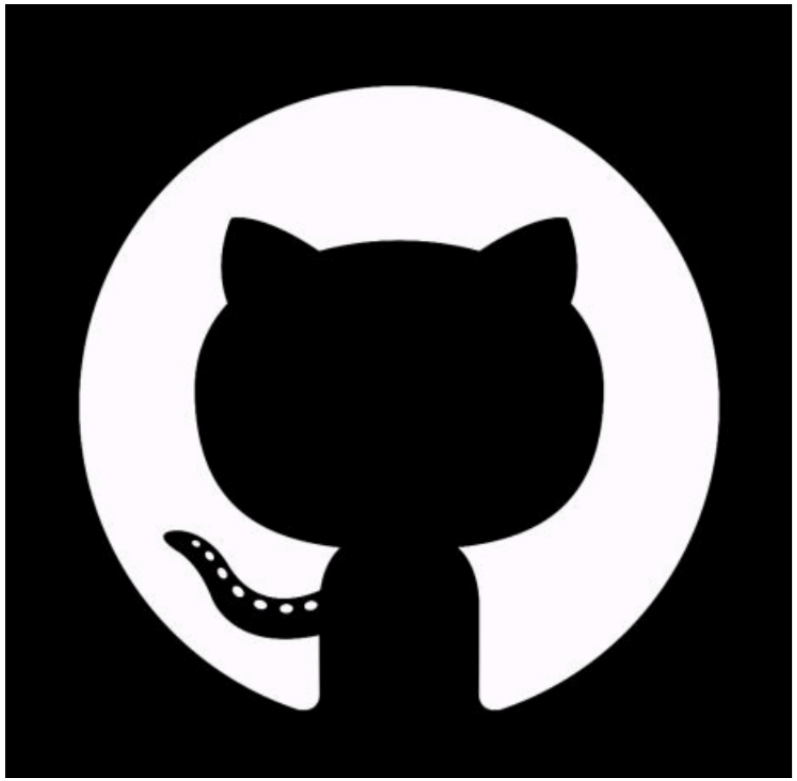
STAT 215A Fall 2019
Week 1

Tiffany Tang

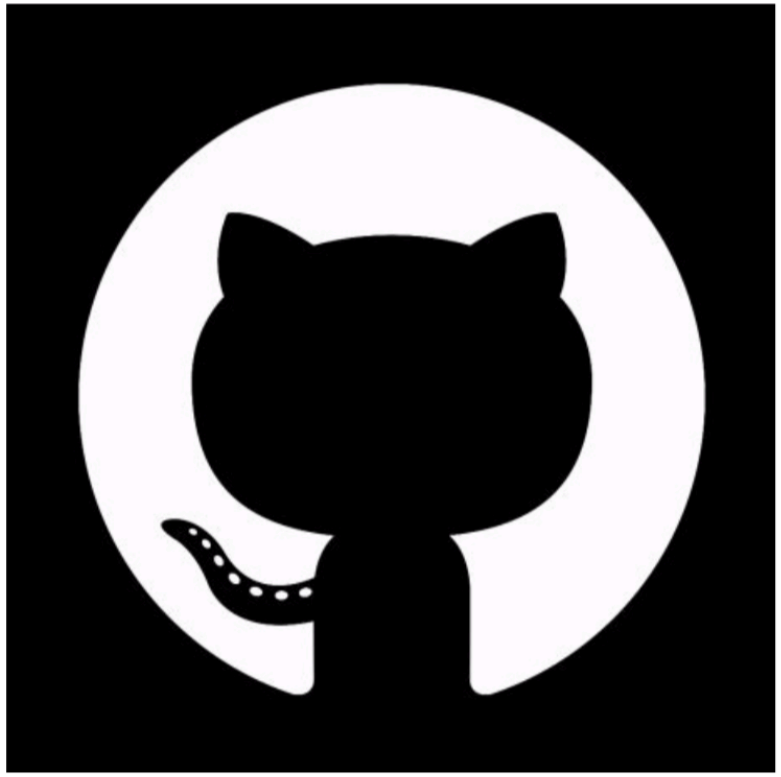
9/5/19

Git & Github



R & the “tidyverse”

Git & Github



- What are Git and GitHub?
- Set up GitHub repository for STAT215A
  - Add me as a collaborator
  - Push something so I can see it

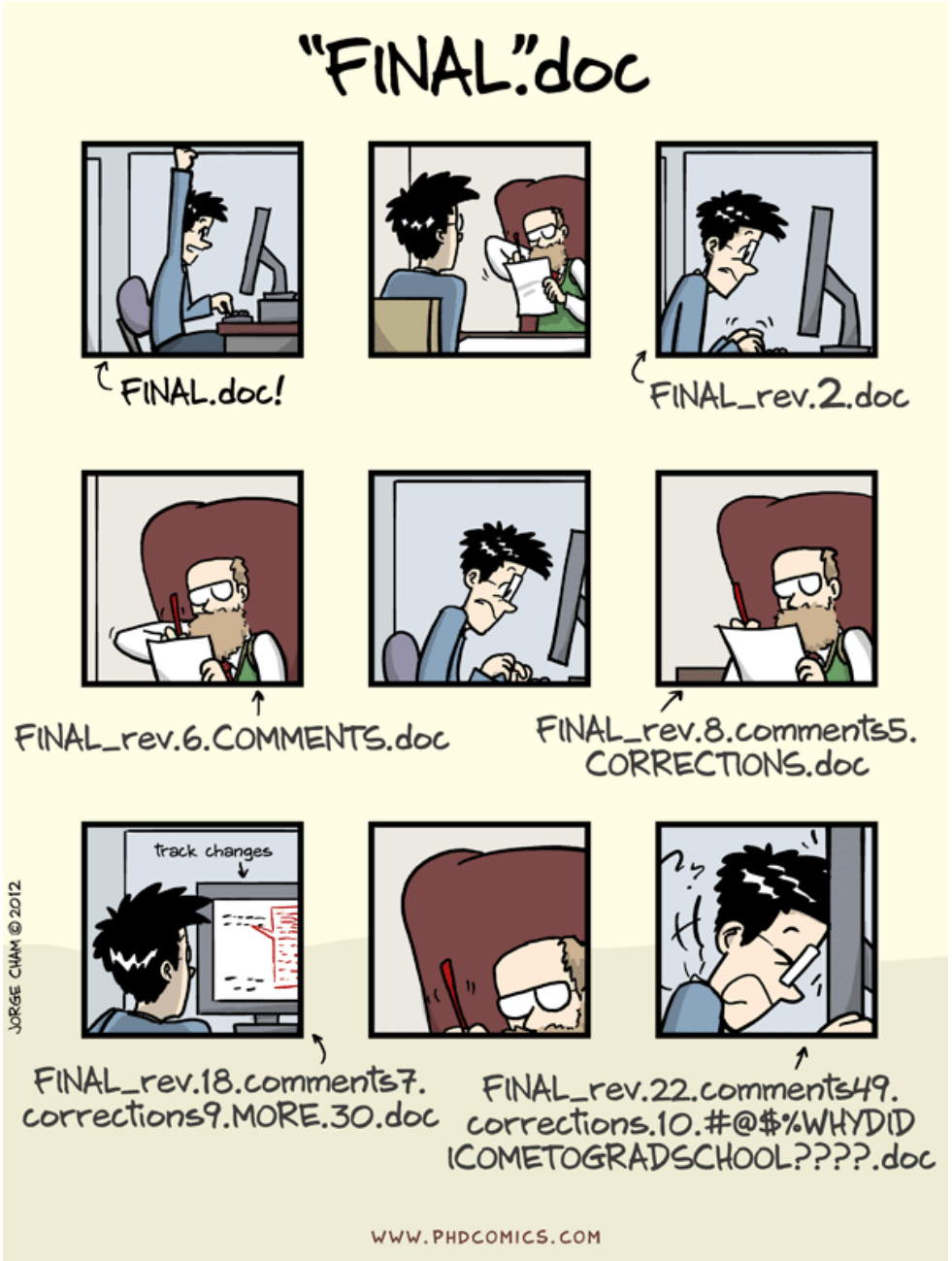
Setting up GitHub

https://github.com/tiffanymtang/STAT-215A-Fall-2019

1. Install Git on your system (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
2. Sign up for GitHub (https://github.com/)
3. Go to https://education.github.com/ and sign up for the student pack to get unlimited private repositories. You are a "student" and you want an "individual account".

What is Git?

- A version control system
- Stores data as a series of snapshots
- If files have not changed, it will simply access the file from a previous commit instead of saving it again
- Allows access to all the committed steps along the way



Git vs. GitHub

Local Git Repository

- You have a local version of the folder on your computer
- History stored in .git file
- Only you can see the changes made in the local version

Remote GitHub Repository

- On the GitHub website lives a remote version of the folder
- Everyone can see these changes (if repository is public)

Why is this important?

- Imagine working on a project with several collaborators…
- Using Git/GitHub allows everyone to have their own local version of the project while still maintaining a “master” version of the project, hosted remotely on GitHub
- You can make changes freely without people seeing what you are doing
  - You can thoroughly test your changes before adding to the master copy
- Version control!!
  - Especially great if your changes create bugs because you can backtrack/revert

Typical Pipeline

Local Repository

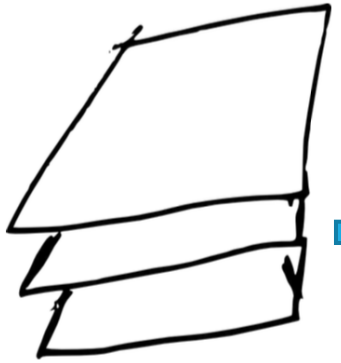
Remote Repository

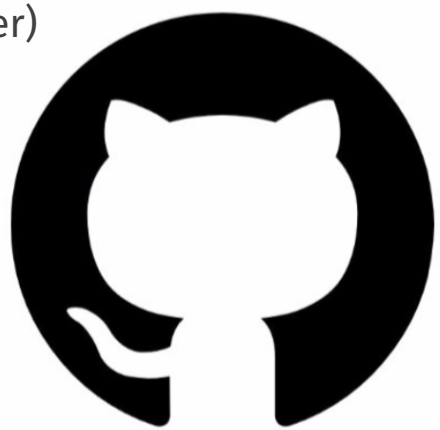
(1) git pull

(to retrieve the most recent version from the server)

(2) make local changes

(e.g. create file called filename.txt)





(3) git add filename.txt

(4) git commit -m “add description”

(add and commit when you have made some changes and want to be able to save your current checkpoint as a snapshot)

(5) git push

(make changes available to everyone with access to the repo)

Warning: May need to “git pull” before “git push” to mitigate merge conflicts

Other Useful Git Commands

- To check the status of working directory and staging area: **git status**
- To show a list of commits in one-line format: **git log –oneline**
- If you run into a merge conflict when trying to push your local changes, error messages will appear in your terminal. One way to resolve these conflicts is to follow the instructions that are typically given in the error messages, but be careful…

Let’s set up GitHub repositories for class

Setting up GitHub

1. Install Git on your system (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
2. Sign up for GitHub (https://github.com/)
3. Go to https://education.github.com/ and sign up for the student pack to get unlimited private repositories. You are a "student" and you want an "individual account".

Setting up GitHub

4. Locally on your machine, open terminal and change directories to where you want to keep class materials

5. Clone my stat-215-a repository: **git clone https://github.com/tiffanymtang/stat-215-a** This will create a copy of my repository on your own computer

```
Tiffany-MacBook-Pro:~ Tiffany$ git clone https://github.com/tiffanymtang/stat-215-a
```

6. On the GitHub website, log in and create a **private** remote repository called **stat-215-a**. Add me (**tiffanymtang**) as a collaborator for this repository (check out settings on the repo website).

Setting up GitHub

7. Back in terminal, change directories to the stat-215-a folder: **cd ./stat-215-a**

8. Set the origin of your local repo to be the remote repo that you just made: **git remote set-url origin https://github.com/USERNAME/stat-215-a.git** (Change USERNAME below to your username). This tells git which remote repository to push your changes to when you git push.

```
Tiffany-MacBook-Pro:~ Tiffany$ cd ./stat-215-a
Tiffany-MacBook-Pro:stat-215-a Tiffany$ git remote set-url origin https://github.com
/tiffanymtang/stat-215-a.git
```

Setting up GitHub

9. Edit *info.txt* to reflect your own information.

10. Now, we need to push the changes you made to the remote repo:
    - In command line, type **git status** to see your changes. If you see a bunch of text including **modified: info.txt**, you are ready to move on
    - In command line, add (**git add info.txt**) and commit (**git commit –m "Updated info.txt with my own information"**) your edited *info.txt* file
    - Finally, push your changes to the remote repository via **git push**

```
Tiffany-MacBook-Pro:stat-215-a Tiffany$ git add info.txt
Tiffany-MacBook-Pro:stat-215-a Tiffany$ git commit -m "Updated info.txt with my own
information"
[master 8ca02bc] Updated info.txt with my own information
 1 file changed, 1 insertion(+), 1 deletion(-)
Tiffany-MacBook-Pro:stat-215-a Tiffany$ git push
```

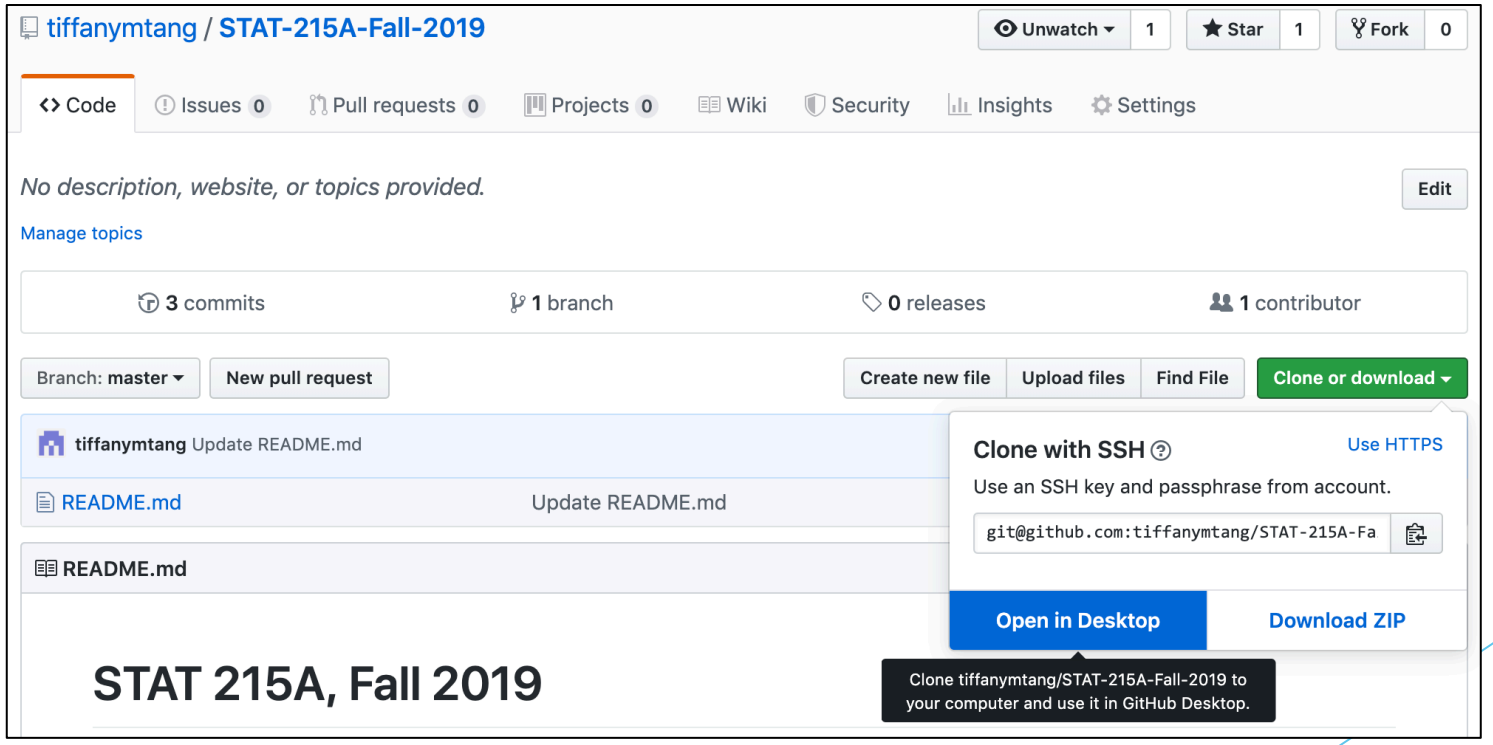
11. Check that *info.txt* has been updated in your remote GitHub repository by navigating to https://github.com/USERNAME/stat-215-a (change USERNAME to your username)

GitHub Desktop

- So far, we have done everything in the command line
- Another option is to use GitHub Desktop, which can be downloaded from https://desktop.github.com/
- GitHub Desktop provides a nice GUI for using git

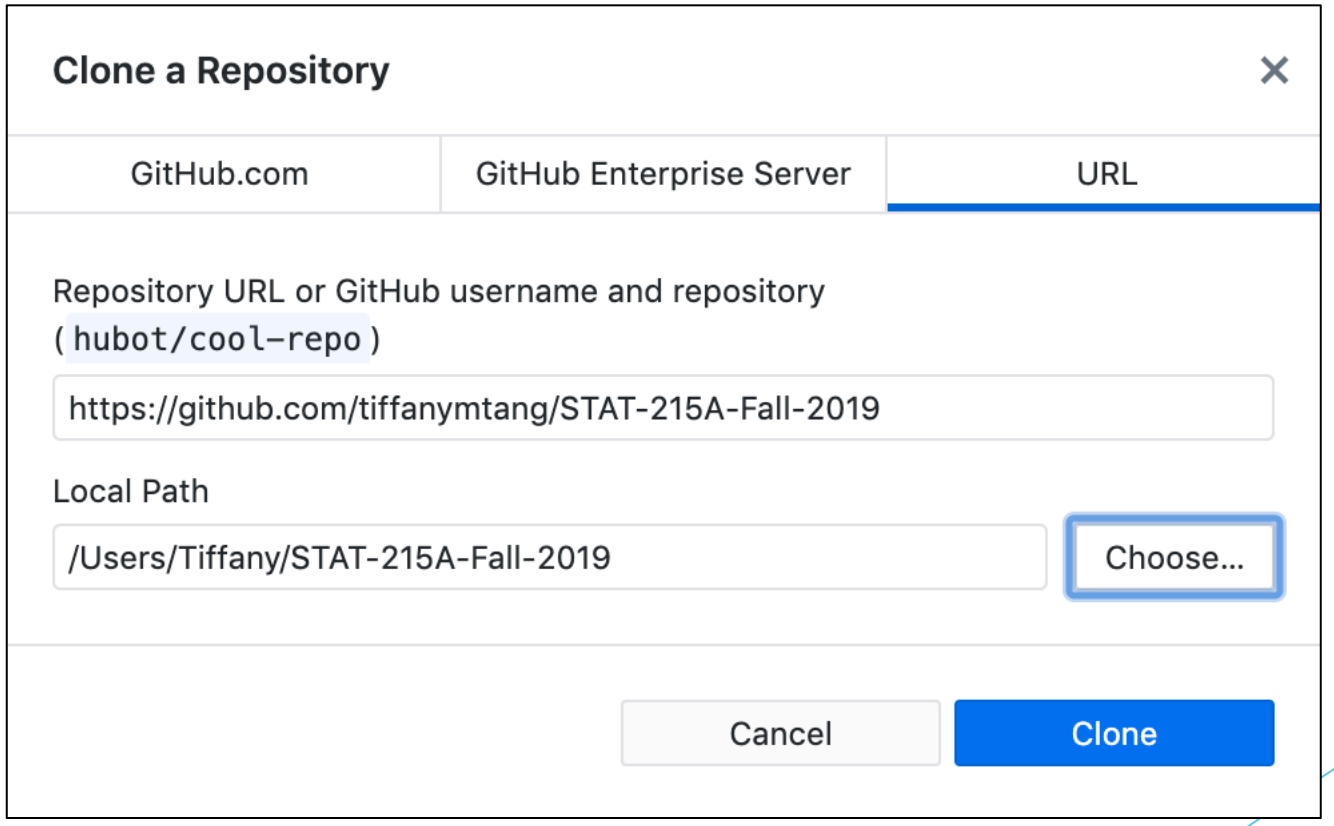
Let's clone my STAT-215A-Fall-2019 Repo using GitHub Desktop

1. Go to https://github.com/tiffanymtang/STAT-215A-Fall-2019 and click on **Clone or download → Open in Desktop**



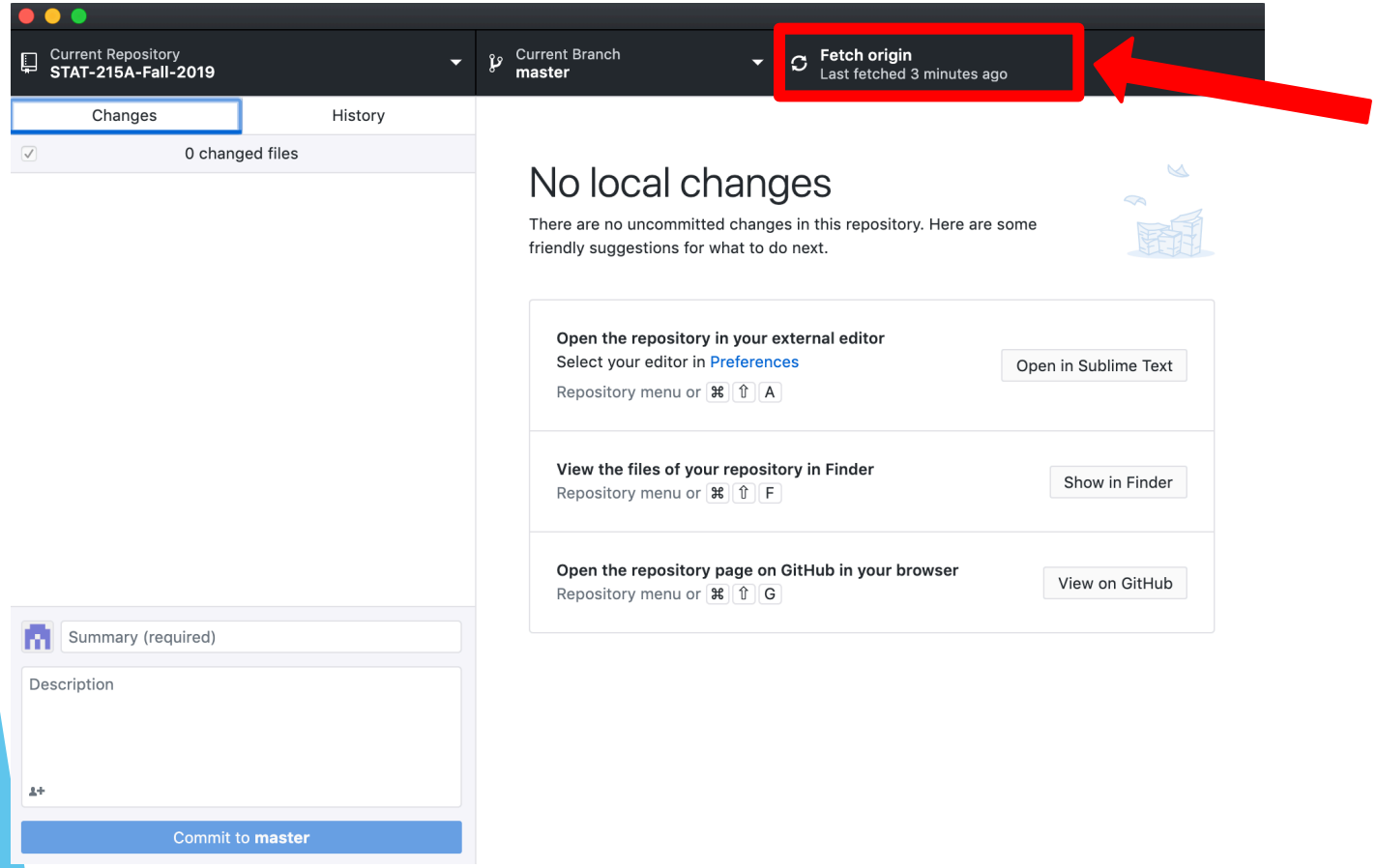
Let's clone my STAT-215A-Fall-2019 Repo using GitHub Desktop

2. Choose local path for repository and clone



Let's clone my STAT-215A-Fall-2019 Repo using GitHub Desktop

3. Each week during lab section, you can easily pull course material from the STAT-215A-Fall-2019 repo as I will be updating this repo each week



This button will say **pull** if there is new material to pull from the remote repo. When it does, simply click to pull material from the remote repo to your local repo. This is equivalent to typing **git pull** in the command line.

Adding our existing stat-215-a repo to GitHub Desktop

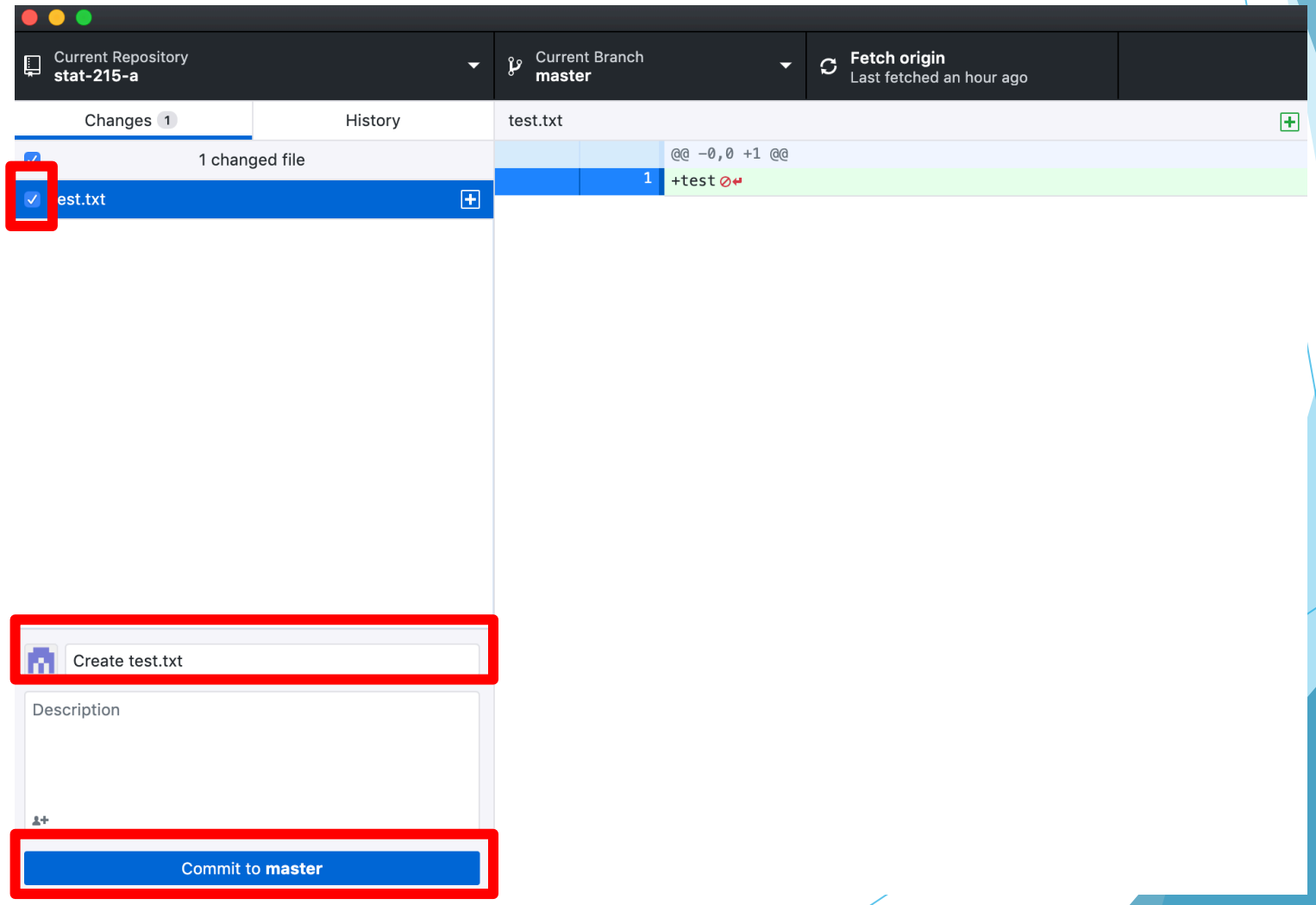
1. Click on the down arrow next to the **Current Repository** tab, then **add**, then **add existing repository**

2. Find and add your stat-215-a repository that you just created

3. After making changes to your local repo, add and commit these changes by clicking on the checkboxes next to the files/changes you'd like to add, write a short description, and click commit

Adding our existing stat-215-a repo to GitHub Desktop

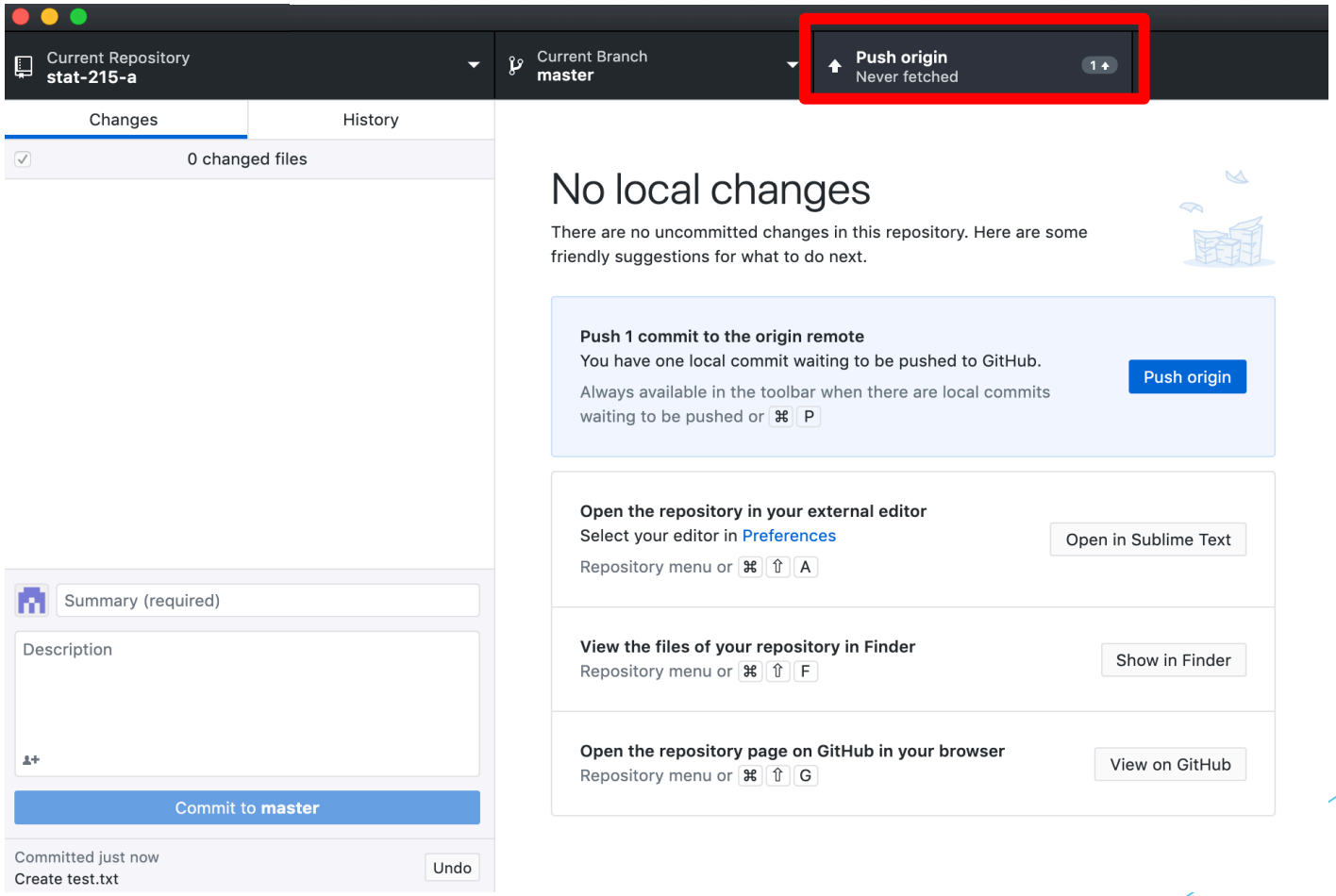
(1) Add changes by checking the box(s); equivalent to **git add test.txt**

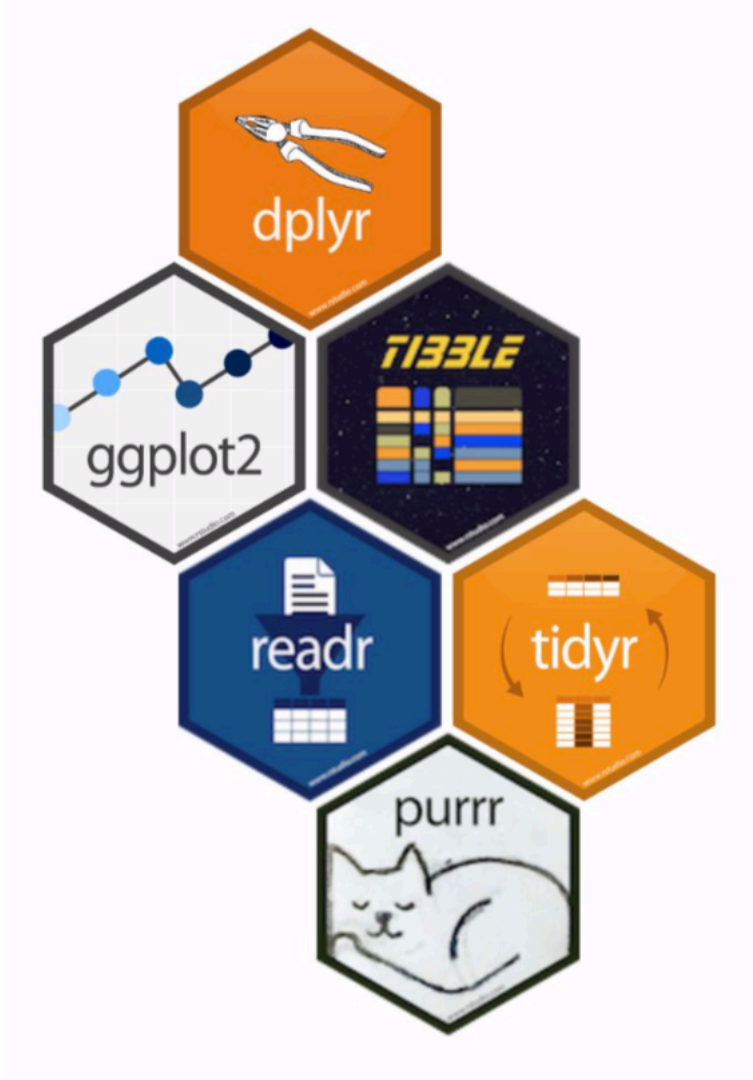


(2) Write a **description** and **commit** changes; equivalent to **git commit –m “Create test.txt”**

Adding our existing stat-215-a repo to GitHub Desktop

4. Finally, you can push the changes by clicking **push** in the upper right




dplyr
ggplot2
TIBBLE
readr
tidyr
purrr


tidyverse
www.rstudio.com

R Tidyverse

Tidy Data

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

For more on tidy data: http://vita.had.co.nz/papers/tidy-data.pdf
For an introduction to R and tidyverse: https://r4ds.had.co.nz/
For more advanced topics in R: https://adv-r.hadley.nz/

If you ever run into questions or need help with R:
Google and StackOverflow are your friend

Tidyverse is great, but not perfect:
https://github.com/matloff/TidyverseSkeptic


dplyr
ggplot2
TIBBLE
readr
tidyr
purrr

dplyr: Piping Operator (%>%)

- Pronounce “%>%” as “then”
- Piping is a way of chaining together functions so that you don’t have to keep redefining variables.
- Piping always puts the object being piped into the first argument of the function

Piping

```
> iris %>%
+   group_by(Species) %>%
+   summarise(n = n())
# A tibble: 3 x 2
  Species        n
  <fct>      <int>
1 setosa        50
2 versicolor    50
3 virginica     50
```

No Piping

```
> iris_by_species <- group_by(iris, Species)
> summarise(iris_by_species, n = n())
# A tibble: 3 x 2
  Species        n
  <fct>      <int>
1 setosa        50
2 versicolor    50
3 virginica     50
```

dplyr: “Grammar of Data Manipulation”

- Contains functions for editing variables in a data frame or tibble
  - **filter():** find rows where the specified condition is true
  - **select():** keep or remove certain variables/features
  - **rename():** for renaming columns
  - **mutate():** modifying/creating new columns
  - **group_by():** change the scope of each function from operating of the entire dataset to operating on groups
  - **summarise():** reduces multiple values down to a single summary
  - **arrange():** order rows in data frame by specific column(s)
  - Also see filter_*(), select_*(), rename_*(), mutate_*(), where * = if, at, all
- The first argument of each function is a data frame (or tibble)
- The result is always a new data frame (or tibble)

dplyr::filter()

- Finds rows where the specified condition is true
- Ex. Extract observations corresponding to the species “versicolor”

Piping + Filter

```
> iris %>%
+   filter(Species == "versicolor") %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          7.0         3.2          4.7         1.4 versicolor
2          6.4         3.2          4.5         1.5 versicolor
3          6.9         3.1          4.9         1.5 versicolor
4          5.5         2.3          4.0         1.3 versicolor
5          6.5         2.8          4.6         1.5 versicolor
6          5.7         2.8          4.5         1.3 versicolor
```

Base R

```
> head(iris[iris$Species == "versicolor", ])
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51          7.0         3.2          4.7         1.4 versicolor
52          6.4         3.2          4.5         1.5 versicolor
53          6.9         3.1          4.9         1.5 versicolor
54          5.5         2.3          4.0         1.3 versicolor
55          6.5         2.8          4.6         1.5 versicolor
56          5.7         2.8          4.5         1.3 versicolor
```

- See also filter_all(), filter_if(), and filter_at()

dplyr::select() and dplyr::rename()

- select(): keep certain variables, or remove certain variables
- rename(): rename columns
- Ex. Get Sepal.Length and Sepal.Width columns and rename to Length and Width, respectively

Piping + select + rename

```
> iris %>%
+   select(Sepal.Length, Sepal.Width) %>%
+   rename(Length = Sepal.Length,
+          Width = Sepal.Width) %>%
+   head()
  Length Width
1    5.1   3.5
2    4.9   3.0
3    4.7   3.2
4    4.6   3.1
5    5.0   3.6
6    5.4   3.9
```

Base R

```
> iris_sepal <- iris[, c("Sepal.Length", "Sepal.Width")]
> colnames(iris_sepal) <- c("Length", "Width")
> head(iris_sepal)
  Length Width
1    5.1   3.5
2    4.9   3.0
3    4.7   3.2
4    4.6   3.1
5    5.0   3.6
6    5.4   3.9
```

- See also select_all(), select_if(), select_at(), rename_all(), rename_if(), rename_at()

dplyr::select_if()

- select_if(): keep or remove certain variables if a condition holds
- Ex. Get numeric columns only

```
> iris %>%
+   select_if(is.numeric) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

dplyr::mutate()

- Create new variables consisting of functions of existing variables.
- Ex. Create a new variable which is the sum of the Sepal Length and Sepal Width

Piping + mutate

```
> iris %>%
+   mutate(Sepal.Sum = Sepal.Width + Sepal.Length) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1          5.1         3.5          1.4         0.2  setosa       8.6
2          4.9         3.0          1.4         0.2  setosa       7.9
3          4.7         3.2          1.3         0.2  setosa       7.9
4          4.6         3.1          1.5         0.2  setosa       7.7
5          5.0         3.6          1.4         0.2  setosa       8.6
6          5.4         3.9          1.7         0.4  setosa       9.3
```

Base R

```
> iris$Sepal.Sum <- iris$Sepal.Width + iris$Sepal.Length
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1          5.1         3.5          1.4         0.2  setosa       8.6
2          4.9         3.0          1.4         0.2  setosa       7.9
3          4.7         3.2          1.3         0.2  setosa       7.9
4          4.6         3.1          1.5         0.2  setosa       7.7
5          5.0         3.6          1.4         0.2  setosa       8.6
6          5.4         3.9          1.7         0.4  setosa       9.3
```

- See also mutate_all(), mutate_if(), and mutate_at()

dplyr::mutate_at()

- Mutate at existing variables via some function
- Ex. Multiply each of Sepal.Length and Sepal.Width by 2

```
> iris %>%
+   mutate_at(vars(contains("Sepal")), list(~ 2 * .)) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1         10.2         7.0          1.4         0.2  setosa      17.2
2          9.8         6.0          1.4         0.2  setosa      15.8
3          9.4         6.4          1.3         0.2  setosa      15.8
4          9.2         6.2          1.5         0.2  setosa      15.4
5         10.0         7.2          1.4         0.2  setosa      17.2
6         10.8         7.8          1.7         0.4  setosa      18.6
```

dplyr::group_by() and dplyr::summarise()

- group_by(): changes the scope of each function from operating on the entire dataset to operating on it group-by-group.
- summarise(): reduces multiple values down to a single summary.
- Ex. Compute the mean and median Sepal.Length for each Species

```
> iris %>%
+   group_by(Species) %>%
+   summarise(Sepal.Length.mean = mean(Sepal.Length),
+             Sepal.Length.median = median(Sepal.Length))
# A tibble: 3 x 3
  Species    Sepal.Length.mean Sepal.Length.median
  <fct>                  <dbl>                <dbl>
1 setosa                  5.01                  5
2 versicolor              5.94                  5.9
3 virginica               6.59                  6.5
```

dplyr::arrange()

- Order rows by an expression involving its variables.
- Ex. Order rows first by decreasing Petal Length and then by increasing Sepal Width (if there are ties among the Petal Length)

```
> iris %>%
+   arrange(desc(Petal.Length), Sepal.Width) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.7         2.6          6.9         2.3 virginica
2          7.7         2.8          6.7         2.0 virginica
3          7.7         3.8          6.7         2.2 virginica
4          7.6         3.0          6.6         2.1 virginica
5          7.9         3.8          6.4         2.0 virginica
6          7.3         2.9          6.3         1.8 virginica
```


dplyr
ggplot2
TIBBLE
readr
tidyr
purrr

tidyr: “Grammar of Tidy Data”

- Used to be called “reshape2”
- Contains functions for changing the shape of the data from long-form to wide-form.
- Main functions
  - **spread():** to go from long to wide format
  - **gather():** to go from wide to long format

tidyr::spread() vs tidyr::gather()

spread: convert long format to wide format

```
> iris_wide <- iris_long %>%
+   spread(key = "Variable", value = "Value")
> head(iris_wide)
   id    Species Petal.Length Petal.Width Sepal.Length Sepal.Sum Sepal.Width
1   1     setosa          1.4         0.2          5.1       8.6         3.5
2  10     setosa          1.5         0.1          4.9       8.0         3.1
3 100 versicolor          4.1         1.3          5.7       8.5         2.8
4 101  virginica          6.0         2.5          6.3       9.6         3.3
5 102  virginica          5.1         1.9          5.8       8.5         2.7
6 103  virginica          5.9         2.1          7.1      10.1         3.0
```

gather: convert wide format to long format

```
> iris_long <- iris %>%
+   rownames_to_column("id") %>%
+   gather(key = "Variable", value = "Value", -Species, -id)
> head(iris_long)
  id Species     Variable Value
1  1  setosa Sepal.Length   5.1
2  2  setosa Sepal.Length   4.9
3  3  setosa Sepal.Length   4.7
4  4  setosa Sepal.Length   4.6
5  5  setosa Sepal.Length   5.0
6  6  setosa Sepal.Length   5.4
```


dplyr
ggplot2
tibble
readr
tidyr
purrr

ggplot2: “Grammar of Graphics”

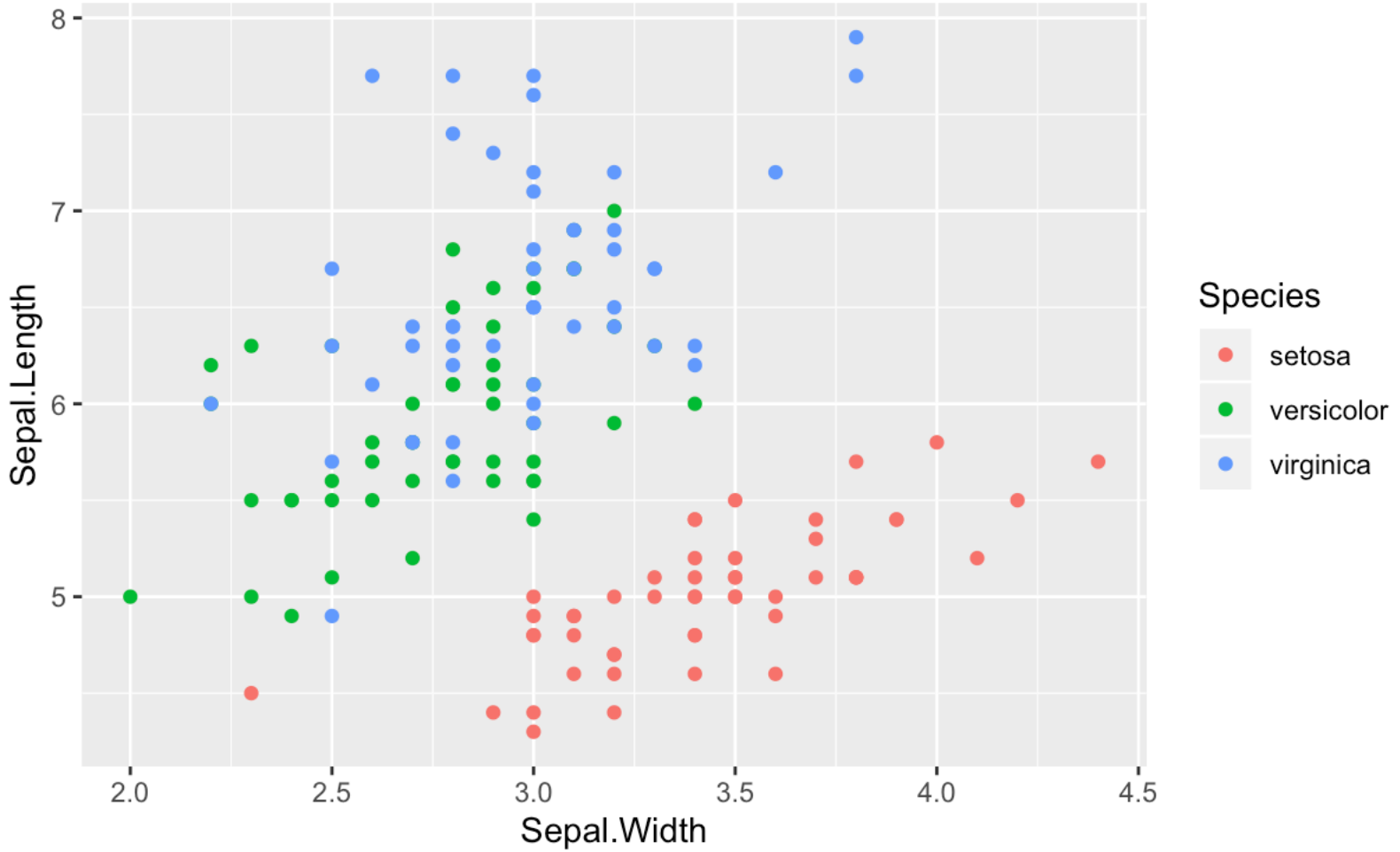
- Always begin with **ggplot(data.frame)**
- Then add layers to the base plot by using + (similar to piping %>%)
  - Examples of layers you may want to add:
    - **geom_point()**
    - **geom_histogram()**
    - **geom_text()**
    - **theme()**
    - **scale_x_continuous()**
    - **labs()**

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length, color = Species))
```

- You can do many many things with ggplot, and I can’t even begin to list the different layers, color themes, formatting options that you can specify in ggplot. **Google will be your best friend here!**

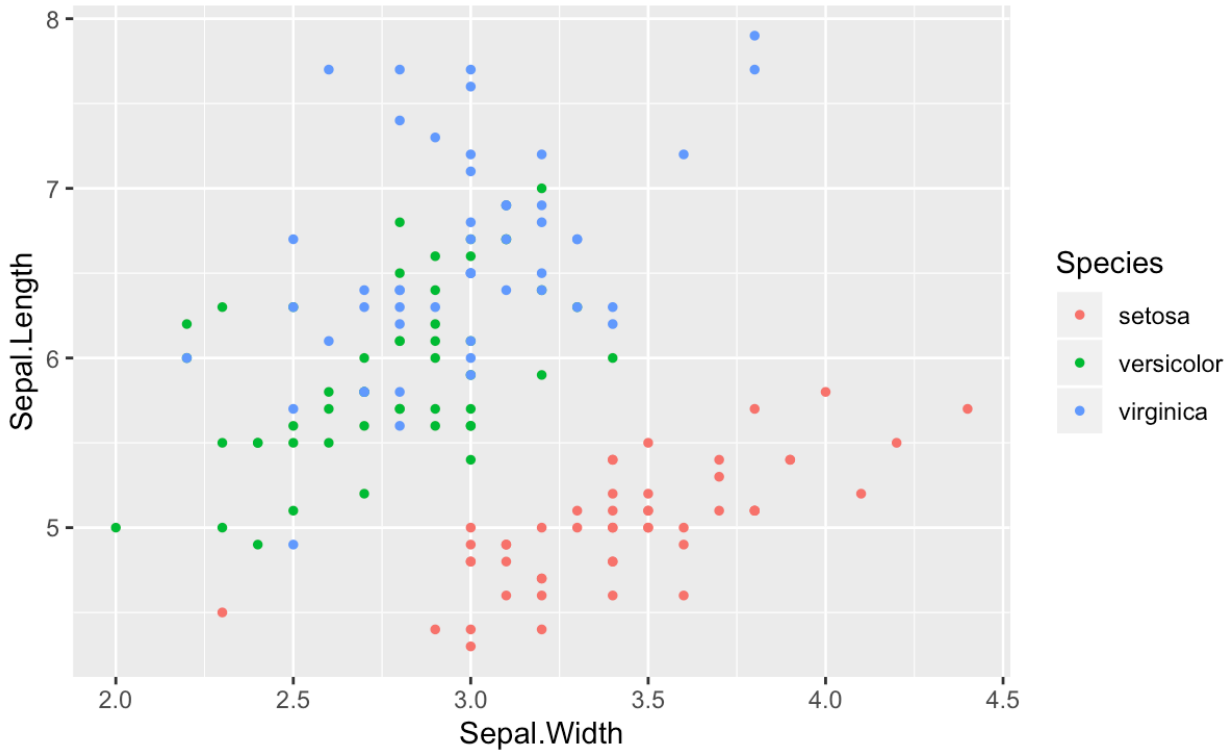
ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length, color = Species))
```

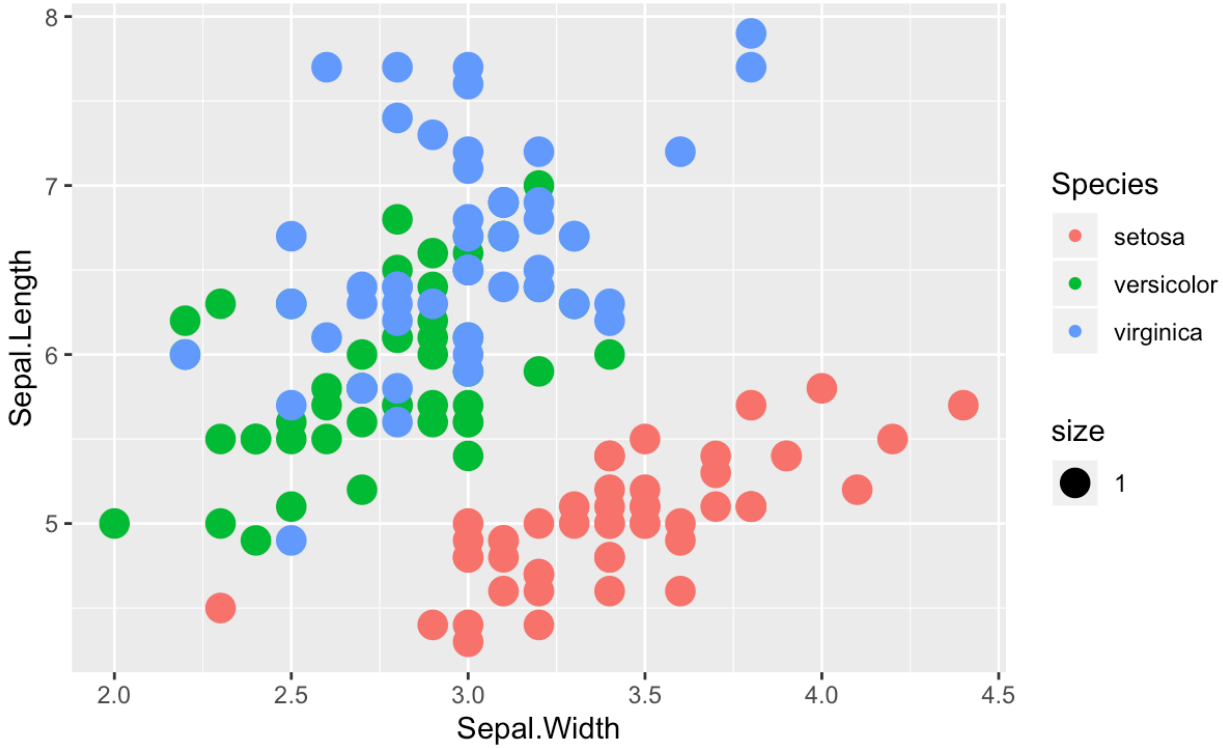


ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length,
+                  color = Species), size = 1)
```

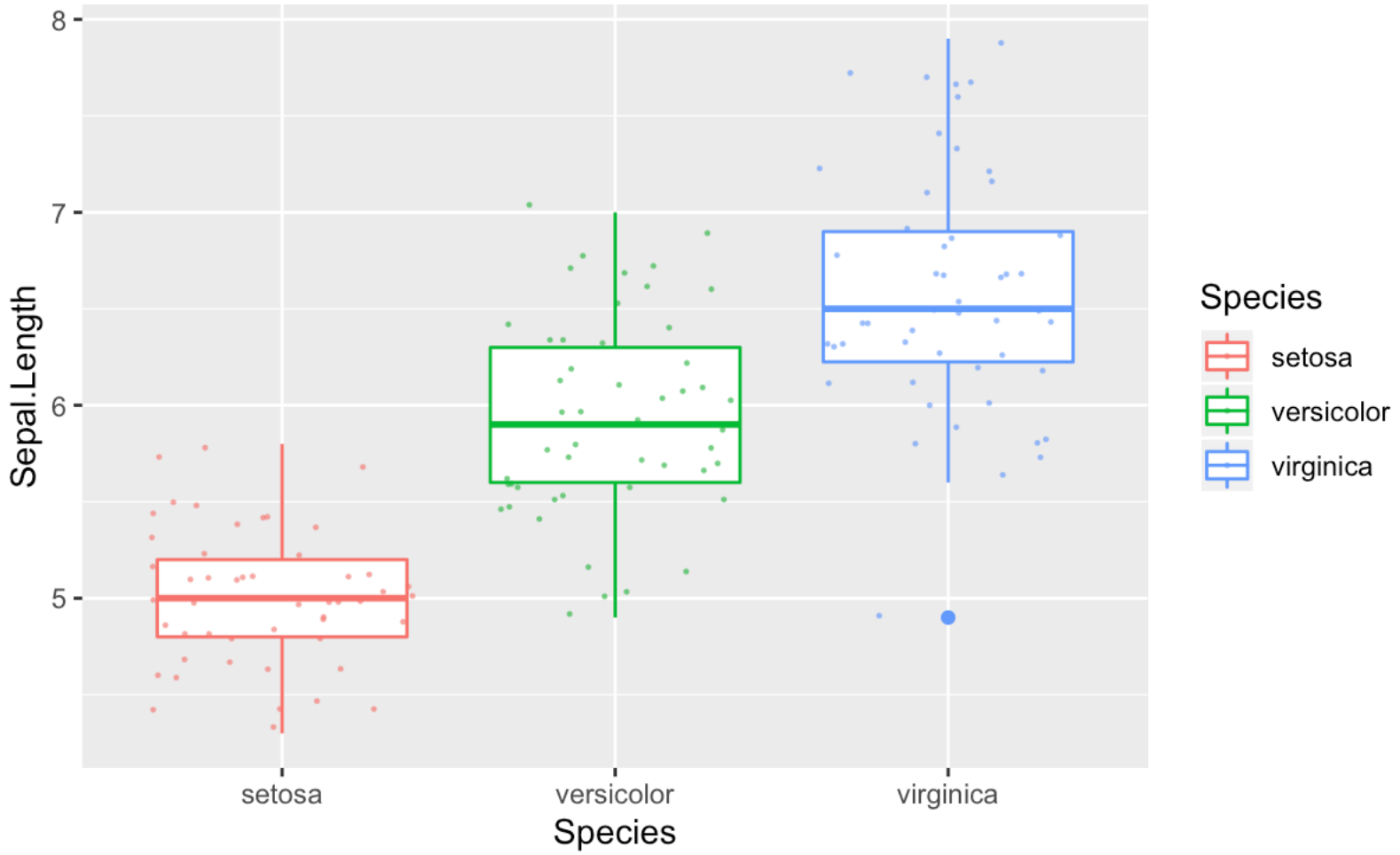



```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length,
+                  color = Species, size = 1))
```



ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   aes(x = Species, y = Sepal.Length, color = Species) +
+   geom_boxplot() +
+   geom_jitter(size = .25, alpha = .5)
```



Let’s get some practice

For Tomorrow

- Lab 0 is due tomorrow (Sep 6)
- Lab 0 is not for a grade, but at a minimum, please submit an empty project/lab so that we can ensure your GitHub is working accordingly
- Tentative agenda for tomorrow:
  - Workflow + R Markdown
  - Tips and tricks to make life easier in R/R Markdown
  - Lab 1 Assigned