Welcome to The Carpentries 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.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try etherpad.wikimedia.org).

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

Good morning everyone!

The link to the UF Carpentries Club is www.uf-carpentries.org
Global Carpentries: https://carpentries.org/

The website for today's workshop: https://uf-carpentry.github.io/2019-11-04-Intro_to_R_etc/
All links for downloading software and data can be found there.

Today's instructors: Brian (introductions), Chandra (spreadsheets), Dmitri (shell), Eve (intro to R)

Sticky notes: Please put red sticky notes on your laptop if you need a helper to help you navigate through something. We ask you to put a green sticky note if there is a hands-on exercise etc once you finish it- so we can ensure everyone is at the same page.

Bathrooms : Out of the front door, take a left, in the next door you encounter.

Introduction to spreadsheets (Chandra)

- Never edit your raw data (always make a copy)
- Keep track of any changes you are making (open up README document to write all modifications)
- Things like color and font etc are generally not read by coding / programming languages (such as R)
- Variables in columns, observations is rows and one data point and one type of information in individual cell
- Data:

https://ndownloader.figshare.com/files/2252083

Exercise 1 - Identify what is wrong with this spreadsheet
should only be one entry per cell, not multiple
should be organzied vertically not horizontally, Freeze row ,

Weight in grams / "(Scale not calibrated)". Species needs a column. 2014 has a different format (plot

based). Mising data in 2014. Colour coding on 2014.

Units in 2013 DM weight column, missing values in 2013 DM weight column, text in 2013 DS weight column, cell color coding in 2014, inconsistent date formatting 2014.

Weight col has empty cells, instead of zeros

adding unit next to the label
missing value *
different number of observation in each table
heterogeneity in plot number
Missing value is different than zero, just never use zero if not necessery.
Don't use any spaces or special characters if unnecessary.
Species are in different tables. We can combine the tables by creating a "Species" column
Multiple tables in same spreadsheet
Spaces/notes should not be in data table
tables not consistent in organization
no units for weight54
species names abbreviated - what is DM, DO, DS?*
there should be a column specifically for scale calibration
no consistency between worksheets
"gray cell means my measurement device wasn't calibrated correctly" <- a computer will not be able to know that
2013 and 2014 are written differently. There is no consistency between data collected from one year to another to be able to combine both years, if needed
All of these tables should be together in 1
the "dates" sheet does not have the year, so we don't know which year it corresponds to.
Excel use date as numbers.

Info re formatting problems: https://datacarpentry.org/spreadsheet-ecology-lesson/02-common-mistakes/index.html

Quality Assurance

- Data validation allows the user to set rules for data entry (under Data tab, Data Validation). Delete unwanted data or warning.

specify permitted data entry

- Data validation also allows the user to input a message (ex. This value is not between 1-24), or a warning if inappropriate values are entered

Quality Control

- Be sure to sort entire data set by one column, not just the one column (otherwise your data will not be properly sorted)
- If your data has headers, check the box so your headers are not sorted
- Make data uniformly formatted; if not they will not sort properly
- Conditional formatting can be found under the 'Home' tab in Excel. This can be used to sort values to colors, gradients, etc.
- Find and replace (Ctrl+H) can be helpful for deleting unnecessary units (like g). Find 'g' and

replace with nothing. Replace all

Keep in mind that other programs might not be able to open your excel spreadsheet

https://github.com/datacarpentry/spreadsheet-ecology-lesson/blob/gh-pages/data/survey_sorting_exercise.xlsx?raw=true
Creating legends for conditional formatting, interesting blog posts : https://uhasct.com/create-a-legend-for-custom-conditional-formatting-in-excel/
https://policyviz.com/2017/04/27/creating-heatmap-legend-excel/
use CSV(Common deliminated) (*.csv) often when saving excel as .csv, when to import into Python, R

*Cant have commas in data if using CSV

###Unix shell: http://swcarpentry.github.io/shell-novice/
##accessing:
Linux
The default Unix Shell for Linux operating systems is usually Bash. On most versions of Linux, it is accessible by running the (Gnome) Terminal or (KDE) Konsole or xterm, which can be found via the applications menu or the search bar. If your machine is set up to use something other than bash, you can run it by opening a terminal and typing bash.
macOS
For a Mac computer, the default Unix Shell is Bash, and it is available via the Terminal Utilities program within your Applications folder.
To open Terminal, try one or both of the following:

- Go to your Applications. Within Applications, open the Utilities folder. Locate Terminal in the Utilities folder and open it.
- Use the Mac ‘Spotlight’ computer search function. Search for: Terminal and press Return.

windows
Install Git for Windows.https://gitforwindows.org
We will use Bash
http://swcarpentry.github.io/shell-novice/

Data: https://swcarpentry.github.io/shell-novice/setup.html
Download data-shell.zip and move the file to your Desktop.
UNIX command (command followed by options and then arguments). Options start with a - or -- and then followed by arguments.
-
--
CASE sensitive
dir=directory
ls = prints the names of the files and directories in the current directory.
cd= change directory
pwd= path to working directory
man= manual

You have to specify file paths relative to position of your working directory in order to access files not located inyour working directory.

absolute part vs relative part - absolute part is the whole path for a file; relative path is shorter

To check which working directory you are in, you type
pwd and hit enter
pwd = print working directory

To change your directory, you type
cd
cd = change directory

Remember that the shell is case sensitive! So if your folder is called Documents with a capital D, you need to type it exactly like that!

To see all the files in a folder, you type
ls
ls = list

If you add the option -l (this is a dash and a lower case letter L), you get a list of your files with more information

To get information or help on a command, you type
the command and --help, for instance
ls --help
It will give a you a long list of options you can add (the dash plus a letter part).

To get out of the help: hit the letter q

On Macs you can also type
man ls
Here you can find more information. man = manual

If you mistype a command - shell will say so.

ali as ll= 'ls -l' gives more infor; ll -h
alias shows alias to the directories - using ll (or ls -l) gives details to the documents
cd . current directory
To go up a directory (folder), type
cd ..

To go back to your home directory, type
cd
(without anything added to it)

To get back to the directory where you were just before the last command, type
cd -

To make a new directory:
mkdir play
You have now created a new directory (mkdir = make directory) called "play",.

Check it is there with ls
Go to the directory with cd play
We will create a text file in this directory

There are text editors you can use to make text files from the command line:
- nano
- vim
We will use nano today. Nano is a text editor.
Type in
nano
at the command line and a blank text file will open from your command line

Use control + O to save your file

Recommended: not to use spaces or special symbols in your file names.
Do not forget to add the extension .txt to your file name
To get out of nano, type control + X

To check your file, number of words in it etc:
wc first.txt
(wc = word count)
To get more information to you file: wc followed by file name
The three numbers given by the wc command, for example, 3 3 15, mean that there are 3 lines, 3 words, and 15 characters total

To open your file again
nano first.txt

To copy a file, use the command
cp
For instance
cp first.txt firstbak.txt
The first file name is the file you are copying, the second file name is the copy

To create an empty file, use the command
touch
For instance
touch second.txt

Then check with ls -l that there is indeed a second.txt file created (as well as the file firstbak.txt that we created when we copied first.txt)

To remove a file, use the command
rm
For instance
rm second.txt
This will delete the file second.txt (don't do this just yet)

touch creates an empty file.

wc * , get information of all the files in current directory

The star * is called a "wild card", it means to use all files that are in that directory. So not just apply wc to one file but apply it to all the files in the directory
Another wild card is the question mark, ?
It means that in a file name, where the question mark is, there could be any one character. A star is for any character (no limit)
So when you use
wc ?i*
You will get word count for the files that fit the description:
- any character as the first character (?)
- an i as the second character
- anything after the i (*)
So in this case you will get information from the files first.txt and firstbak.txt, but NOT second.txt

wc fi* gives you the same result as above

wc [a-f]* will give you all files that start with a,b,c,d,e or f and anything (*) after that letter
wc [a-f,s]* will give you all files that start with a,b,c,d,e, f or s and anything (*) after that letter (so in this case you will also see the word count for the file second.txt)
wc [a-f,s]*, added file start with "s"

Remove second.txt: once you remove the file, the fil is gone.

If you want to rename y afile, the command is mv - which states for move.
- You can move the file to another directory.
The first file in the command is the original file (location), the second is where you are moving it to
- You can also use this command to rename the file.
For instance: mv second.txt empty.txt (this renames the file, it stays in the same directory)

Now to move this file:
mv empty.txt tmp/veryempty.txt
(tmp is your temporary file directory)

The cat command can be used for two things:
- You can concatenate two files
- You can use the command to pull up the contents of a file. Yu can also use the command 'less' to display the content of a file.

cat first.txt will show you the text in your file on the screen (in the command line, it will not open nano)
BUT
If you use the cat command with a greater sign: cat > first.txt - it will overwrite the data. !!! SO BE CAREFUL
cat >> first.txt will add text to the existing text
Use control + z to get out of typing text when you use cat > or cat >>

less first.txt also shows you the text (use q to get out)
press q to go back to command line

To find a particulare file:
find . -name "hai*"
find . -name hai*
to find the file haiku.txt

This is in the directory called "writing"
cd writing
ls -l
cat haiku.txt

To look for a line with a certain word or letter in a text file, use the command
grep

For instance:
grep the haiku.text
(command, the word you are looking for, the file you want to search)
Remember: the shell is case sensitive! We did not find the word "The" with a capital t

grep -i the haiku.txt
The addition of the option "-i" will turn off case sensitivity

Other options:
grep -i -w the haiku.txt (-w = word search)
grep -i -w -v the haiku.txt (-v = invert match)
You can write the last command as grep -iwv the haiku.txt

if you want the top/beginning of the text file: you can use the command head and if you do
head -n3 haiku.txt - it will display the first three lines of the haiku text file.

If you want to avoid the loss of information
You can rename the command by making an alias. For example remove:
alias rm='rm-i' (no spaces in the last part) #MacOS>> alias rm='rm -i'
This means that next time you type in rm your shell will be executing rm -i
This is a particularly useful example because this way you are making the remove command a lot safer!
The option -i will ask you if you really want to remove the file.
So now, every time you use rm, you will get the question if you really want to remove the file

Even safer: alias rm='mv'
Because now instead of removing, you're only moving the file!

set -o | grep clobber
this turns on "noclobber" which protects you from overwriting files
cat empty.txt > first.txt

As for permissions for specific files, this is shown in the first column if you type in ls -l
d = if it's a directory
r = reading
w = writing
x = execute

(the first four are for your computer (root), the second set of four for the group owner (if you work with a group), the third set for everyone else
See for more information: http://linuxcommand.org/lc3_lts0090.php (also shows you how to use chmod to change permissions)

To save everything we did today:
history > session1.txt

For more commands and tricks, see http://swcarpentry.github.io/shell-novice/

LUNCH!

######### Introduction to R ###########

We will use Rstudio to learn R.

Create a project (Rproj) that we will work in. This makes sure all your code and data stays together.

Organizing your working directory. If you want to create a new folder in your directory, under the files tab you can add 'new folder'.

#code for installing packages
install.packages(c("DBI", "dbplyr", "dplyr", "RSQLite", "tidyverse"))

getwd = get working directory
setwd("C:/Users/...") = set working directory to....

Window on top in RStudio can save script, window on bottom (console) will not

Commands can be run by pressing Ctrl+Enter
you can also run ocmmands by pressing 'Run' in the top right

?+function = what it does in the 'help' tab (ex. ?getwd)
You can also search in the help tab

Google is your friend when it comes to searching for help with R
https://datacarpentry.org/R-ecology-lesson/00-before-we-start.html#getting_set_up

You can assign values to objects by using <-
So
weight_kg <- 17
means that you assign the value 17 to the name weight_kg
If you type in weight_kg, you'll get 17 displayed in the console

You can do calculations in R, e.g. 12/7 will give the result for 12 divided by 7

You can also do calculations with the objects:
    weight_lb <- 2.2 * weight_kg

will give you the result of 2.2 * 17 and assign it to weight_lb

R will calculate everything in order. So if you now do
weight_kg <- 100
this will NOT update the value for weight_lb
You will have to run that line of code again to update the value of weight_lb

Challenge
What are the values after each statement in the following?

```
mass <- 47.5            # mass?
age  <- 122             # age?
mass <- mass * 2.0      # mass?
age  <- age - 20        # age?
mass_index <- mass/age  # mass_index?
```

Functions
There are existing function in R to make certain calculations easier.
Functions always have () behind them, this is where you specify the arguments you give the function. This can be an object, but sometimes functions have certain
settings that you can change.
You can find the arguments existing function take by typing
?sqrt (the function for the square root)
?round (the function for rounding)
The window on the bottom right, under the tab "help" will show the details of the function
Some basic functions:
Square root: sqrt()
Rounding the decimal to nearest whole number: round(object, desired no. of decimal places)

A vector stores a number of "things": numbers, character strings, etc
You make a vector by using the function c()
weight_g <- c(50, 60, 65, 82)
is a vector object called weight_g that holds 4 numbers
animal <- c("mouse", "rate", "dog")
To see how long a vector is - how many numbers or strings:
length(weight_g)
To see the class of the weight, whether it's numbers of characters for instance:
class(animals)
To see the structure of a vector (the class, the number of entries in a vector):
str(animals)
str(weight_g)

You can add to an existing vector by simply using c() on it:
weight_g <- c(weight_g, 90) this adds a number at the end
weight_g <- c(30, weight_g) this adds a number at the beginning

Challenge: what happens if you mix classes / data types in a vector?
Answer: R will force them into one data type

The five basic classes are:

- logical (e.g., TRUE, FALSE)
- integer (e.g,, 2L, as.integer(3))
- numeric (real or decimal) (e.g, 2, 2.0, pi)
- complex (e.g, 1 + 0i, 1 + 4i)
- character (e.g, "a", "swc")

Subsetting and indexing

You can use square brackets [ ] to subset a vector (select only few values), to subset only first value of the vector 'a', you can type a.1<- a[1]

You can also select more than one value or vector entry, by putting a vector inside the square brackets:

animals[2] only give you the second entry

animals[c(2,3)] give you the second and third entry, the vector you are putting inside the square brackets is c(2,3)

If you do

more_animals <- animals[c(1,2,3,2,1,4)]

you are creating a new vector called new_animals that you are populating with entries from the vector animals()

you can use logical vector string to subset your vector as well

weight_g<- c(21,31,34,39)

weight_g_subset<- weight_g[c(TRUE, FALSE,TRUE,TRUE)]

weight_g_subset = 21,34,39

##

to state a condition of OR, use '|' ; to state a condition of AND , use '&'

You can use %in% to check the condition if you vector is nested within another vector.

Missing values

If you have NAs (missing values) in your data, if you have NA value then some functions like mean and sd (standard deciation) would not work. So for your vector, a<-c(2,3,4,5,6,NA), you can use mean(a, na.rm=T). It does not remove na from the dataset but just removes it from the calculation.

To check vectors for NAs, there are several options:

is.na(heights) This will give you a logical vector with TRUE for NA, and FALSE if it not NA

na.omit(heights) This will take the NAs out

complete.cases(heights) This will give you a logical vector with TRUE for if it is not NA, and FALSE it is NA

Challenge

1. Using this vector of heights in inches, create a new vector, heights_no_na, with the NAs removed.

heights <- **c**(63, 69, 60, 65, NA, 68, 61, 70, 61, 59, 64, 69, 63, 63, NA, 72, 65, 64, 70, 63, 65)

1. Use the function median() to calculate the median of the heights vector.
2. Use R to figure out how many people in the set are taller than 67 inches.

Reminder about indexing:

R always start with 1

(other languages, like Python, will start with 0 for the first entry)

BREAK

Starting with data

https://datacarpentry.org/R-ecology-lesson/02-starting-with-data.html

We wil work with an existing file, that you can download from
https://ndownloader.figshare.com/file/2292169

You can also do this directly in RStudio:

download.file(url="https://ndownloader.figshare.com/files/2292169", destfile =
"data_raw/portal_data_joined.csv")
url tells R where to donwload the data from
destfile tells R where to store the data (which is the directory data_raw that we have in our project

To read the data into RStudio, use the command read.csv()
surveys <- read.csv("data_raw/portal_data_joined.csv")

The data is now loaded into R as a "data frame" i.e. data.frame

If you type in surveys you will get all the data displayed in the console - which is a lot
head(surveys) gives you only the first 6 records of your data

Check the structure of your data:
str(surveys)

More ways to get information on your data:

dim(surveys)	get the dimensions
nrow(surveys)	number of rows
ncol(surveys)	number of columns

Challenge
Based on the output of str(surveys), can you answer the following questions?

- What is the class of the object surveys?
- How many rows and how many columns are in this object?
- How many species have been recorded during these surveys?

To subset from data.frames, you can use indices, just like in vectors, as we did earlier, but in 2D:
subsetting in data.frames is [rows, columns]:
surveys[1, 6] gives you the values in row 1, column 6
surveys[1,] Now you're only selecting th first row NOTE: if there is nothing before or after the comma, it means you are selecting all rows (before the comma) or all columns (after the comma).
Because a data.frame is 2-dimensional (rows x columns) you must ALWAY use the comma.

surveys[1:6,] This gives the first 6 rows

surveys[, -1] With the minus sign, you're removing. So here you are getting the whole data.frame MINUS the first column
surveys[-c(7:34786),] Here you are using a vector to remove rows (rows 7 through 34786 to be precise)

And remember: if you're not assigning any of this to a new object, you are not altering the original data.frame, and you're not storing this. You're just looking at it.

But you can also use the column names to select the column you want:
surveys["species_id"] This will give you a data.frame
surveys[, "species_id"] ---> note the comma, to show you are selecting a column
surveys[["species_id"]]
surveys$species_id --> these last 3 options give you vectors.

Challenge
1. Create a data.frame (surveys_200) containing only the data in row 200 of the surveys dataset.
2. Notice how nrow() gave you the number of rows in a data.frame?

- Use that number to pull out just that last row in the data frame.
- Compare that with what you see as the last row using tail() to make sure it's meeting expectations.
- Pull out that last row using nrow() instead of the row number.
- Create a new data frame (surveys_last) from that last row.

3. Use nrow() to extract the row that is in the middle of the data frame. Store the content of this row in an object named surveys_middle.
4. Combine nrow() with the - notation above to reproduce the behavior of head(surveys), keeping just the first through 6th rows of the surveys dataset.

Challenge: Solutions
1. surveys_200<-surveys[200,]
2. n_rows<- nrow(surveys)
surveys_last<-surveys[n_rows,]
3. surveys_middle<-surveys[n_row/2,]
4. surveys_head<-surveys[-(7:n_rows),] deleted all the rows from 7 to the last column using the minus sign

We have many factors in the structure of our data
species_id is a factor with 48 levels

- for any categories you have, you want them to be factors, rather than characters, makes it easier to run categorical data analysis in R

want to know how to identify what format your data is in, and can convert between types
converting between types is one point where we can make mistakes

sex<-factor(c("male", "female", "female", "male")) # don't necessarily have to put four of these, this is just creating some test data for us to use
We have four observations, with two levels
have used factor() to create a vector with two different levels: female, male

can see what levels are assigned using levels(sex)
nlevels(sex) #checks how many levels you have

In some cases, we want to have our factors in a particular order
sex<-factor(sex, levels=c("male", "female")) # levels() can be used to assign order for the levels
What happens if you have too few or too little levels when organizing? Often you will get an NA

Use str() to check the structure of your data, you can use this to look for typos, issues with factor assignments etc.

How do we convert between data types?

as.character(sex) #will convert your factor to a character
this will give you: "male" "female" "female" "male"

in some cases, you may want a numeric value; for example, a year or Julian date can be either a factor or continuous variable (depending on how you want to analyze your data)

as.numeric() used on factors will return the index values, not your factors

year_fct<-factor(c(1990, 1983, 1977, 1998, 1990))
as.numeric(year.fct) will give you 3 2 1 4 3
so we need to first convert to a character first
as.character(year.fct) #will convert years to characters, you can tell when they are characters because they will show up in your console within " double quotes "
two options:
(1) as.numeric(as.character(year_fct)) # this will give you years as numeric values
(2) as.numeric(levels(year_fct))[year_fct] # this will also give you years as numeric values as well, this is recommended by the carpentries, but both (1) and (2) will let you convert years from factor to numeric

- the square brackets are used to subset in option (2)

To plot things, can use either base R or the package ggplot2 (which we'll learn tomorrow - it's an awesome tool)
plot(survey$sex)
this will show you that some of the values are blank (they plot on the x axis that there are blank, M, and F)
we want to change the labels so the blank values are renamed

sex<-surveys$sex #let's pull out the sex data to check what's going on

head(sex) #this will let us look at the data
levels(sex) #check what levels you have, you can see that we have a "" blank quote as a level
levels(sex)[1] <- "undetermined" # this will rename the blank factor as undetermined

- # The [1] says that you want the first factor to be renamed as "undetermined"

Challenge:
1. Rename "F" and "M" to "female" and "male" respectively
2. Now that we have renamed the factor level to "undetermined", can you recreate the barplot such that

"undetermined" is last (after "Male")

Challenge Solutions:
1. levels(sex)[2:3]<-c("female", "male") if you had other values you wanted to change, you could do 2:4, etc.
2. sex<-factor(sex, levels=c("female", "male", "undetermined"))
plot(sex)

When reading into csv files, your factors may be coerced (automatically turned into) characters
surveys<-read.csv("data_raw/portal_data_joined.csv", stringsAsFactors=TRUE) #Character strings are going to be factors by default
but you can change this so stringsAsFactors is FALSE (shown below), and when the .csv is imported, it will be imported as a character
surveys<-read.csv("data_raw/portal_data_joined.csv", stringsAsFactors=FALSE)

We might want to change the format of the plot_type to be a factor, so you can use the factor() function to convert it into a factor
surveys$plot_type<-factor(survey$plot_type)
now plot_type is a factor with 5 levels, but everything else is still a character (because we imported our .csv using stringsAsFactors=FALSE)

Loading packages that are already pre-developed
if you don't have the package "lubridate" installed, you'll get an error message, then you can install the package
install.packages("lubridate")
once it's installed, you can load it using
library(lubridate)
it will notify you (in red) in your console, that it is "attaching package "lubridate"" which tells you that the package is ready to use
Note: you don't have to install a package every time you use it, but you do need to load it using library() when you want to use it (only need to do it once during a session)

The lubridate pakcage has a function ymd that can be useful
my_date<-ymd("2015-01-01")
my_date<-ymd(paste("2015", "1", "1", sep="-")) #this will give you the same output as the line above, but uses the fuction paste, which is really useful.
paste will take whatever is in the "quotes", paste them together, using the defined separator, sep="-"

my_date<-ymd(paste(surveys$year, surveys$mothn, surveys$day, sep="-")
warnings are useful, but sometimes you can just ignore them
errors you can't ignore, stops the line from running

The date column does not exist yet, but we would like to create one by adding it in with $date, this will append to the last column of your data
surveys$date<-ymd(paste(surveys$year, surveys$mothn, surveys$day, sep="-")

summary(surveys$date) #this will let you check out the new column that you created
but it looks like some of the data are NA...why is that?

Let's check it out and figure out what the issue is:
missing_dates<-surveys[is.na(surveys$date), c("year", "month", "day")]
this will show you the na values that are in your dataset
The issue is that the dates are impossible, someone must have imported it incorrectly)

########################## End of Day 1

Tuesday:
morning: Version control with Git (led by Brian)
afternoon: Data manipulation and visualization with R (led by Geraldine)

Slides from Dimitri for the UNIX shell lesson: https://drive.google.com/open?id=1sa_oe0XdXm-RM5CMjFCnpW23LOSAc2_I

Day 2

Version control with Git

If you have and work with multiple versions of the same file (e.g., final3_final2_final_Updated_comments_v4.R) and you think you will know what file that is while you are just about to create final figures for that one paper that has dragged on for many year- good luck! This is where Git is a life saver.

Git is a software system that automatically handles all the problems we talked about. It wraps one file up in a single file.
There are different kinds of version control software and Git is just an example of it. It was written by creator of Linux
Github:
Git:

General Idea of how Git works:
Check out files from respository-> make changes to those files -> commit files to respository-> cycle continues!
Repository: where all files live (typically organized by project, you can have many respositories!)
Commit: Pushes changes that you have made to files back into the respository
#Commands
git config --list #configuration settings
git config --global user.name "insert your username" # to change your username, you can choose whatever you would like
git config --global user.email "insert e-mail" #to change e-mail address
git config --global color.ui "auto" #display the output in colors so we can intrepret what the output means
git config --list # to see changed configuration settings

mdir recipes #make a new directory for your repository

ls #confirm you have made the new directory
cd recipes #change working directory to recipes
ls #see inside recipes (should be empty right now!)
git initi #make recipes your respository (this must be done when you are in the folder you want to make the respositorty)

- command response should be: Initialized empty Git repository in /Users/your_comp_name/recipes/.git/

ls -a #show everything in the folder, you should now see the directory
git status #shows no commits yet, remember how we utilize the repository by commiting files to the respository
Usually any command you want to pass has the following structure:

- git 'command' [arguement]
- sometimes its takes other arguement ~ like the config command

nano #text editor to make a new file
You can use any other text editor as well, Git does not care
** Pizza **
pizza dough
pizza sauce
green peppers
mushrooms
pineapple

Control O # to save the file we just created
pizza.txt #give it a name
Control X #to leave the text editor

ls # check to make sure the pizza file is there
cat pizza.txt #show the file within the terminal
git status # still no commits yet
git add pizza.txt #to add pizza.txt to repository
git status #now we see that a new file, pizza.txt, has been added but there are no commits, but changes are ready to be committed
git commit -m "Initial version of pizza recipe." #commit the pizza file to the respository and include a message associated with that file using the -m command

- 1 file changed, 7 insertions(+) ## commit message is echoed back to you and tells you what was changed since the last version

If you get:
warning: LF will be replaced by CRLF in pizza.txt.
The file will have its original line endings in your working directory
This means that a line ending has been detected and that git will change it for a more compatible one.

It is a two step process to push changes to a new file in git, you have to follow these steps:
git add myfile.txt
git commit -m "my recent changes"

nano pizza.txt #lets include instructions

- Instructions:
- 1. Spread dough evenly
- 2. Add sauce and toppings

Control O #to save file
Control X #to exit file

nano

- Make pizza sauce reciepe

Control O # to save
save as sauce.txt #save your file
Control X #to exit

ls #check to see if sauce is in the folder
cat sauce.txt #view the text inside terminal
git status
git add pizza.txt sauce.txt
git status #two files are ready to be committed
git commit -m "added sauce recipe and instructions." #commit these files to the repository

- 2 files changed, 13 insertiosn(+), 1 deletedion(-)
- create mode 100644 sauce.txt

git status #git has tracked all of those changes, nothing to commit

Let's see how we can view how our files have changed over time!
nano pizza.txt

- add mozzarella to the list of ingredients
- and add 3) Sprinkle cheese over time

Control O #to leave save
Control X #to leave text editor

git diff pizza.txt #show me the difference between current verison of a file and last verison stored in the repository
git log #shows the commit history of your respository, shows identifier (a unique ID to each commit), also shows username and time of when committ occured. You can use the identifier (or first few digits) with diff command to see the differenced between the two versions

git diff ##### pizz.txt # include the unique identifier in the #### to compare the differences associated with current verison of your file to the one you have identified in the repository
git diff ##### #### pizza.txt # can compare two verisons of the script using the two identifiers rather than comparing against your current verison of the file
git diff HEAD pizza.txt #instead of using identifier for the last commit, use HEAD
git diff HEAD~1 pizza.txt #two verisons ago
git diff HEAD HEAD~1 pizza.txt #compare last verison with second to last verison

- If you are going way back in time, use the unique identifier. BUT, if you are interested in compare your current verison with the previous verison by using HEAD

git log #also shows the messages you have included, like a content tag to help you understand what you did in that last commit.

git add pizza.txt #add file to the staging area
git commit -m "Added cheese!" #commit to the repositoryand added message
git status #nothing commit

git diff pizz.txt #nothing shows up...the current verison of the file is being compared to what was last saved in the repository, which is the same! Nothing is different!

nano pizza.txt

- deleted work

Control O
Control X

cat pizza.txt
git checkout HEAD pizza.txt #go back and retrieve and replace the current file with the one I have selected using checkout and HEAD (the last file in repository)
cat pizza.txt #file is back to the previous verison

rm pizza.txt #deletes files
ls # pizza.txt is gone!
git checkout HEAD pizza.txt #restore file from the last committed file in the repository
ls #file is back!
cat pizza.txt

git log --name-status # get more information. tells which files were modified (M) and which file was added (A)

Git works with all types of files, not only with text files. Images and other may not be displayed but the mechanics of the repository is maintained.

~Morning Break~

Introducing a remote repository for data protection

Synchronizing with a remote copy of the repository uses git push to send data to the repository and git pull to pull down data from the repository

github.com Create an account or sign in if you already have one
Create new repository called recipes

Now, we have a remote repository created on GitHub. To link your computer to this remote repository, you can copy the URL address in Quick setup: http://github.com/.....

Go to Terminal
#Commands: **Only have to do this one time**

git remote add origin (paste the copied information from the GitHub quick setup)
git remote #add the remote location
git remote -v # confirms the address of added remote repository
git push -u origin master #telling git that we want to push contents to remote repository (origin); -u: automatically push to remote repository; need to include your GitHub username

Back to GitHub Webpage: refresh
You should see pizza.txt and sauce.txt in GitHub repository

Back to Terminal:
Commands:
nano pizza.txt

- add instruction 4. Dice green peppers; sprinkle on top
- Control O
- Control X

git status # pizza.txt has been modified
git add pizza.txt # Add to staging area
git commit -m "Added green pepper instructions" # commit to repository
git status #everything is up-to-date but "Your branch is ahead of 'origin/master' by 1 commit" means your local repository is ahead of remote, need to match them
git push # push the changes to remote repository

You can synchronize as often as you want! You can synchronize each week, month, it is up to you!

rm -rf recipes/ #deletes repository
cd recipes # No repository available

Go to GitHub-> go to your repository -> Clone or download
Copy the link

Go to Terminal:
git clone (copy that directory) # restore/clone that repository
cd recipes/ # ah! It's back on your local repository
git log #to see commit history
git push # Shows that everything is up-to-date

Colloboration: Working with other people on shared files. Individuals can commit files asynchronous
Select one person who has the shared remote:
USER A:

- invite colloborator to share access: Click on settings -> colloborators -> input username of colloborator-> add colloborator

USER B: Check e-mail to accept colloborator

- Clone/download the repository

Terminal:
git clone (insert the copied repository)
ls #check to see the files
cd recipes

nano pizza.txt

- make somesort of changes
- git add pizza.txt
- git commit -m "changed recipe and instructions."
- git push

USER A:

nano pizza.txt #make changes to the pizza.txt file

- git add pizza.txt #
- git commit -m "added red peppers and added 5th instruction."
- git push
    - WARNING!!! Rejected

User A;

- git add pizza.txt
- git commit -m "Resolved conflict with Geraldine's edits."
- git push
- git log #showcases the committed changes

info about git and Rstudio http://swcarpentry.github.io/git-novice/14-supplemental-rstudio/index.html
there is info there about finding the git executable if RStudio doesn't recognize it automatically

~ LUNCH ~

Data management and visualization with R

Advantages of working in a Project:
-This is a working directory, Your colloborator doesn't have to change anything (i.e. working directories, calling files) and has all the files needed to run the code you have
- Can run different pieces of code at the same time using different projects

You need to have R downloaded on your computer to use RStudio

Open up a script using the paper with a plus icon or File -> New File -> R Script

Commands:
Install packages:
install.packages("tidyverse") #to download package to your computer
library(tidyverse) # to activate the package
Tidyverse is a number of packages (dplyr, readr, ggplot2, etc)

surveys<- read_csv("data_raw/portal_data_joined.csv")
str(surveys) #Structure of data

View(surveys) # view your dataframe in a seperate tab . You can also do this by clicking on "surveys" in the Environment Tab on the right hand side of RStudio

Subsetting;
select(surveys, plot_id, species_id, weight) #selecting for specific columns
select(surveys, -record_id, -species_id) # can select using the minus sign. We don't want record_id and species_id
filter(surveys, year== 1995) #filtering all the data for the year that equals (==) 1995

Old way of subsetting:
survey2<-filter(surveys, weight<5)
surveys_small<-select(surveys, species_id, sex, weight)

Can subset using pipes! %>%
Shortcut on windows: Control Shift M
Shortcut on Macs:

surveys_sml<-surveys %>% #Work Flow (1) Using the survey data

- filter(weight < 5) %>% # (2) Filter that dataframe to have weights less than 5
- select(species_id, sex, weight) # (3) Of those observations remaining, only produce the columns species_id, sex, and weight

View(surveys_sml)

Challenge:
#Find only the animals collected in 1995 using a tibble or dataframe and only retain the columns year, sex, and weight

surveys_1995<- surveys %>%

- filter(year==1995)%>%
- select(year, sex, weight)

View(surveys_1995)

Introducing mutate.

- Mutate is a function that allows you to mutate or calculate something and make a new column

surveys %>%

- mutate(weight_kg= weight/1000) #make a new column which transforms weight from grams into kilograms

surveys_kg<- surveys %>%

- mutate(weight_kg=weight/1000,
- weight_lb= weight_kg* 2.2)

What to do with NA's? Order is very important!
surveys %.>%

- filter(!is.na(weight)) #removes rows that have NA in weight column
- mutate(weight_kg= weight/1000)

Challenge:

create a dataframe/tibble with only the species id column and a new column called hindfoot_half. hindfoot_half contains values that are half the length of hindfood. No NAs in hindfoot and that all values are less than 30.

Hint: Remember order of piping matters!

Answer:
surveys_hindfoot_half<- surveys %>%

- filter(!is.na(hindfoot) %>%
- mutate(hindfoot_half= hindfoot/2) %>%
- filter(hindfoot_half<30) %>%
- select(species_id, hindfoot_half)

View(surveys_hindfoot_half)

Introducing group_by and summarise:
Allows you to group your data and summarize information
surveys %>%

- group_by(sex)%>%
- summarise(mean_weight= mean(weight, na.rm=TRUE)) #remove all the NA's

surveys %>%

- group_by(sex, species_ID) %>% #Group by two groups sex and species
- summarise(mean_weight= mean(weight, na.rm=TRUE))

survey_group<- surveys $>$

- filter(!is.na(weight)) %>%
- group_by(sex, species_ID) %>% #Group by two groups sex and species
- summarise(mean_weight= mean(weight)
    - min_weight=min(weight))

survey_group<- surveys $>$

- filter(!is.na(weight)) %>%
- group_by(sex, species_ID) %>% #Group by two groups sex andspecies
- summarise(mean_weight= mean(weight)
    - min_weight=min(weight)) %>%
    - arrange(desc(mean_weight)) #arrange the order in a descending weight

surveys %>%

- count(sex) #another way to group by and count the number of observations within each sex, this is a built in function

could also "count" doing this way:

```
surveys %>%
group_by(sex)  %>%
summarize(count=n())
```

Count by sex and species!
surveys %>%

- count(sex, species)

Let's order these by species and descending order of number of observations
surveys %>%

- count(sex, species)
- arrange(species, desc(n)) #arranged by species and then by count

Challenges:
1. How many animals were caught in each plot_type that was surveyed?
Answer:

- surveys %>%
    - count(plot_type)

2. Use group_by() and summarize() to find the mean, min, max hindfoot lengths for each species (use species_id). Also add the # of observations.
Answer:

- surveys %>%
    - filter(!is.na(hindfoot_length)) %>%
    - group_by(species_id) %>%
    - summarize(mean_hindfoot=mean(hindfoot.length),
        - min_hindfoot=min(hindfoot.length),
        - max_hindfoot=max(hindfoot.length),
        - count=n())

note: you don't have to put the mean, min, max on their own lines, but it makes it much easier to read the code

Gather and spread in R - two functions in R
However, these functions are on their way out, they're trying to make it more intuitive
From the developer: "names are not intuitive...surprisingly hard to remember the arguments"
It can be confusing! but they're working on making it easier to understand

if we want to reshape the data to explore other aspects of our data, how do we do it?

mean weight of all animals from each species between plots
conceptually, we want a table where we have:

Column names: PlotID, NL, CM...species name
with cells filled with species mean weight for each plot

When you spread, makes a wider column, spread out the data so genera are spread and each genus is a column
spread(data, key, value, fill=NA)
key is the column variable which values will become the new columns (in this case it is specie)

#Commands:
to spread this out, we will create a new dataframe names surveys_gw, which we'll use to get the mean weight per genus, and then we'll spread surveys_gw to get it in the format we want
surveys_gw<- surveys %>%

- filter(!is.na(weight)) %>% #get rid of NA values
- group_by(genus, plot_id) %>% #grouping by genus and plot id because we want to get the mean weight per genus
- summarize(mean_weight= mean(weight))

View(surveys_gw)

surveys_spread <- surveys_gw %>% #use the surveys_gw dataframe as our input

- spread(key=genus, value=mean_weight, fill=0) #functions take arguments, the "data" argument is the surveys_gw "coming out" of the pipe

Note: when you use a pipe, the first argument of the function *after* the pipe is automatically set to the data coming "out" of the pipe.
be careful when manipulating data like this
To change it back to "long" format:
survey_gather <- surveys_spread %>% gather(key=genus, value=mean_weight, -plot_id) #have -plot_id because we want to maintain the plot_id, we just don't want to include it in our gather function, but will still be in our final tibble

In spread you tell it which columns to look at when you spread, but with gather, it starts reading from the first column, if we didn't have -plot_id, then plot_id would show up under the genus column

Can also select certain genera that you want to include
survey_select <- surveys_spread %>% gather(key=genus, value=mean_weight, Baiomys:Spermophilus)
so we gathered it back, but didn't use all of the genera to gather

Challenge:
Spread the surveys data with year as columns, plot_id as rows, and the number of genera per plot as the values.
Solution:
Need to summarize() the data first using n_distinct() to get the counts of genera per plot.
rich_time<- surveys %>%

- group_by(year, plot_id) %>%
  - summarize(n_genera=n_distinct(genus)) %>%
  - spread(year, n_genera)

rich_time <- group_by(surveys, year, plot_id) %>%

- summarize(n_genera=n_distinct(genus)) %>%
- spread(key=year, value=n_genera)

note: n_distinct(genus) will give you the number of unique genera - it counts the different genera that there were, not the number of animals...if you wanted to count the number of animals you would use the function n()

Make this dataset and save it, so we can plot the data in the later part of this afternoon:

First: clean up the dataset, remove NAs
surveys_complete <- surveys %>%

- filter(!is.na(weight),
    - !is.na(hindfoot_length),
    - !is.na(sex))

Then get the most common species:
species_counts <- surveys_complete %>%

- count(species_id) %>%
- filter(n >= 50)

Overriding the surveys_complete dataframe that we made before!
# Filter out all species with fewer than 50 observations.
surveys_complete <- filter(surveys_complete, species_id %in% species_counts$species_id)

remember that $ refers to a column
this way has us filtering in a nested way

Save the dataframe that we just created:
write_csv(surveys_complete, "data/surveys_complete.csv") #first argument is the argument of the dataframe, second argument is where to save the dataframe and what to call it

~ Afternoon Break ~

If you work in a project with several scripts, may want to export your cleaned up data (using write_csv) and then load your cleaned up data (using read_csv) at a later date for plotting
library(ggplot2)
gg = grammar of graphics; there is a specific logic in plotting with this package, it is very organized and plots are built in layers, much more customizable

However, you can do plotting in basic R as well (without ggplot2), but quite a bit more limited
plot(survey_complete$weight, survey_complete$hindfoot)

If you want to have a quick look at the data, base R is totally fine to use, but if you want to make publication-worthy plots, ggplot2 is most useful, easy to remove/add things to plots in ggplot

Basic order of things:
#you can omit the data = and mapping = if you'd like; makes it a little easier to type

ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length)) + # tells ggplot what data you are pulling from and what x and y axes

- geom_point() #what type of plot you want

This will give you a plot with points

aes stands for aesthetics, how you want things to look, however when you run just the ggplot with mapping, nothing plots...why would that be?
it's becuause we didn't specify what type of plot we want, so we add + and other layers to specify plot type, etc.

Some geometry options (very extensive):
geom_point()
geom_line()
geom_boxplot()
etc.

Can save the figure the same way as we would save an object, note that nothing will show up initially
surveys_plot<- ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length)) +

- geom_point()

surveys_plot # now the plot will appear

The way you add layers to your plot is by adding a + sign at the end of your line

However, if you have a trailing +, then R will try to go to the next line, will give you an error
In your console, > means that R is ready and waiting for your command, but if a + is shown in your console, it means R is waiting for further instruction

You can use the plot object that you've created and start adding layer to it, like below
surveys_plot +

- geom_point()

For the rest of the lesson, we'll use the full ggplot syntax, so you get familiar with the syntax

ggplot(data=survey_complete, maaping= aes(x=weight, y=hindfoot_length))+

- geom_point(alpha=0.1, color="blue")

alpha indicates the transparency of your geometry (in this example, the transparency of your points)
color is the color of your geometry, make sure your color is in double quotes ""

In the plot viewer, you can scroll back through old plots that you've made
You can also use the little broom in the top right corner of the plot viewer to clean up/remove a plot

We can use the color to distinguish between different species_id, set it so that color will be based on the species_id...ggplot2 will use its default colors to give a different color to each species, note that the species_id is not in quotes ""
ggplot(data=survey_complete, maaping= aes(x=weight, y=hindfoot_length))+

- geom_point(alpha=0.1, aes(color=species_id))

We can also define the color earlier on in our code, within the aes() parentheses, the first line of your plot is the global settings for your plot, if you want everything you do to show the same colors based on species_id, then specify it in the global environment; the first line is the "master" settings

ggplot(data=survey_complete, maaping= aes(x=weight, y=hindfoot_length, color=species_id))+

- geom_point(alpha=0.1)

if you add the color id later, it can override the original color

A lot of the points are overlapping, so we can use geom_jitter() to make the data a little more visible; moves the data slightly to minimize overlapping, can be useful, especially with columns

Challenge:
create a scatter plot of weight over species_id with the plot types showing in different colors
How do you feel about this plot? The plot_type is a categorical variable, so there are better ways to show it
Answer:

- ggplot(surveys_complete, aes(x=species, y=weight))+
    - geom_point(aes(color=plot_type))
- Not the best way to show this data

Let's make a box plot instead!
ggplot(surveys_complete, aes(x=species, y=weight))+

- geom_boxplot()+
- geom_jitter (alpha=0.3, color="tomato")

Remember, we are layering when we use ggplot. Let's see what happens when we change the order!
ggplot(surveys_complete, aes(x=species, y=weight))+

- geom_jitter (alpha=0.3, color="tomato")+
- geom_boxplot()

Let's make the boxplot completely transparent. We do this through alpha. Alpha=0 completely see through, alpha=1 solid white
ggplot(surveys_complete, aes(x=species, y=weight))+

- geom_jitter (alpha=0.3, color="tomato")+
- geom_boxplot(alpha=0)

Challenge:

- Change from a box plot to violin plot
    - Change the scale of the y axis to log

Answer:

- ggplot(surveys_complete, aes(x=species, y=weight)) +
    - geom_jitter (alpha=0.3, color="tomato") +
    - geom_violin(alpha=0) +
    - scale_y_log10()

as you start typing scale_ a dropdown menu will pop up and you will see a lot of options that you can

choose from
if you use just log in R, its the natural log by default, need to specify if you want to use log10()

Challenge:

- Make a boxplot for hindfoot_length. Overlay the boxplot layer on a jitter layer to show actual measurements.
- Add color to the data points on our boxplot according to the plot from which the sample was taken (plot_id)

where would you add the color aesthetic? it's good to add it to the global line (first line), add it inside the aes() parentheses

- ggplot(surveys_complete, aes(x=species, y=weight, color=plot_id)) +
    - geom_jitter (alpha=0.3) +
    - geom_boxplot(alpha=0)

Time Series Data;
yearly_counts<- surveys_complete %>%

- count(year, genus)

Let's graph it!

- ggplot(yearly_counts, aes(x=year, y=n))+
    - geom_line()

Ewwww.... not so visually pleasing

Why is that? Every genus has a point, but we'd like to show a separate line for each genus

- ggplot(yearly_counts, aes(x=year, y=n, group=genus)) +
    - geom_line()

Almost! But we don't know which line goes with which genus

group = is similar to group_by, which we covered earlier today

- ggplot(yearly_counts, aes(x=year, y=n, color=genus)) +
    - geom_line()

Let's seperate each genus into individual figures

ggplot(yearly_counts, aes(x=year, y=n)) +

- geom_line() +
- facet_wrap(facets=vars(genus)) #vars= variable you want to seperate into each graph by

New dataframe including counts of sexes:
yearly_sex_counts<- surveys_complete %>%

- count(year, genus, sex)

Graph different genera in their own graph and showcase the difference in sexes

ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex))+
geom_line()+
facet_wrap(facets = vars(genus))

Facet Grid does something similar but you can select the columns and the rows
ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex))+

- geom_line()+
- facet_grid(rows=vars(sex), cols=vars(genus))

Only want rows?
ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex))+

- geom_line()+
- facet_grid(rows=vars(genus))

Only want columns?
ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex))+

- geom_line()+
- facet_grid(cols=vars(genus))

The instructor's screen is actually grey, the projector is just not showing the grey background

ggplot has some standard colors that come pre-loaded, but there are a lot of color schemes

Change the background color of your graph using themes. E.g.:

ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex)) +

- geom_line() +
- facet_wrap(vars(genus)) +
- theme_bw() # changes the background to black and white

there are lots of different themes available
theme_classic()

Challenge: Create a plot that shows the average weight of each species over the years.
Answer:
yearly_weight<- surveys_complete %>%

- group_by(year, species_id) %>%
- summarize(avg_weight=mean(weight))

ggplot(yearly_weight, aes(x=year, y=avg_weight))+

- geom_line()+
- facet_wrap(vars(species_id))

For more customization, like adding titles, legends, changing fonts -> https://datacarpentry.org/R-

ecology-lesson/04-visualization-ggplot2.html#customization

** Saving plots **

1. Assign ggplot() result to an object.
2. Call ggsave() on plot object.

E.g.:

my_plot <- ggplot(yearly_sex_counts, aes(x = year, y = n, color = sex))+

- geom_line()+
- facet_grid(cols=vars(genus))

ggsave("figs/some_plot_file_name.png", my_plot)

Other references:

Geraldine:

09:23 This is the chat box: if you have a quick question, you can put it in here, instead of in the notes
Lisa Scott:09:37 What is the weblink for the Data Organization webpage she is showing?
Geraldine:09:38 If you go to the workshop page, you can click on the link that goes with the section
Geraldine:09:39 https://uf-carpentry.github.io/2019-11-04-Intro_to_R_etc/
Geraldine:09:40 This is the one for the spreadsheet lesson: https://datacarpentry.org/spreadsheet-ecology-lesson/ but you can find all the links for all the sections on the workshop page (go to the schedule, all sections are links)
Lisa Scott:09:40 thank you
Dahao:10:34 where is the lesson link please?
Vratika:10:35 http://swcarpentry.github.io/shell-novice/
Geraldine:10:47 http://swcarpentry.github.io/shell-novice/setup.html
Geraldine:10:47 This is the shell setup website