R SYNTAX

# Navigate to your "Documents" folder

cd ~/Documents

# Create a new folder called "quick-exercise"

mkdir quick-exercise

# Navigate into that folder

cd quick-exercise

# Using the text-edit of your choice (e.g., Atom), create a new (empty) file

# called "constitution.txt" inside the "quick-exercise" folder you made

# No command for this one!

# List the contents of the folder to verify the new file is there

ls

# Search online for the text of the US Constitution (while it's still around!)

# and paste it into the "constitution.txt" file you created. Save your changes in

# the editor.

# No command for this one either!

# Display the contents of the in the terminal

# Hint: use the `less` command so you can easily scroll up and down.

# For windows, use the `more` command.

less constitution.txt

For windows:

more constitution.txt

# Navigate to the "Desktop" folder for your machine

cd ../../Desktop #path may vary

# Display the contents of the "Documents/quick-exercise/constitution.txt" file in the Terminal

# \*without changing directories again!\*

cat ../Documents/quick-exercise/constitution.txt

# cat shows entire contents in one dump; less is also and acceptable command

# Bonus: Create 10 files with one single command in your terminal. Then delete them all!

touch myFile{1..10}.txt # Google to learn more about brace expansion and the "touch" command

rm myFile\*.txt # Google to learn about Wildcards

C3

In this exercise you'll practice using git to clone and edit repositories, as will be required for later exercises.

1. If you haven't already, configure your name and email address for your GitHub account using the terminal.

```bash

# enter your full name (without the dashes)

git config --global user.name "your-full-name"

# enter your email address (the one associated with your GitHub account)

git config --global user.email "your-email-address"

```

2. Fork this repository (`ch4-git-basics`) to your own GitHub account by clicking the `fork` button on the GitHub interface.

```bash

# Click the `fork` button to fork it to your account

```

3. Using the terminal, clone \*\*your forked repository\*\* to your machine (make sure you're in the desired directory on your terminal).

```bash

# Enter a desired directory

cd ~/Documents

# Get the URL by clicking the "Clone or Download" button on GitHub, then clicking the clipboard icon

# Clone the repository

git clone https://github.com/YOUR-USER-NAME/ch4-git-basics.git

```

4. On your machine, open up this file (`exercise-1/README.md`) in a text editor of your choice.

```bash

# Open up the file

```

5. In the `README.md` file, below these instructions, add an ordered list of what you ate for breakfast today.

```markdown

1. Coffee

2. Milk

3. Cheerios

4. Coffee

5. Banana

6. Coffee

7. Coffee

```

6. Using your terminal, add and commit the changes you've made to your repository

```bash

# Make sure that you're in the cloned repository

pwd

cd ch4-git-basics

# Add changes from all files in the repository

git add .

# Commit changes making sure to include a descriptive message

git commit -m "Adds breakfast"

```

7. Push changes up to GitHub, and view them in your web browser. Make sure you are looking at the repository under \_your\_ account!

```bash

# Push changes

git push origin master

```

C5

# Create a variable `puppies` equal to the number of puppies you'd like to have

puppies <- 8

# Create a variable `puppy\_price`, which is how much you think a puppy costs

puppy\_price <- 250

# Create a variable `total\_cost` that has the total cost of all of your puppies

total\_cost <- puppies \* puppy\_price

# Create a boolean variable `too\_expensive`, set to TRUE if the cost is greater

# than $1,000

too\_expensive <- total\_cost > 1000 # Bummer!

# Create a variable `max\_puppies`, which is the number of puppies you can

# afford for $1,000

max\_puppies <- 1000%/%puppy\_price # %/% is "divide and ignore remainder"

C6

# Exercise 1: calling built-in functions

# Create a variable `my\_name` that contains your name

my\_name <- "Joel Ross"

# Create a variable `name\_length` that holds how many letters (including spaces)

# are in your name (use the `nchar()` function)

name\_length <- nchar(my\_name)

# Print the number of letters in your name

print(name\_length)

# Create a variable `now\_doing` that is your name followed by "is programming!"

# (use the `paste()` function)

now\_doing <- paste(my\_name, "is programming!")

# Make the `now\_doing` variable upper case

toupper(now\_doing)

### Bonus

# Pick two of your favorite numbers (between 1 and 100) and assign them to

# variables `fav\_1` and `fav\_2`

fav\_1 <- 12

fav\_2 <- 87

# Divide each number by the square root of 201 and save the new value in the

# original variable

fav\_1 <- fav\_1 / sqrt(201)

fav\_2 <- fav\_2 / sqrt(201)

# Create a variable `raw\_sum` that is the sum of the two variables. Use the

# `sum()` function for practice.

raw\_sum <- sum(fav\_1, fav\_2)

# Create a variable `round\_sum` that is the `raw\_sum` rounded to 1 decimal place.

# Use the `round()` function.

round\_sum <- round(raw\_sum, 1)

# Create two new variables `round\_1` and `round\_2` that are your `fav\_1` and

# `fav\_2` variables rounded to 1 decimal places

round\_1 <- round(fav\_1, 1)

round\_2 <- round(fav\_2, 1)

# Create a variable `sum\_round` that is the sum of the rounded values

sum\_round <- sum(round\_1, round\_2)

# Which is bigger, `round\_sum` or `sum\_round`? (You can use the `max()` function!)

max(sum\_round, round\_sum)

**# Exercise 2: using built-in string functions**

# Create a variable `lyric` that contains the text "I like to eat apples and

# bananas"

lyric <- "I like to eat apples and bananas"

# Use the `substr()` function to extract the 1st through 13th letters from the

# `lyric`, and store the result in a variable called `intro`

# Use `?substr` to see more about this function

intro <- substr(lyric, 1, 13)

# Use the `substr()` function to extract the 15th through the last letter of the

# `lyric`, and store the result in a variable called `fruits`

# Hint: use `nchar()` to determine how many total letters there are!

fruits <- substr(lyric, 15, nchar(lyric))

# Use the `gsub()` function to substitute all the "a"s in `fruits` with "ee".

# Store the result in a variable called `fruits\_e`

# Hint: see http://www.endmemo.com/program/R/sub.php for a simpmle example (or

# use `?gsub`)

fruits\_e <- gsub("a", "ee", fruits)

# Use the `gsub()` function to substitute all the "a"s in `fruits` with "o".

# Store the result in a variable called `fruits\_o`

fruits\_o <- gsub("a", "o", fruits)

# Create a new variable `lyric\_e` that is the `intro` combined with the new

# `fruits\_e` ending. Print out this variable

lyric\_e <- paste(intro, fruits\_e)

print(lyric\_e)

# Without making a new variable, print out the `intro` combined with the new

# `fruits\_o` ending

print(paste(intro, fruits\_o))

**# Exercise 3: writing and executing functions**

# Define a function `add\_three` that takes a single argument and

# returns a value 3 greater than the input

add\_three <- function(value) {

value + 3 # return the result }

# Create a variable `ten` that is the result of passing 7 to your `add\_three`

# function

ten <- add\_three(7)

# Define a function `imperial\_to\_metric` that takes in two arguments: a number

# of feet and a number of inches

# The function should return the equivalent length in meters

imperial\_to\_metric <- function(feet, inches) {

total\_inches <- feet \* 12 + inches

meters <- total\_inches \* 0.0254

meters # return the value in meters }

# Create a variable `height\_in\_meters` by passing your height in imperial to the

# `imperial\_to\_metric` function

height\_in\_meters <- imperial\_to\_metric(5, 11)

**Exercise 4: functions and conditionals**

# Define a function `is\_twice\_as\_long` that takes in two character strings, and

# returns whether or not (e.g., a boolean) the length of one argument is greater

# than or equal to twice the length of the other.

# Hint: compare the length difference to the length of the smaller string

is\_twice\_as\_long <- function(str1, str2) {

diff <- abs(nchar(str1) - nchar(str2))

min\_length <- min(nchar(str1), nchar(str2))

diff >= min\_length # if difference is more than short

}

# Call your `is\_twice\_as\_long` function by passing it different length strings

# to confirm that it works. Make sure to check when \_either\_ argument is twice

# as long, as well as when neither are!

is\_twice\_as\_long("dog", "elephant")

is\_twice\_as\_long("elephant", "dog")

is\_twice\_as\_long("dog", "cat")

# Define a function `describe\_difference` that takes in two strings. The

# function should return one of the following sentences as appropriate

# "Your first string is longer by N characters"

# "Your second string is longer by N characters"

# "Your strings are the same length!"

describe\_difference <- function(first, second) {

diff <- nchar(first) - nchar(second)

if (diff > 0) {

sentence <- paste("Your first string is longer by", diff, "characters")

} else if (diff < 0) {

sentence <- paste("Your second string is longer by", -diff, "characters")

} else {

sentence <- "Your strings are the same length!"

}

sentence # return the sentence

}

# Call your `describe\_difference` function by passing it different length strings

# to confirm that it works. Make sure to check all 3 conditions1

describe\_difference("dog", "elephant")

describe\_difference("elephant", "dog")

describe\_difference("dog", "cat")

**C7**

**# Exercise 1: Creating and Operating on Vectors**

**# Create a vector `names` that contains your name and the names of 2 people next to you. Print the vector.**

names <- c("Joel", "Mike", "Dave")

print(names)

**# Use the colon operator : to create a vector `n` of numbers from 10:49** n <- 10:49

**# Use the `length()` function to get the number of elements in `n`** length(n)

**# Add 1 to each element in `n` and print the result** print(n + 1)

**# Create a vector `m` that contains the numbers 10 to 1 (in that order).** m <- seq(10, 1)

**# Subtract `m` FROM `n`. Note the recycling!**

n\_less\_m <- n - m

print(n\_less\_m)

**# Use the `seq()` function to produce a range of numbers from -5 to 10 in `0.1` increments. Store it in a variable `x\_range`**

x\_range <- seq(-5, 10, 0.1)

**# Create a vector `sin\_wave` by calling the `sin()` function on each element in `x\_range`.**

sin\_wave <- sin(x\_range)

**# Create a vector `cos\_wave` by calling the `cos()` function on each element in `x\_range`.**

cos\_wave <- cos(x\_range)

**# Create a vector `wave` by multiplying `sin\_wave` and `cos\_wave` together, then adding `sin\_wave` to the product**

wave <- sin\_wave \* cos\_wave + sin\_wave

**# Use the `plot()` function to plot your `wave`!** plot(wave)

**# Exercise 2: Indexing and Filtering Vectors**

**# Create a vector `first\_ten` that has the values 10 through 20 in it (using the : operator)**

first\_ten <- 10:20

**# Create a vector `next\_ten` that has the values 21 through 30 in it (using the seq() function)**

next\_ten <- seq(21, 30)

**# Create a vector `all\_numbers` by combining the previous two vectors**

all\_numbers <- c(first\_ten, next\_ten)

**# Create a variable `eleventh` that contains the 11th element in `all\_numbers`**

eleventh <- all\_numbers[11]

**# Create a vector `some\_numbers` that contains the 2nd through the 5th elements of `all\_numbers`**

some\_numbers <- all\_numbers[2:5]

**# Create a vector `even` that holds the even numbers from 1 to 100**

even <- seq(2, 100, 2) # start at first even number!

**# Using the `all()` function and `%%` (modulo) operator, confirm that all of the numbers in your `even` vector are even**

test <- all(even %% 2 == 0)

**# Create a vector `phone\_numbers` that contains the numbers 8, 6, 7, 5, 3, 0, 9**

phone\_numbers <- c(8, 6, 7, 5, 3, 0, 9)

**# Create a vector `prefix` that has the first three elements of `phone\_numbers`**

prefix <- phone\_numbers[1:3]

**# Create a vector `small` that has the values of `phone\_numbers` that are less than or equal to 5**

small <- phone\_numbers[phone\_numbers <= 5]

**# Create a vector `large` that has the values of `phone\_numbers` that are strictly greater than 5**

large <- phone\_numbers[phone\_numbers > 5]

**# Replace the values in `phone\_numbers` that are larger than 5 with the number 5**

phone\_numbers[phone\_numbers > 5] <- 5

**# Replace every odd-numbered value in `phone\_numbers` with the number 0**

phone\_numbers[phone\_numbers %% 2 == 1] <- 0

**# Exercise 3: Vector Practice**

**# Create a vector `words` of 6 (or more) words.**

words <- c("the", "quick", "brown", "fox", "jumped", "over", "lazy", "dog")

**# Create a vector `words\_of\_the\_day` that is your `words` vector with the string "is the word of the day!" pasted on to the end**

**# BONUS: Surround the word in quotes (e.g., `'data' is the word of the day!`)**

words\_of\_the\_day <- paste0("'", words, "' is the word of the day!")

words\_of\_the\_day

**# Create a vector `a\_f\_words` which are the elements in `words` that start with "a" through "f"**

# Hint: use a comparison operator to see if the word comes before "f" alphabetically!

# Tip: make sure all the words are lower-case, and only consider the first letter of the word!

a\_f\_words <- words[substring(words, 1, 1) <= "f"]

a\_f\_words

**# Create a vector `g\_m\_words` which are the elements in `words` that start with "g" through "m"**

g\_m\_words <- words[words >= "g" & substring(words, 1, 1) <= "m"]

g\_m\_words

#g\_words <- words[words >= "g"] # alternative approach

#g\_m\_words <- g.words[g\_words <= "m"]

**# Define a function `word\_bin` that takes in three arguments: a vector of words, and two letters. The function should return a vector of words that go between those letters alphabetically.**

word\_bin <- function(words, start, end) {

words[words >= start & substring(words, 1, 1) <= end] }

**# Use your `word\_bin` function to determine which of your words start with "e" through "q"**

word\_bin(words, "e", "q")

**C8**

**# Exercise 1: Creating and Accessing Lists**

**# Create a vector `my\_breakfast` of everything you ate for breakfast**

my\_breakfast <- c("toast", "eggs", "tea")

**# Create a vector `my\_lunch` of everything you ate (or will eat) for lunch**

my\_lunch <- c("soup", "pb + j")

**# Create a list `meals` that has contains your breakfast and lunch**

meals <- list(breakfast = my\_breakfast, lunch = my\_lunch)

**# Add a "dinner" element to your `meals` list that has what you plan to eat for dinner**

meals$dinner <- c("curry", "rice")

**# Use dollar notation to extract your `dinner` element from your list and save it in a vector called 'dinner'**

dinner <- meals$dinner

**# Use double-bracket notation to extract your `lunch` element from your list and save it in your list as the element at index 5 (no reason beyond practice)**

meals[[5]] <- meals[["lunch"]]

**# Use single-bracket notation to extract your breakfast and lunch from your list and save them to a list called `early\_meals`**

early\_meals <- meals[1:2]

**# Create a list that has the number of items you ate for each meal**

# Hint: use the `lappy()` function to apply the `length()` function to each item

items <- lapply(meals, length)

**# Write a function `add\_pizza` that adds pizza to a given meal vector, and returns the pizza-fied vector**

add\_pizza <- function(meal) {

meal <- c(meal, "pizza")

meal # return the new vector }

**# Create a vector `better\_meals` that is all your meals, but with pizza!**

better\_meals <- lapply(meals, add\_pizza)

**# Exercise 2:** **Using `\*apply()` Functions**

**# Create a \*list\* of 10 random numbers. Use the `runif()` function to make a vector of random numbers, then use `as.list()` to convert that to a list**

nums <- as.list(runif(10, 1, 100))

**# Use `lapply()` to apply the `round()` function to each number, rounding it to the nearest 0.1 (one decimal place)**

lapply(nums, round, 1)

**# Create a variable 'sentence' that contains a sentence of text (something longish). Make the sentence lowercase; you can use a function to help.**

sentence <- tolower("I do not like green eggs and ham. I do not like them, Sam-I-Am")

**# Use the `strsplit()` function to split the sentence into a vector of letters.**

# Hint: split on `""` to split every character

# Note: this will return a \_list\_ with 1 element (which is the vector of letters)

letters\_list <- strsplit(sentence, "")

**# Extract the vector of letters from the resulting list**

letters <- letters\_list[[1]]

**# Use the `unique()` function to get a vector of unique letters**

letters\_unique <- unique(letters)

**# Define a function `count\_occurrences` that takes in two parameters: a letter and a vector of letters. The function should return how many times that letter occurs in the provided vector.**

# Hint: use a filter operation!

count\_occurrences <- function(letter, all\_letters) {

length(all\_letters[all\_letters == letter]) }

**# Call your `count\_occurrences()` function to see how many times the letter 'e' is in your sentence.**

count\_occurrences("e", letters)

**# Use `sapply()` to apply your `count\_occurrences()` function to each unique letter in the vector to determine their frequencies. Convert the result into a list (using `as.list()`).**

frequencies <- as.list(sapply(letters\_unique, count\_occurrences, letters))

**C10**

**#Exercise 1**: **Creating Data Frames**

**# Create a vector of the number of points the Seahawks scored in the first 4 games of the season (google "Seahawks" for the scores!)**

points <- c(12, 3, 37, 27) # example from 2016 season

**# Create a vector of the number of points the Seahwaks have allowed to be scored against them in each of the first 4 games of the season** points\_allowed <- c(10, 9, 18, 17)

**# Combine your two vectors into a dataframe called `games`**

games <- data.frame(points, points\_allowed)

**# Create a new column "diff" that is the difference in points between the teams**

# Hint: recall the syntax for assigning new elements (which in this case will be a vector) to a list!

games$diff <- games$points - games$points\_allowed

**# Create a new column "won" which is TRUE if the Seahawks won the game**

games$won <- games$diff > 0

**# Create a vector of the opponent names corresponding to the games played**

opponents <- c("Dolphins", "Rams", "49ers", "Jets")

**# Assign your dataframe rownames of their opponents**

rownames(games) <- opponents

**# View your data frame to see how it has changed**! View(games)

**#Exercise 2:** **Working With Data Frames**

**# Create a vector of 100 employees ("Employee 1", "Employee 2", ... "Employee 100")**

# Hint: use the `paste()` function and vector recycling to add a number to the word "Employee"

employees <- paste("Employee", 1:100)

**# Create a vector of 100 random salaries for the year 2017**

# Use the `runif()` function to pick random numbers between 40000 and 50000

salaries\_2017 <- runif(100, 40000, 50000)

**# Create a vector of 100 annual salary adjustments between -5000 and 10000.**

# (A negative number represents a salary decrease due to corporate greed)

salary\_adjustments <- runif(100, -5000, 10000)

**# Create a data frame `salaries` by combining the 3 vectors you just made**

# Remember to set `stringsAsFactors=FALSE`!

salaries <- data.frame(employees, salaries\_2017, salary\_adjustments, stringsAsFactors = FALSE)

**# Add a column to the `salaries` data frame that represents each person's salary in 2018 (e.g., with the salary adjustment added in).**

salaries$salaries\_2018 <- salaries$salaries\_2017 + salaries$salary\_adjustments

**# Add a column to the `salaries` data frame that has a value of `TRUE` if the person got a raise (their salary went up)**

salaries$got\_raise <- salaries$salaries\_2018 > salaries$salaries\_2017

### Retrieve values from your data frame to answer the following questions

### Note that you should get the value as specific as possible (e.g., a single cell rather than the whole row!)

**# What was the 2018 salary of Employee 57**

salary\_57 <- salaries[salaries$employees == "Employee 57", "salaries\_2018"]

**# How many employees got a raise?**

nrow(salaries[salaries$got\_raise == TRUE, ])

**# What was the dollar value of the highest raise?**

highest\_raise <- max(salaries$salary\_adjustments)

**# What was the "name" of the employee who received the highest raise?**

got\_biggest\_raise <- salaries[salaries$salary\_adjustments == highest\_raise, "employees"]

**# What was the largest decrease in salaries between the two years?**

biggest\_paycut <- min(salaries$salary\_adjustments)

**# What was the name of the employee who recieved largest decrease in salary?**

got\_biggest\_paycut <- salaries[salaries$salary\_adjustments == biggest\_paycut, "employees"]

**# What was the average salary change?**

avg\_increase <- mean(salaries$salary\_adjustments)

**# For people who did not get a raise, how much money did they lose on average?**

avg\_loss <- mean(salaries$salary\_adjustments[salaries$got\_raise == FALSE])

**## Consider:** do the above averages match what you expected them to be based on how you generated the salaries?

**# Write a .csv file of your salary data to your working directory** write.csv(salaries, "salaries.csv")

**# Exercise 3**: **Working With Built-in Data Sets**

**# Load R's "USPersonalExpenditure" dataset using the `data()` function**

# This will produce a data frame called `USPersonalExpenditure`

data("USPersonalExpenditure")

**# The variable `USPersonalExpenditure` is now accessible to you. Unfortunately, it's not a data frame (it's actually what is called a matrix) Test this using the `is.data.frame()` function**

is.data.frame(USPersonalExpenditure)

**# Luckily, you can pass the USPersonalExpenditure variable as an argument to the `data.frame()` function to convert it a data farm. Do this, storing the result in a new variable**

us\_exp <- data.frame(USPersonalExpenditure)

**# What are the column names of your dataframe?**

colnames(us\_exp)

**## Consider:** Why are they so strange? Think about whether you could use a number like 1940 with dollar notation!

**# What are the row names of your dataframe?**

rownames(us\_exp)

**# Add a column "category" to your data frame that contains the rownames**

us\_exp$category <- rownames(us\_exp)

**# How much money was spent on personal care in 1940?**

care\_1940 <- us\_exp["Personal Care", "X1940"]

**# How much money was spent on Food and Tobacco in 1960?**

food\_1960 <- us\_exp["Food and Tobacco", "X1960"]

**# What was the highest expenditure category in 1960?**

highest\_1960 <- us\_exp$category[us\_exp$X1960 == max(us\_exp$X1960)]

**# Define a function `lowest\_category` that takes in a year as a parameter, and returns the lowest spending category of that year**

lowest\_category <- function(year) {

col <- paste0("X", year)

us\_exp$category[us\_exp[, col] == min(us\_exp[, col])] }

**# Using your function, determine the lowest spending category of each year**

# Hint: use the `sapply()` function to apply your function to a vector of years

lowest <- sapply(seq(1940, 1960, 5), lowest\_category)

**# Exercise 4**: **External Data Sets: Gates Foundation Educational Grants**

**# Use the `read.csv()` functoin to read the data from the `data/gates\_money.csv`**

# file into a variable called `grants` using the `read.csv()`

# Be sure to set your working directory in RStudio, and do NOT treat strings as factors!

grants <- read.csv("data/gates\_money.csv", stringsAsFactors = FALSE)

**# Use the View function to look at the loaded data**  View(grants)

**# Create a variable `organization` that contains the `organization` column of the dataset**

organization <- grants$organization

**# Confirm that the "organization" column is a vector using the `is.vector()` function.**

# This is a useful debugging tip if you hit errors later!

is.vector(organization)

## Now you can ask some interesting questions about the dataset

**# What was the mean grant value?**

mean\_spending <- mean(grants$total\_amount)

**# What was the dollar amount of the largest grant?**

highest\_amount <- max(grants$total\_amount)

**# What was the dollar amount of the smallest grant?**

lowest\_amount <- min(grants$total\_amount)

**# Which organization received the largest grant?**

largest\_recipient <- organization[grants$total\_amount == highest\_amount]

**# Which organization received the smallest grant?**

smallest\_recipient <- organization[grants$total\_amount == lowest\_amount]

**# How many grants were awarded in 2010?**

length(grants$total\_amount[grants$start\_year == 2010])

**# Exercise 5: Large Datasets: Baby Name Popularity Over Time**

# Read in the female baby names data file found in the `data` folder into a variable called `names`. Remember to NOT treat the strings as factors!

names <- read.csv("data/female\_names.csv", stringsAsFactors = FALSE)

**# Create a data frame `names\_2013` that contains only the rows for the year 2013**

names\_2013 <- names[names$year == 2013, ]

**# What was the most popular female name in 2013?**

most\_popular\_name\_2013 <- names\_2013[names\_2013$prop == max(names\_2013$prop), "name"]

**# Write a function `most\_popular\_in\_year` that takes in a year as a value and returns the most popular name in that year**

most\_popular\_in\_year <- function(year) {

names\_year <- names[names$year == year, ]

most\_popular <- names\_year[names\_year$prop == max(names\_year$prop), "name"]

most\_popular # return most popular }

**# What was the most popular female name in 1994?**

most\_popular\_1994 <- most\_popular\_in\_year(1994)

**# Write a function `number\_in\_million` that takes in a name and a year, and returns statistically how many babies out of 1 million born that year have that name.**

# Hint: get the popularity percentage, and take that percentage out of 1 million.

number\_in\_million <- function(name, year) {

name\_popularity <- names[names$year == year & names$name == name, "prop"]

round(name\_popularity \* 1000000, 1) }

**# How many babies out of 1 million had the name 'Laura' in 1995?**

number\_in\_million("Laura", 1995)

**# How many babies out of 1 million had your name in the year you were born?**