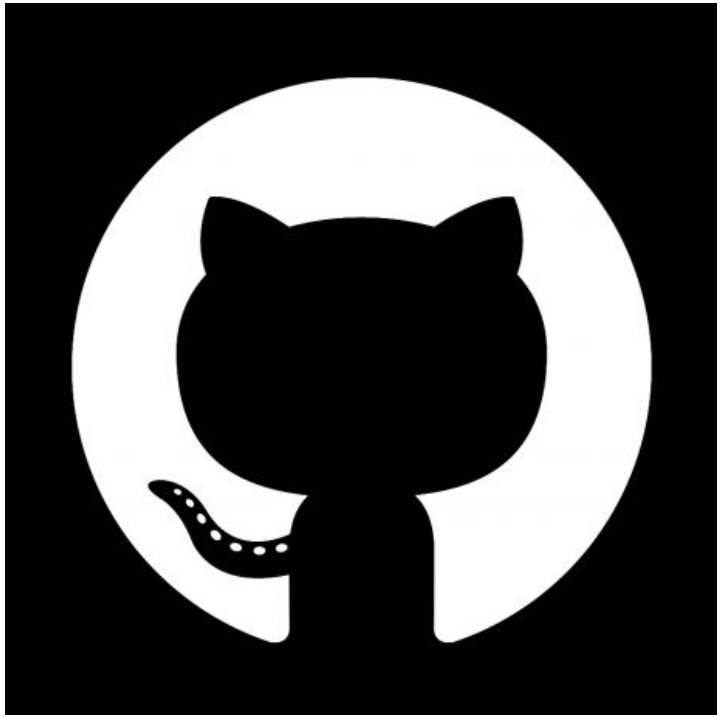
STAT 215A Fall 2017
Week 1

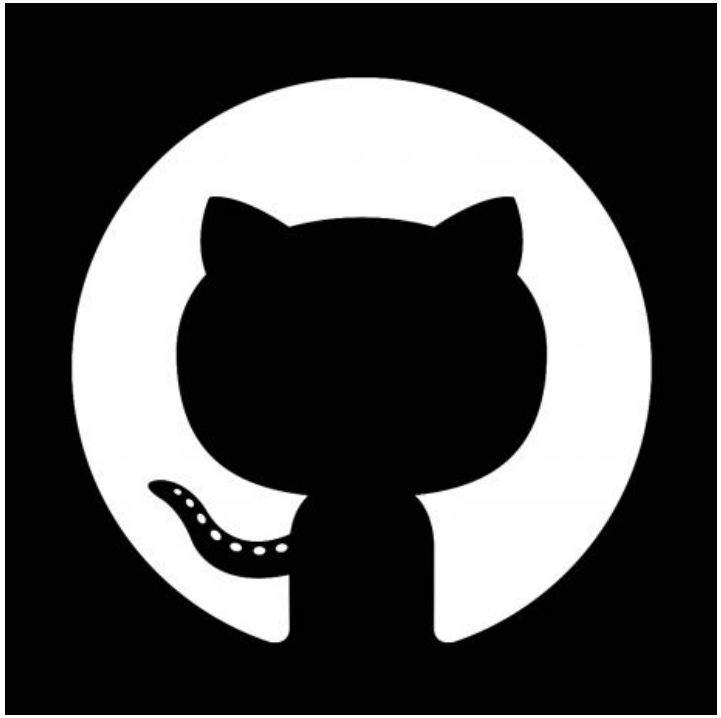
Rebecca Barter
08/25/2017

Git & Github



R & the “tidyverse”

Git & Github



GitHub repository for class materials:

https://github.com/rlbarter/STAT-215A-Fall-2017

GitHub repository that you will clone and use to submit your projects:

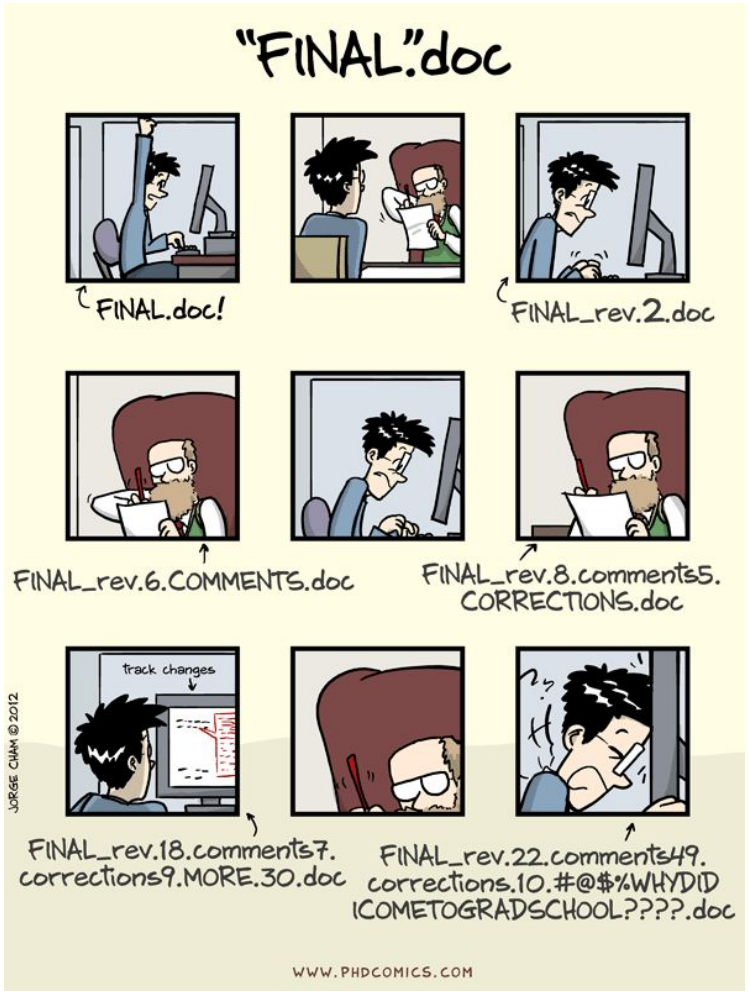
https://github.com/rlbarter/stat215a

My goals for today

Give everyone at least a vague understanding of what Git and GitHub are and why they are useful.

Have everyone (1) set up a GitHub repository for this class, (2) add me as a collaborator, and (3) push something so that I can see it.

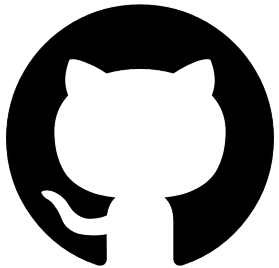
If there is time... learn some tidyverse!


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

What are Git and Github

Local Git repository:
You have a local version of the folder. Its history is saved in a .git file. Only you can see changes you make here.

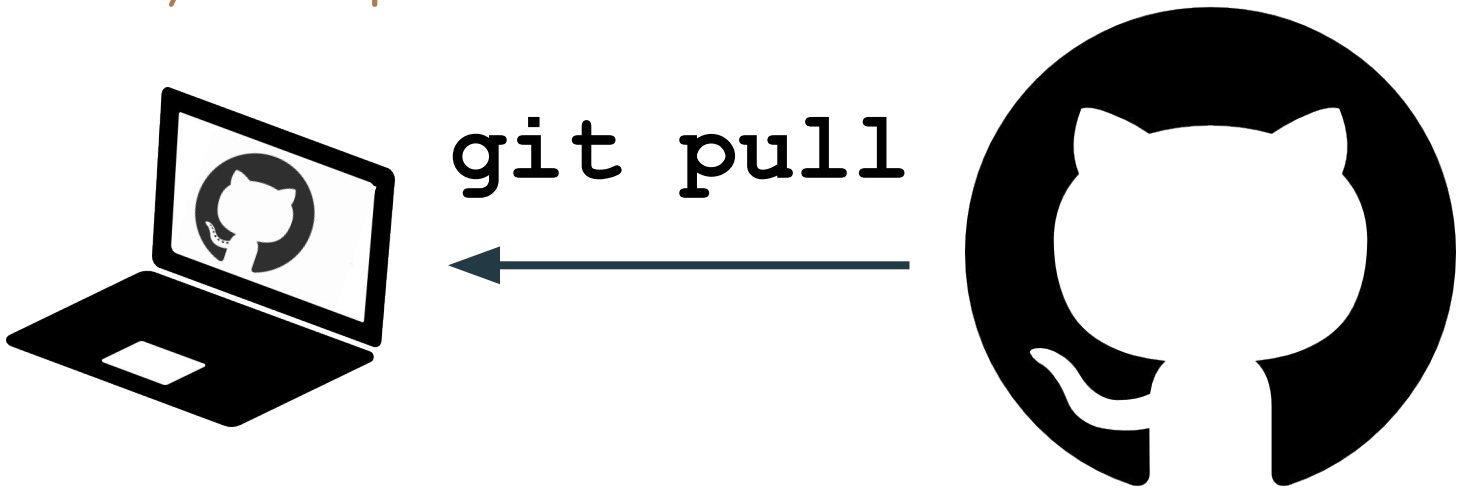
Remote GitHub repository:
On the GitHub website lives a remote version of the folder that everyone can see.



What is the point of all of this?

Everyone can have their own local version of a project

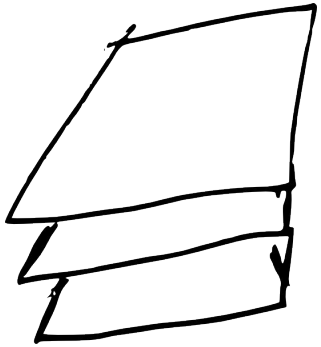
To make sure you have the most recent version, you should always first pull from the server



What is the point of all of this?

When you want to make changes, you can do so freely without other people seeing what you do.

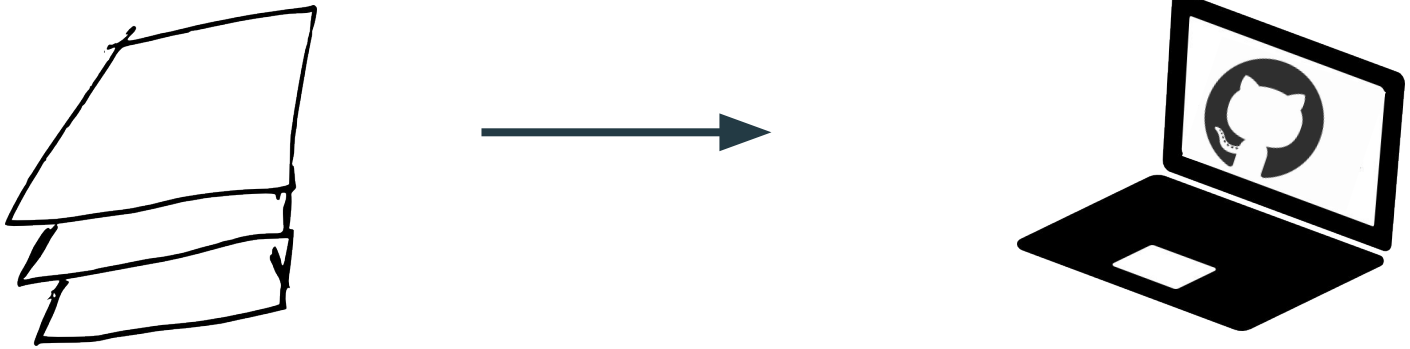
When you have made some changes and want to be able to **save your current checkpoint as a snapshot** you can **add and commit**.



What is the point of all of this?

When you want to make changes, you can do so freely without other people seeing what you do.

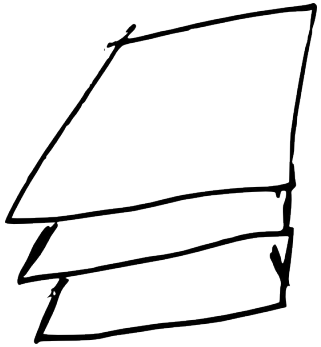
When you have made some changes and want to be able to **save your current checkpoint as a snapshot** you can **add and commit**.



What is the point of all of this?

When you want to make changes, you can do so freely without other people seeing what you do.

When you have made some changes and want to be able to **save your current checkpoint as a snapshot** you can **add and commit**.

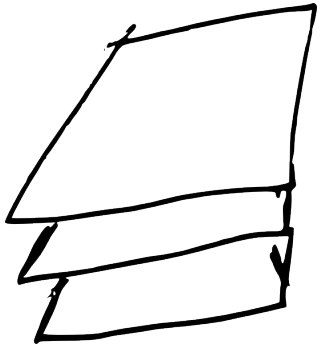


`git add important_file.txt`

What is the point of all of this?

When you want to make changes, you can do so freely without other people seeing what you do.

When you have made some changes and want to be able to **save your current checkpoint as a snapshot** you can **add and commit**.



```
git add important_file.txt
```

```
git commit -m "added a
description of the
clustering algorithm"
```

What is the point of all of this?

When you are ready for your changes to be available to everyone, first you want to

`git pull`

so that you don't create conflicts, then you can push all of your commits to the server

Summary

Pull most current content from GitHub remote (may need to deal with conflicts):

```
git pull
```

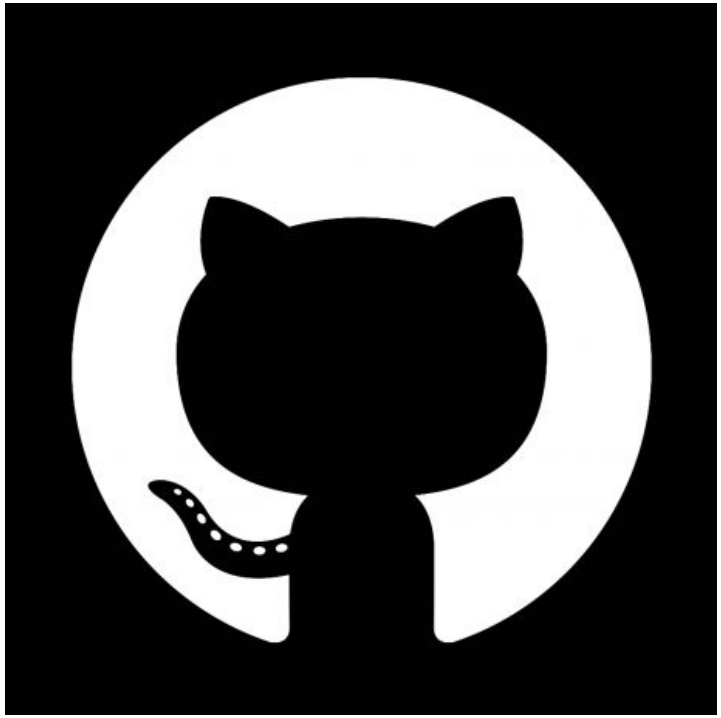
Make local edits to interesting_file.txt. Take a snapshot.

```
git add interesting_file.txt
git commit -m “clarified explanation of clustering”
```

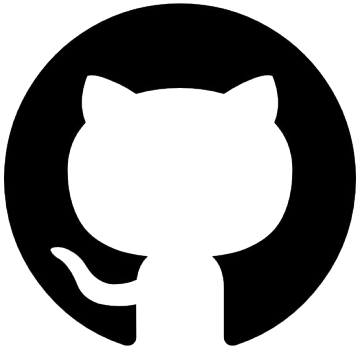
Push to GitHub remote

```
git push
```

Submitting your projects

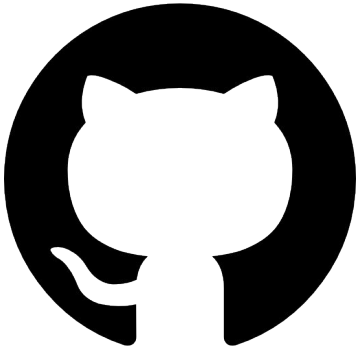


Submitting your projects

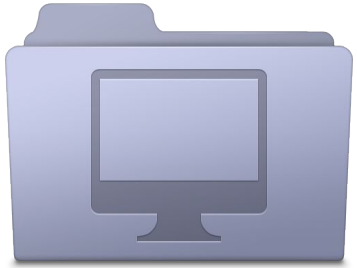


```
git clone https://gith...
```

Submitting your projects

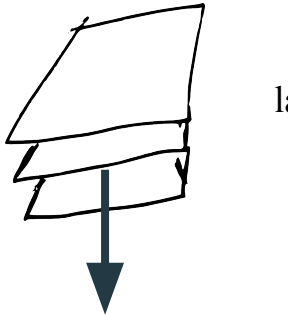


`git clone https://gith...`

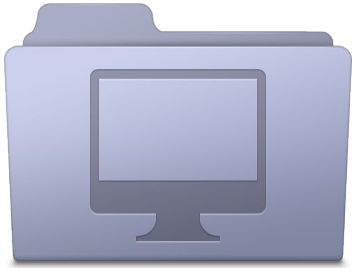


`stat215a/`

Submitting your projects

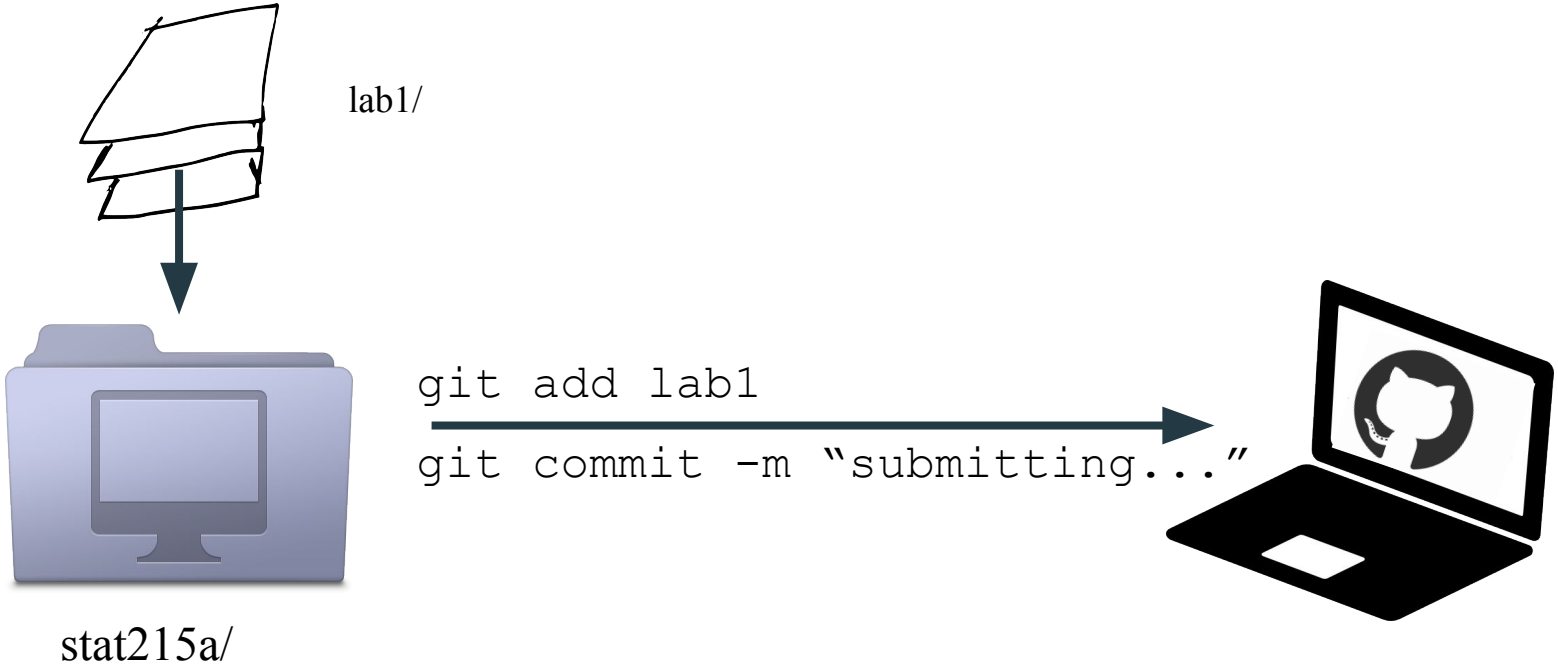


lab1/

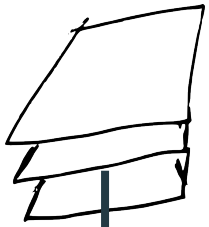


stat215a/

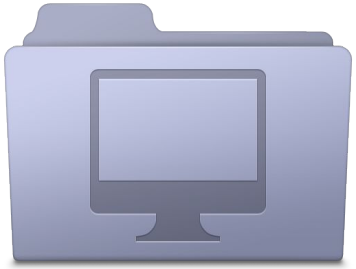
Submitting your projects



Submitting your projects



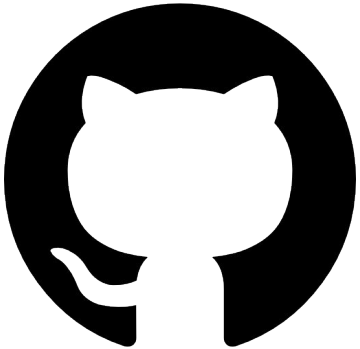
lab1/



stat215a/

```
git add lab1
git commit -m "submitting..."
```

```
git push
```



Setting up our Repositories for this class

Install Git on your system: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Sign up for GitHub: https://github.com/

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 our Repositories for this class

Locally on your machine, clone my stat215a repository:

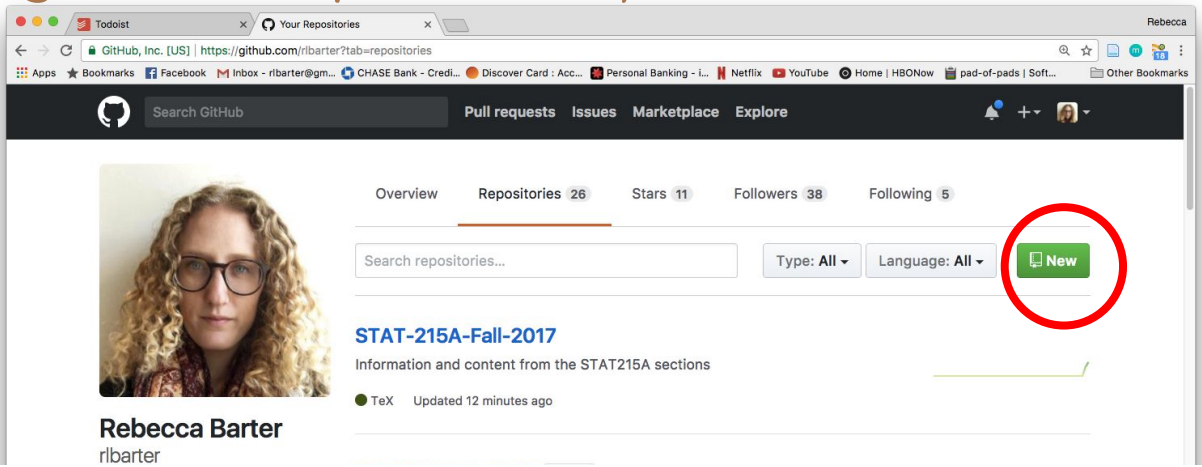
```
git clone https://github.com/rlbarter/stat215a
```

This will create a copy of the repository on your own computer.

Setting up our Repositories for this class

On the github website, log in and create a **private** repository called *stat215a*

Add me (*rlbarter*) as a collaborator for this repository (check out settings on the repo website).

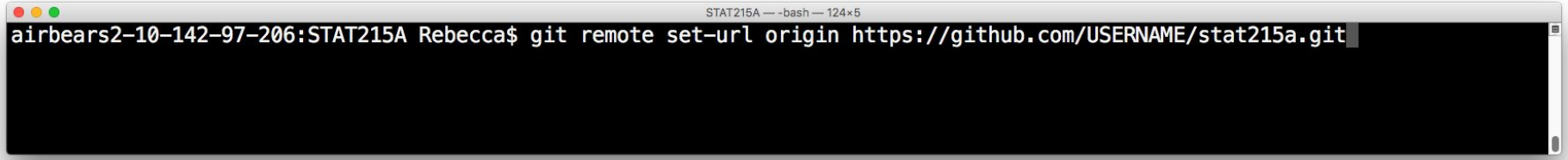


Setting up our Repositories for this class

Locally on your machine, set the origin of your local repository to be the remote repo that you just created:

```
git remote set-url origin
   https://github.com/USERNAME/stat215a.git
```

This tells git which remote repo to push and pull from



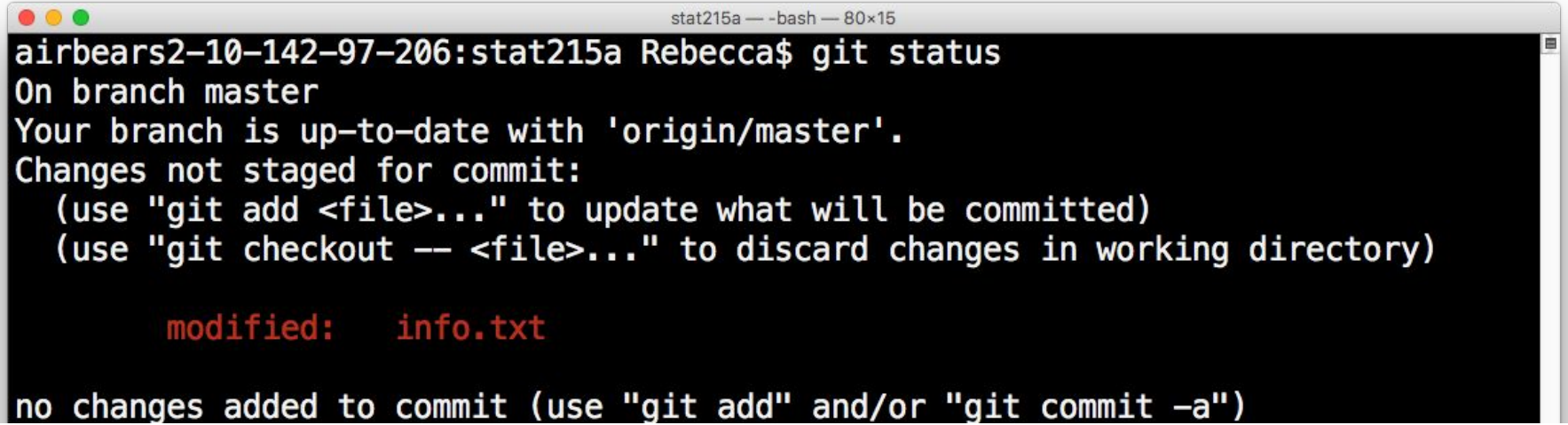
Setting up our Repositories for this class

In your *stat215a* folder, you will find a file called *info.txt*

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

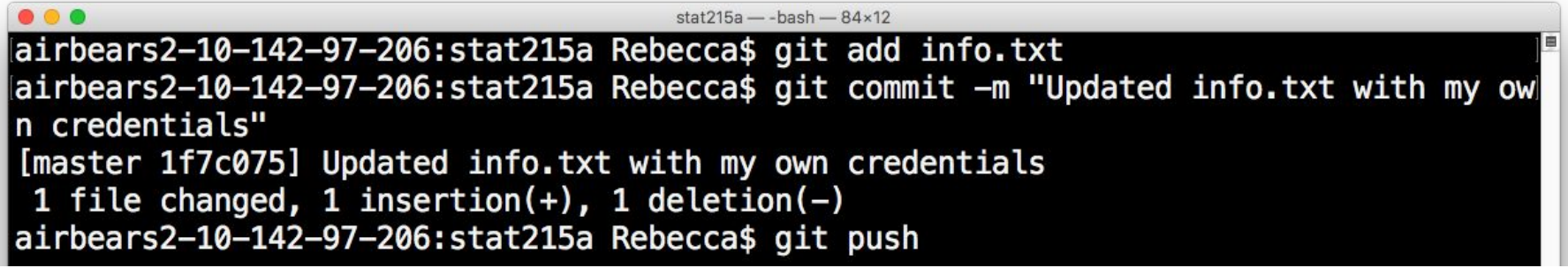
Setting up our Repositories for this class

Check `git status`. You should see that *info.txt* has been modified.



Setting up our Repositories for this class

Add, commit and push these changes to your remote repository.



Tidyverse

R Markdown & R Sweave (knitr)

Filetypes for combining text and code (but easily hide code when compiling)

R Markdown (.Rmd):

- Essentially a markdown document
- Difficult (but not impossible) to add LaTeX preamble
- Can use LaTeX equations
- Better for html output (but can also produce pdf output)
- Code chunks look like:

```
```{r, echo = FALSE}
data <- read.csv("dat.txt")
```
```

R Sweave/knitr (.Rnw):

- Essentially a LaTeX document
- Can add LaTeX preamble
- Can use LaTeX equations
- Better for pdf output
- Code chunks look like:

```
<<echo = FALSE>>=
data <- read.csv("dat.txt")
@
```

The tidyverse

Tidy data:

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

Resources:

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

http://vita.had.co.nz/papers/tidy-data.pdf


dplyr
ggplot2
tibble
readr
tidyr
purrr

dplyr: intro

“Grammar of data manipulation”

Contains functions for editing the variables in a df/tibble

- `filter()`
- `select()`
- `mutate()`
- `group_by()`
- `summarise()`
- `arrange()`

The first argument of each function is a data frame.

The result is a new data frame

dplyr: piping (%>%)

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

dplyr: piping (%>%)

Pronounce "%>%" as "then"

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

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

dplyr: filter

Finds rows where the specified condition is true.

You can do more with filter using `filter_all()`, `filter_if()`, and `filter_at()`.

dplyr: filter

```
> head(filter(iris, Species == "versicolor"))
  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
> 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
```

dplyr: select and rename

Keep only certain variables, or remove only certain variables

You can do more with select using `select_all()`, `select_if()`, and `select_at()`.

dplyr: select and rename

```
> iris %>%
+   filter(Species == "versicolor") %>%
+   select(Sepal.Length, Species) %>%
+   head
  Sepal.Length    Species
1          7.0 versicolor
2          6.4 versicolor
3          6.9 versicolor
4          5.5 versicolor
5          6.5 versicolor
6          5.7 versicolor
```

dplyr: mutate

Create new variables consisting of functions of existing variables.

You can do more with select using `mutate_all()`, `mutate_if()`, and `mutate_at()`.

You can make more complicated conditions using `case_when()`, `if_else()`, and more...

dplyr: mutate

Add a variable equal to the sum of Sepal.Length and Sepal.Width

```
> iris %>%
+   filter(Species == "versicolor") %>%
+   mutate(Sepal.Sum = Sepal.Length + Sepal.Width) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Sepal.Sum
1          7.0         3.2          4.7         1.4 versicolor      10.2
2          6.4         3.2          4.5         1.5 versicolor       9.6
3          6.9         3.1          4.9         1.5 versicolor      10.0
4          5.5         2.3          4.0         1.3 versicolor       7.8
5          6.5         2.8          4.6         1.5 versicolor       9.3
6          5.7         2.8          4.5         1.3 versicolor       8.5
```

dplyr: mutate

Multiply each of Sepal.Length and Sepal.Width by 2

```
> iris %>%
+   filter(Species == "versicolor") %>%
+   mutate_at(vars(contains("Sepal")), funs(2 * .)) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1         14.0         6.4          4.7         1.4 versicolor
2         12.8         6.4          4.5         1.5 versicolor
3         13.8         6.2          4.9         1.5 versicolor
4         11.0         4.6          4.0         1.3 versicolor
5         13.0         5.6          4.6         1.5 versicolor
6         11.4         5.6          4.5         1.3 versicolor
```

dplyr: group_by

Changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

dplyr: summarise

Reduces multiple values down to a single summary.

dplyr: group_by & summarise

```
> iris %>%
+   group_by(Species) %>%
+   summarise(Sepal.Length.mean = mean(Sepal.Length))
# A tibble: 3 x 2
     Species Sepal.Length.mean
      <fctr>              <dbl>
1     setosa              5.006
2 versicolor              5.936
3  virginica              6.588
```


dplyr
ggplot2
tibble
readr
tidyr
purrr

tidyr: intro

“Grammar of tidy data”

(Used to be called “reshape2”)

Contains functions for changing the shape of the data from long-form to wide-form.

The main functions are

- `spread()`
- `gather()`

tidyr: gather

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed.

`gather(df, key, value, names of` *`variables to gather/exclude`*`)`

tidyr: gather

```
> iris %>%
+   gather(key = "Variable", value = "Value", -Species) %>%
+   arrange(Species, Variable)
      Species     Variable Value
1      setosa Petal.Length   1.4
2      setosa Petal.Length   1.4
3      setosa Petal.Length   1.3
4      setosa Petal.Length   1.5
5      setosa Petal.Length   1.4
6      setosa Petal.Length   1.7
7      setosa Petal.Length   1.4

        ...

240 versicolor Petal.Length   4.0
241 versicolor Petal.Length   4.4
242 versicolor Petal.Length   4.6
243 versicolor Petal.Length   4.0
244 versicolor Petal.Length   3.3
245 versicolor Petal.Length   4.2
```

tidyr: gather

All the variables that were not explicitly ignored become the “key”

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

The corresponding values become the “value”

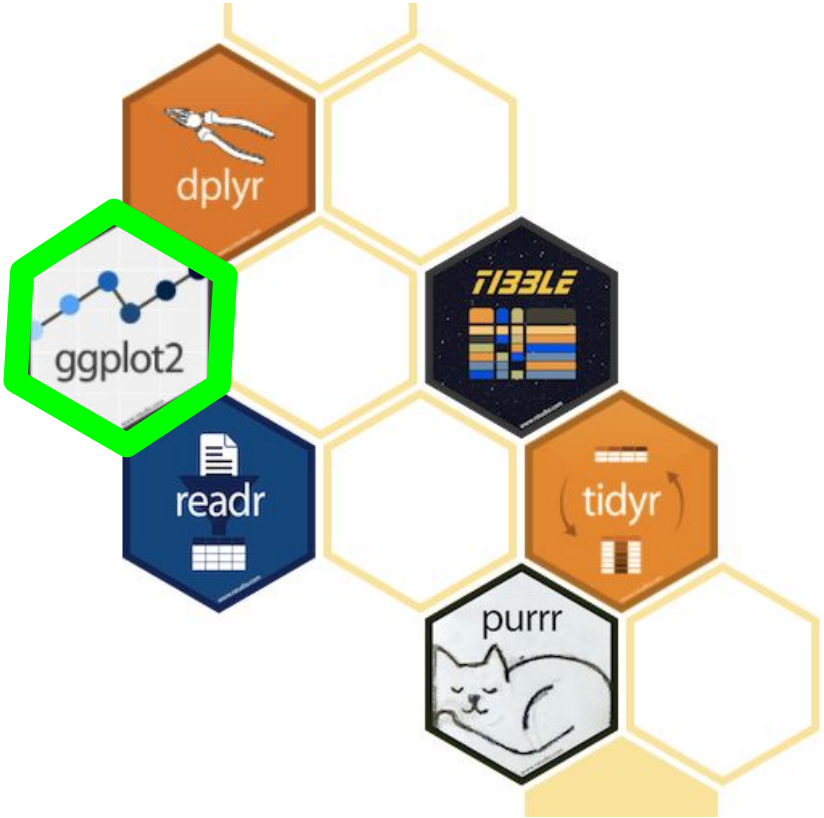
tidyr: spread

Spreads a key-value pair across multiple columns

```
spread(df, key, value, ...)
```

tidyr: spread

```
> iris_long <- iris %>%
+   mutate(row = row_number()) %>%
+   gather(key = "Variable", value = "Value", -Species, -row)
> iris_wide <- iris_long %>%
+   spread(key = Variable, value = Value)
> head(iris_wide)
  Species row Petal.Length Petal.Width Sepal.Length Sepal.Width
1  setosa   1          1.4         0.2          5.1         3.5
2  setosa   2          1.4         0.2          4.9         3.0
3  setosa   3          1.3         0.2          4.7         3.2
4  setosa   4          1.5         0.2          4.6         3.1
5  setosa   5          1.4         0.2          5.0         3.6
6  setosa   6          1.7         0.4          5.4         3.9
```


dplyr
ggplot2
tibble
readr
tidyr
purrr

ggplot2: intro

“Grammar of graphics”

- `ggplot()`

The first argument of ggplot() is a data frame.

- **`geom_point()`**
- **`geom_histogram()`**
- **`geom_text()`**

The first argument of all other functions is a ggplot() object

- **`theme()`**
- **`scale_x_continuous()`**

ggplot2: intro

Instead of linking functions together by piping `%>%`, ggplot has a similar syntax which adds layers using `+`.

We always start with a base `ggplot` layer.

```
ggplot(data) + ...
```

ggplot2: intro

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