library()
library(ggplot2)   # why is this not in quotes? quotes not used in library, only install.packages()

ggplot needs 3 things: data, mapping aesthetic, and layers
to plot gdp (x or horizontal axis) by life expectancy (y or vertical axis)
data = gapminder, mapping x is dbpPer
just this is not enough : ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp))
need to have a layer (aka a geom) to say how we want the mapped data to show up :

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
geom_point()
```

challenge plot by year instead https://swcarpentry.github.io/r-novice-gapminder/08-plot-ggplot2.html#challenge-1
challenge 2 add color by continent, a 3rd data frame column. Hint: 'color' is an 'aesthetic' (an aspect of the layer/geom) so it goes in the 'aes()' part of the function.
 - can change x and y to different variables.
 - geoms are added at then end.   geom_line()

- note that geom_line() works best with grouped data, and grouping is a mapping aesthetic , so add group=country to aes() to have lines follow the country.

 - aesthetics in top function are applied to all geoms, but they can have their own mapping = aes

 add geoms with + and will put them on the graph e.g. geom_line() + geom_point()
 for example can color the lines but not the points geom_line(mapping=aes(color=continent) + geom_point()

can change the scale of axes using geom
the notes have nice examples with the outcome: https://swcarpentry.github.io/r-novice-gapminder/08-plot-ggplot2.html

use geom_smooth to add a trend line. full command reference: https://ggplot2.tidyverse.org/reference/geom_smooth.html

for example, to use linear model (aka linear regression) line: + geom_smooth(method="lm")
some of these options are not super intuitive and would be hard to guess and ggplot2 is very complex so just have to look them up

TOMORROW: will use git with R.

HOMEWORK: create a github account if you do not have one.

- https://swcarpentry.github.io/git-novice/#creating-a-github-account

Feedback: what went well and what didn't (feel free to add to this etherpad document)

Notes: