R for bioinformatics (or anything else)

Cait Harrigan & Gabi Morgenshtern

Before we begin….

pollev.com/charrigan888

Please vote on what you want to get out of today's workshop

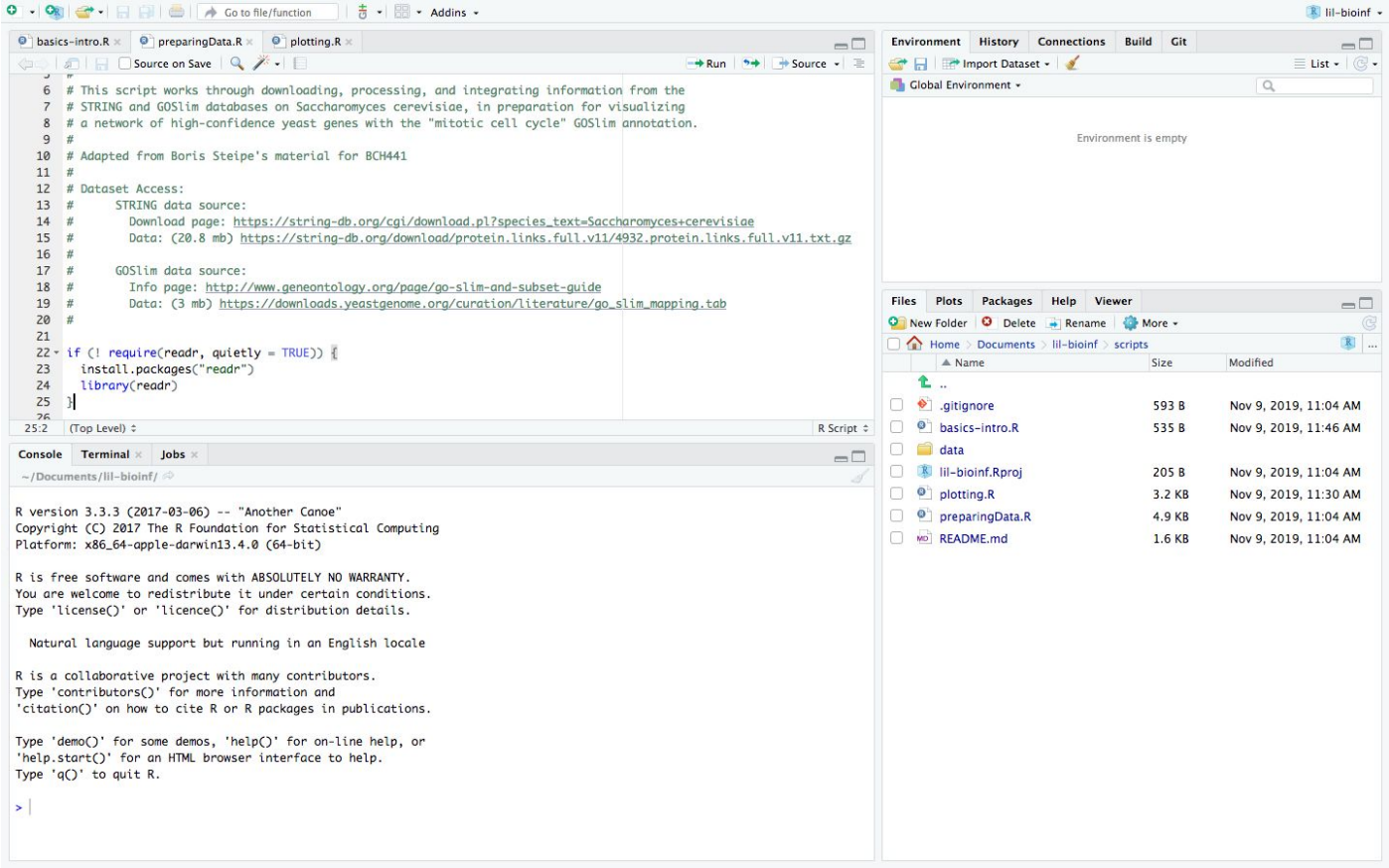
In this tutorial we will use two pieces of software, R and RStudio. Both are freely available online for download

What is R? What is Rstudio?

R is a programming language - a set of rules for how to tell a computer to do something

RStudio is an interface for coding with R. You may see it called an “environment” or an “IDE”

RStudio environment



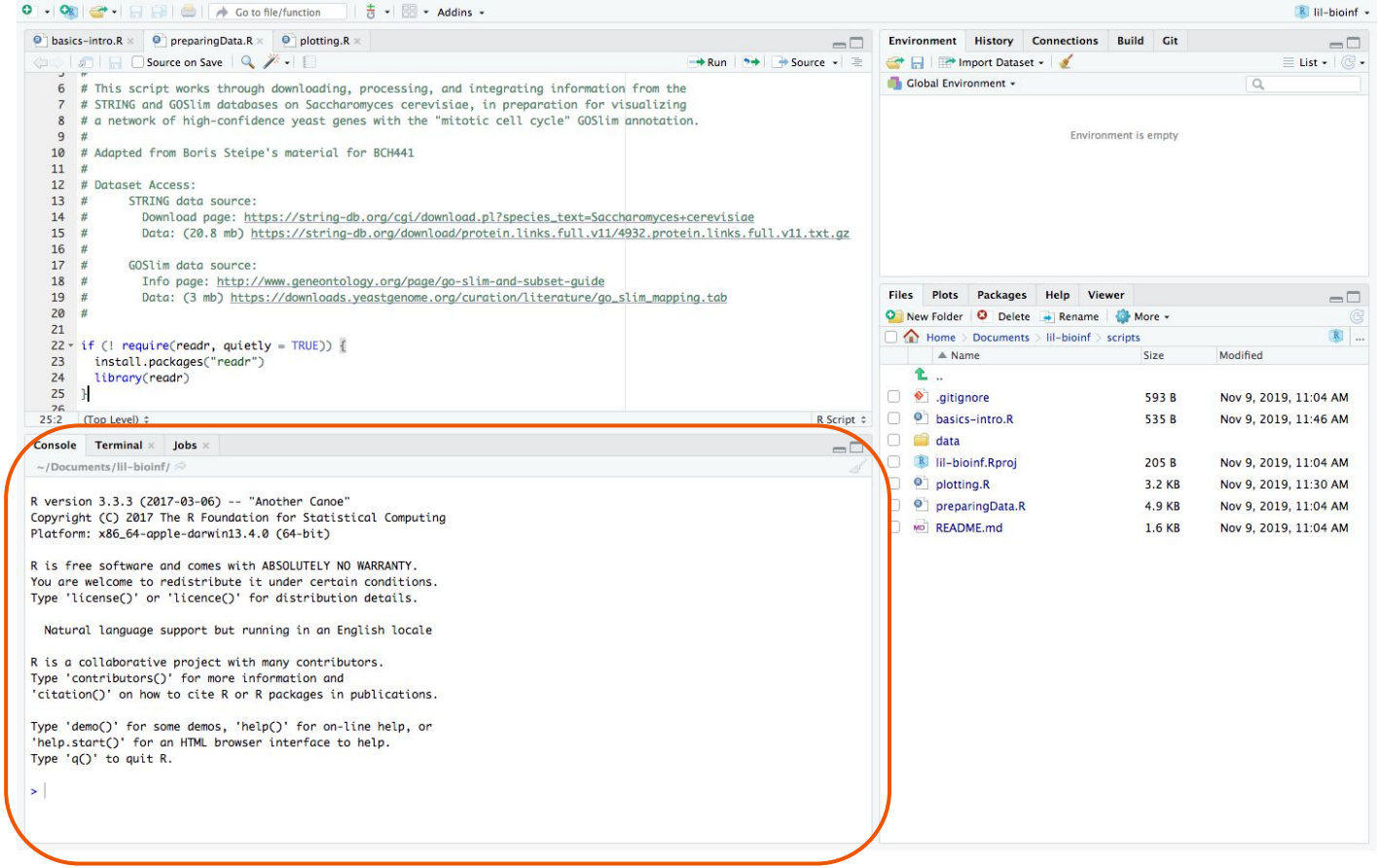
The console:
- run R code

Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts



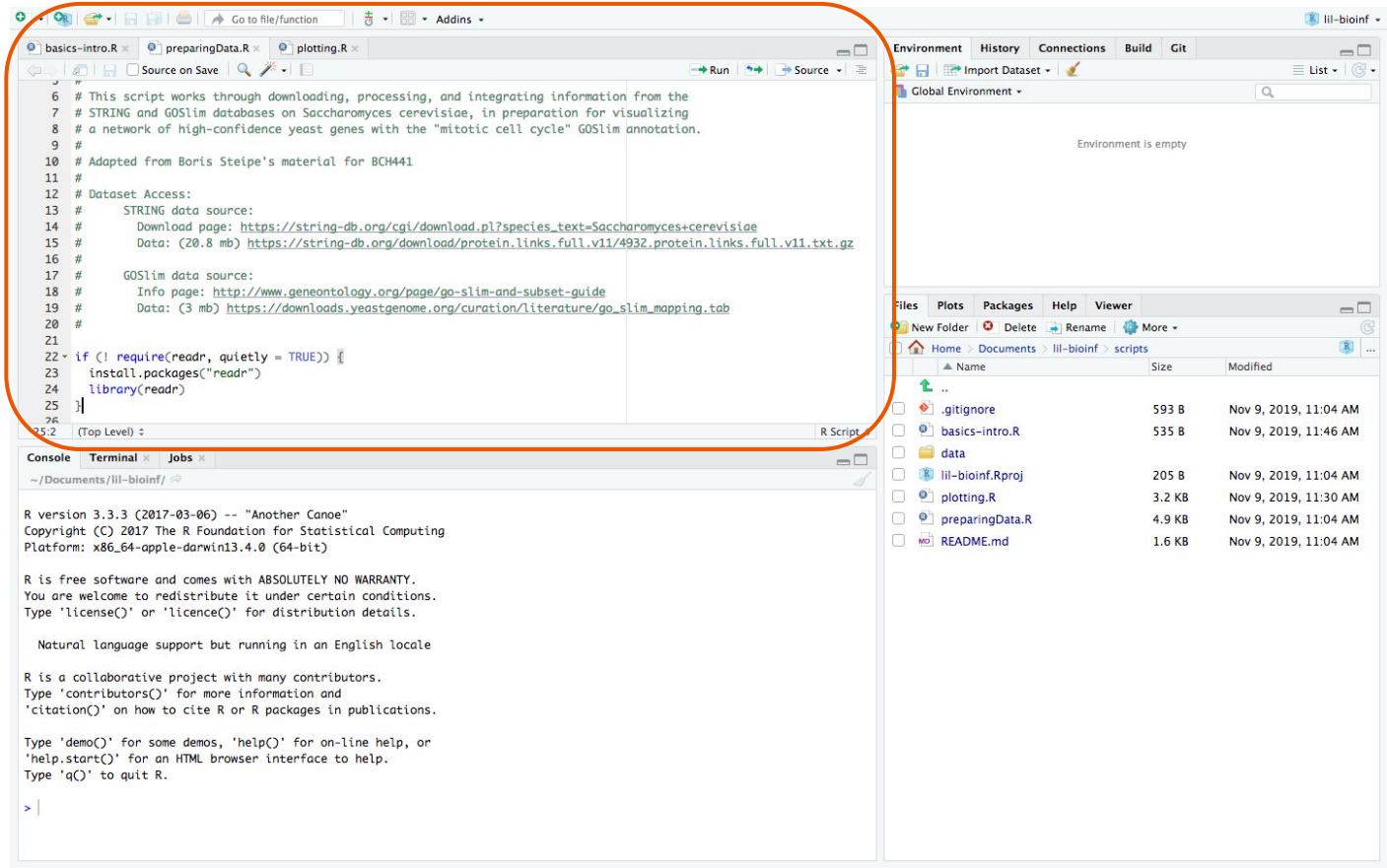
The console:
- run R code

Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts



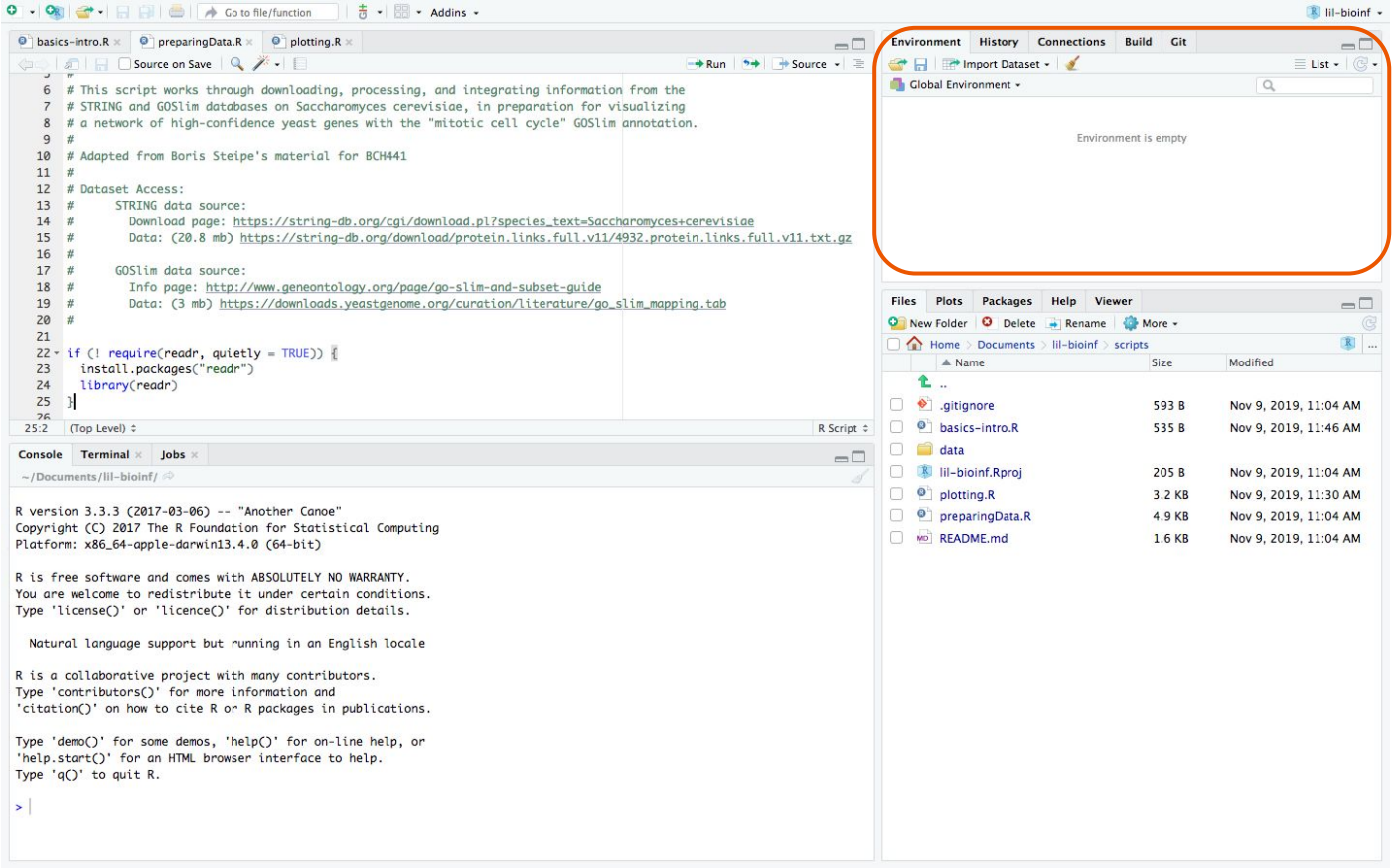
The console:
- run R code

Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts



The console:
- run R code

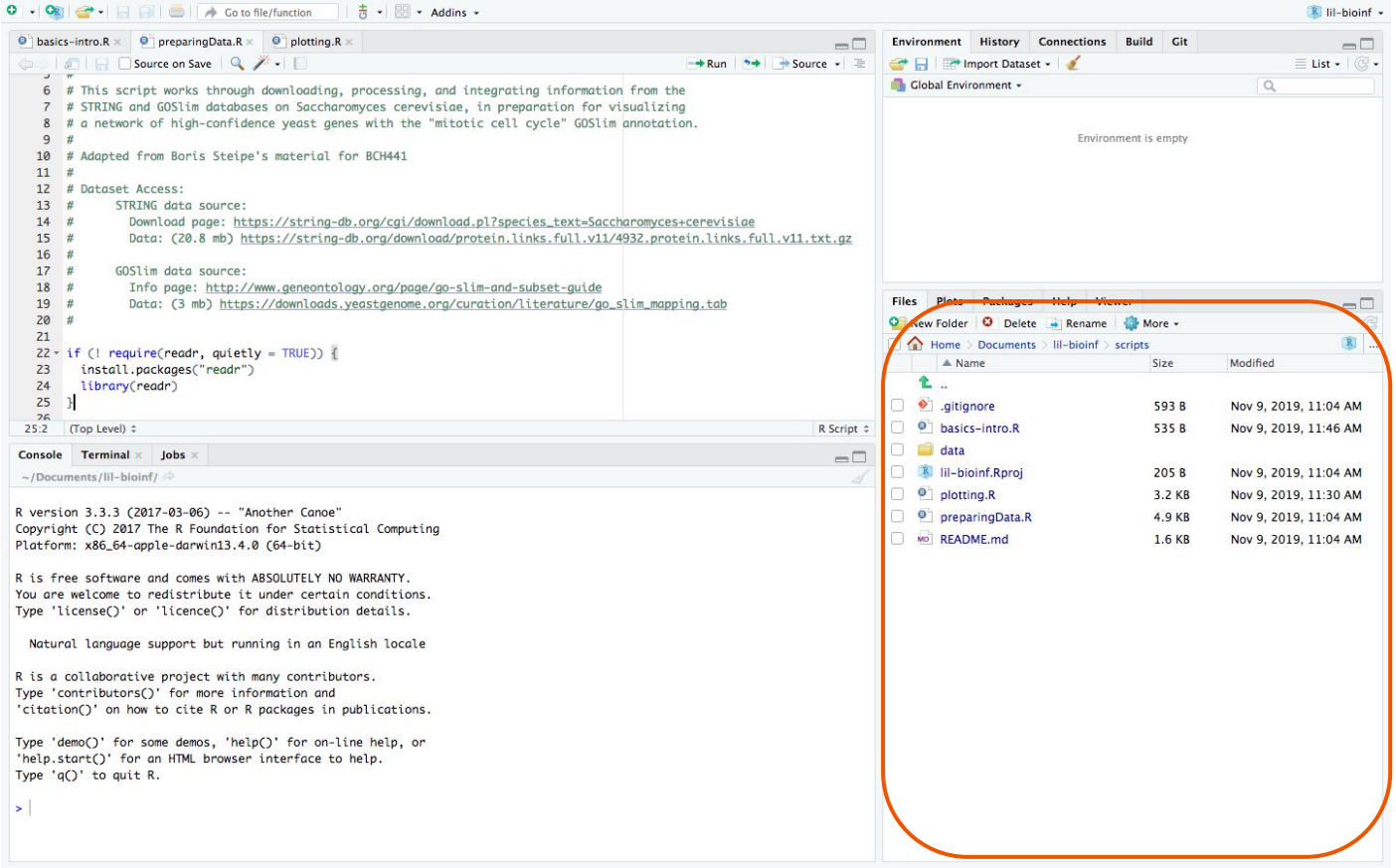
Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts

RStudio environment



The console:
- run R code

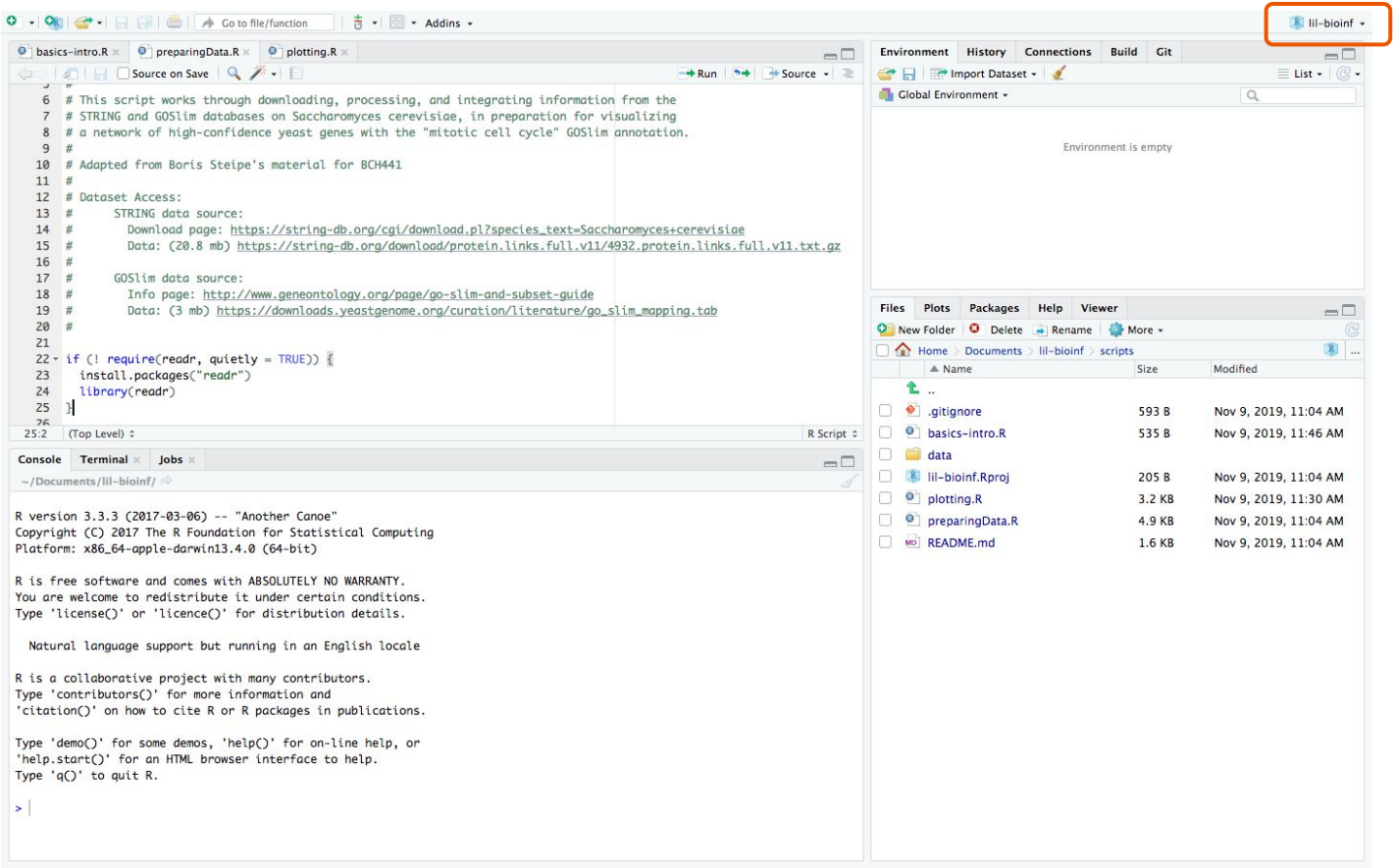
Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts

RStudio environment



The console:
- run R code

Scripts:
- write and save R code
- can be sourced to re-use useful code

Environment:
- what information is available to the console

Files:
- files on your machine

Project:
- a certain set-up for a group of scripts

Objects

Objects let us talk about the things we want our code to do. Objects are part of a useful paradigm that lets us turn abstract concepts into concrete actions.

Some examples of objects: a number, a string of characters, a list, or even a function.

Abstract concept → Concrete action

Objects

Objects let us talk about the things we want our code to do. Objects are part of a useful paradigm that lets us turn abstract concepts into concrete actions.

Some examples of objects: a number, a string of characters, a list, or even a function.

Abstract concept → Concrete action

What's the average speed of these three snails?

$$\frac{0.013\text{m/s} + 0.008\text{m/s} + 0.011\text{m/s}}{3}$$

Objects

Objects let us talk about the things we want our code to do. Objects are part of a useful paradigm that lets us turn abstract concepts into concrete actions.

Some examples of objects: a number, a string of characters, a list, or even a function.

Abstract concept → Concrete action

What's the average speed of these three snails?

```
Console   Terminal ×   Jobs ×
~/Documents/lil-bioinf/
>
> (0.013 + 0.008 + 0.011) / 3
[1] 0.01066667
>
```

Variables

We can store objects in variables. They act as a shorthand name for the object that they refer to.

We use the assignment operator `<-` to set the value of a variable

```
Console   Terminal ×   Jobs ×
~/Documents/lil-bioinf/
> num_snails <- 3
> num_snails
[1] 3
```

We can manipulate variables in the same way that we did the objects they refer to.

```
Console   Terminal ×   Jobs ×
~/Documents/lil-bioinf/
> num_snails <- 3
> speed_snail1 <- 0.013
> speed_snail2 <- 0.008
> speed_snail3 <- 0.011
> (speed_snail1 + speed_snail2 + speed_snail3) / num_snails
[1] 0.01066667
```

Functions

We use functions to write reusable code. They let us make modifications to objects, or get information about them.

A function takes input parameters, and returns output.

Console Terminal Jobs

~/Documents/lil-bioinf/

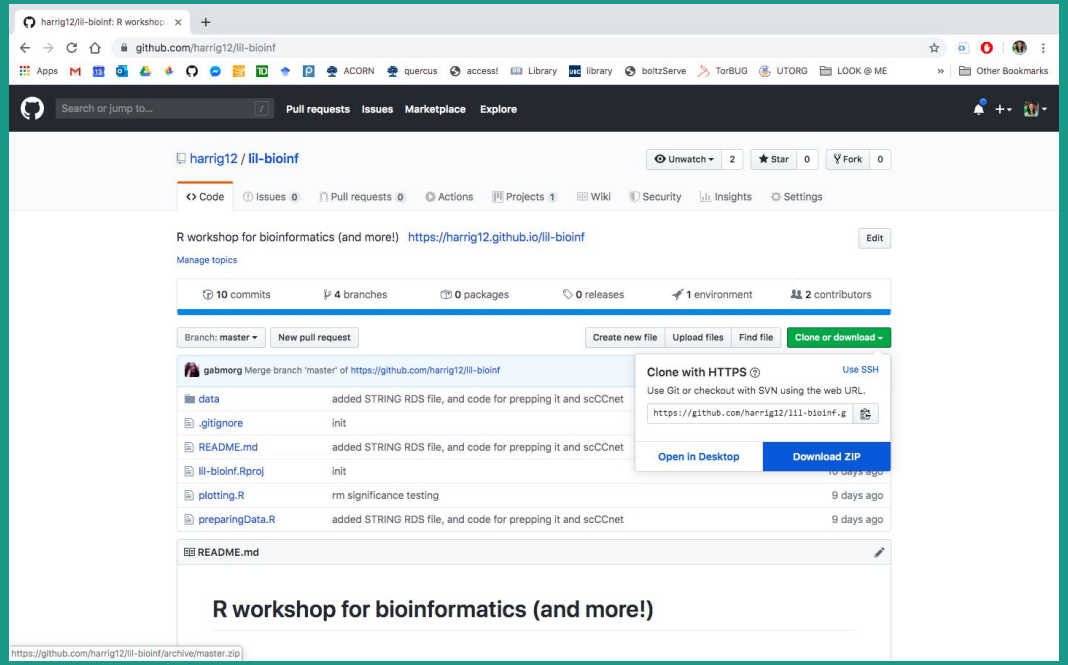
```
> fastest_snail <- function(speeds){
+
+   snail_num <- which(speeds == max(speeds))
+   message <- sprintf("The fastest is snail number %s", snail_num)
+
+   return(message)
+ }
>
> fastest_snail(snail_speeds)
[1] "The fastest is snail number 1"
>
```

Function definition

Function call - the input is the object snail_speeds and the return value is a message about the snails

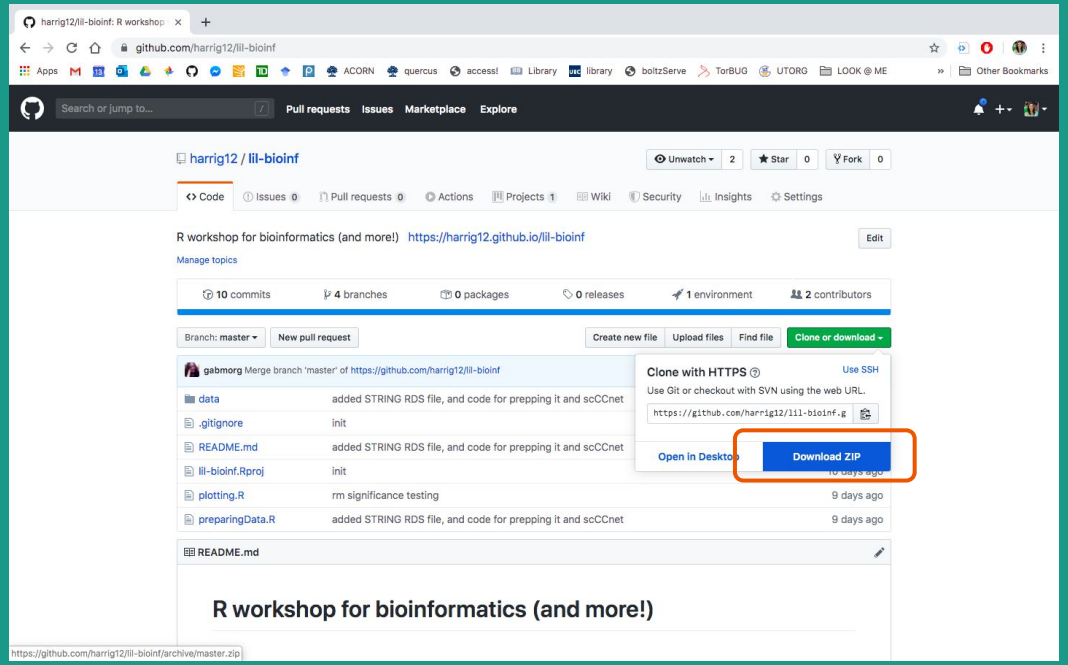
Let’s dive in!

github.com/harrig12/lil-bioinf



Let’s dive in!

github.com/harrig12/lil-bioinf



Scripts

Save code for reusability.

For example - we saw the basics-intro.R script

Scripts can be executed with “run” and this will *sequentially* step through all the code in the script.

This can be done without explicitly opening an R session (ex. from the command line). Running the script will start a fresh one, different from your console session.

Get more functionality

CRAN

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

CRAN and Bioconductor are both great places to get *maintained* R packages. After you install a new package, load it at the start of your scripts or in the console with the library() function

Data structures

To hold and manipulate data, we tend to use particular objects. Dataframes are great for their flexibility and versatility. You may also see data in lists, matrices, etc.

Certain packages implement their own data structures with “slots” for different pieces of information.

For now, we’ll focus on dataframes.

Dataframes can be constructed with named or un-named columns.

```
Console   Terminal ×   Jobs ×
~/Documents/lil-bioinf/
> snail_data <- data.frame(speed = snail_speeds, shell_colour = snail_shells_factor)
> print(snail_data)
  speed shell_colour
1 0.013        brown
2 0.008       orange
3 0.011        brown
```

If you try to add data that is mismatched in length to other columns in the dataframe, you might get an error.

```
Console   Terminal ×   Jobs ×
~/Documents/lil-bioinf/
> data.frame(1:5, c("one", "two"))
Error in data.frame(1:5, c("one", "two")) :
  arguments imply differing number of rows: 5, 2
```

Useful functions with dataframes

Try each of these functions out on snail_data. What does each output?

Function	Output
head()	
dim()	
colnames()	
summary()	
table()	
plot()	

Useful functions with dataframes

Try each of these functions out on snail_data. What does each output?

Function	Output
head()	A preview of the dataframe
dim()	Dimensions of the dataframe
colnames()	Column names of the dataframe
summary()	Summary of each column
table()	Counts of matching entries
plot()	A plot

Accessing and subsetting dataframes

Subsetting is an important part of data manipulation!

For columns, we call this “selecting”. For rows, we call this “filtering” or “subsetting”.

In base R, there are a few ways to do this. In the wild, you may see the following forms:

snail_data$speed

snail_data[[“speed”]]

snail_data[,1]

You may find it easier to use subset()

Accessing and subsetting dataframes

Subsetting is an important part of data manipulation!

For columns, we call this “selecting”. For rows, we call this “filtering” or “subsetting”.

A great way filter rows is to use boolean vectors, that record whether a desired property is true or not

Console Terminal Jobs

~/Documents/lil-bioinf/

```
> print( snail_data[snail_data$shell_colour == "brown" ,] )
  speed shell_colour
1 0.013        brown
3 0.011        brown
```

Exit survey

pollev.com/charrigan888

Thanks!

These slides and workshop materials are available at harrig12.github.io/lil-bioinf