~/>_

Data and dplyr

~/> previously …

git

– A Ledger of your actions,
– that can replay what you did,
– keep track of what others did,
– and help branch and merge code

git: local use case

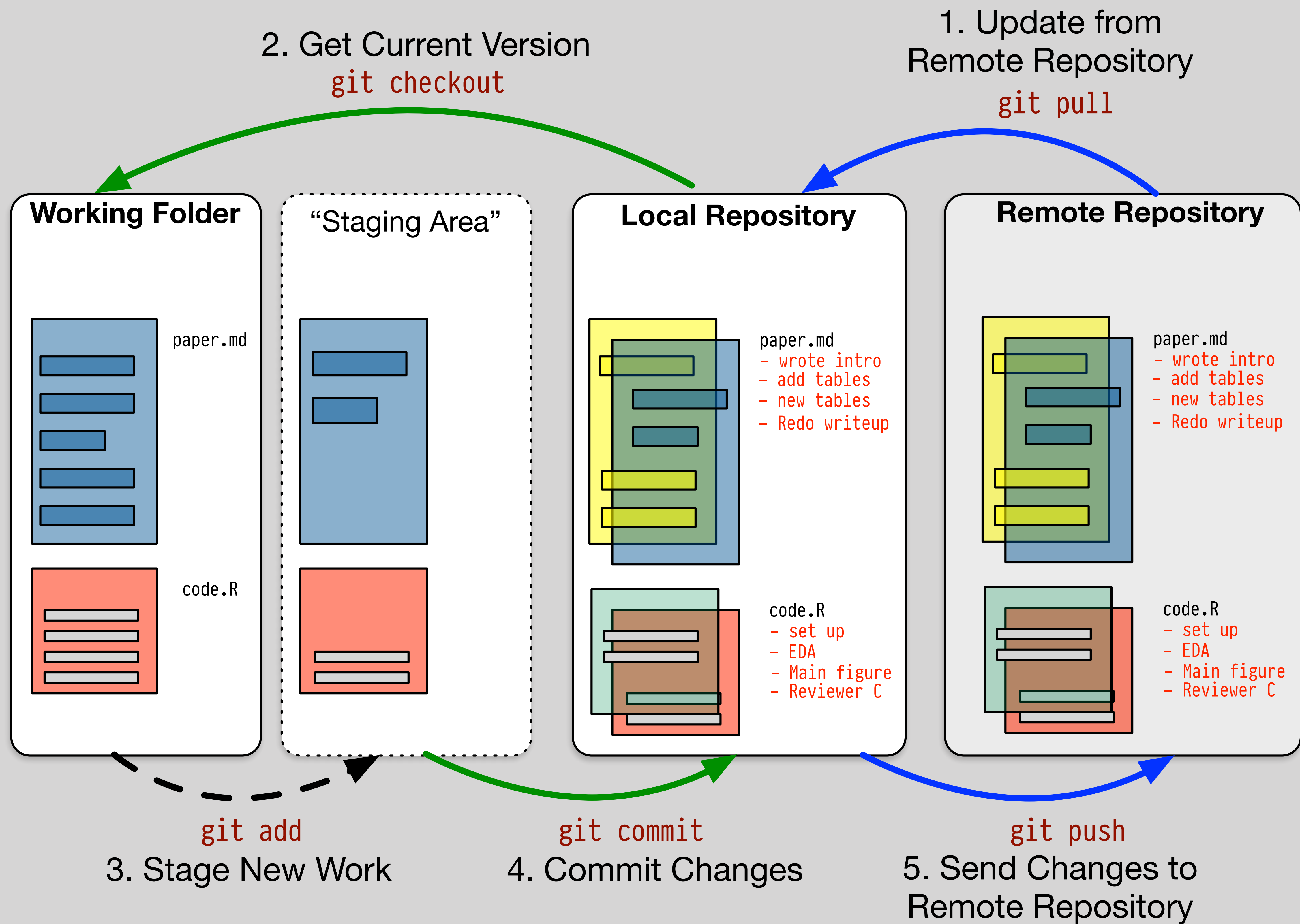
- Your project lives on your Computer
- You add and commit changes as you go
- That’s it.
- It’s just a ledger

```
~/> git add new_analysis.r

~/> git commit -m "Fixed MLE issue"
```

git: GitHub use case

- Your project lives on GitHub
- There's a working copy on your Computer
- You add and commit changes locally
- You push the changes to GitHub



2. Get Current Version
git checkout
1. Update from
Remote Repository
git pull
Working Folder
"Staging Area"
Local Repository
Remote Repository
paper.md
code.R
paper.md
- wrote intro
- add tables
- new tables
- Redo writeup
code.R
- set up
- EDA
- Main figure
- Reviewer C
paper.md
- wrote intro
- add tables
- new tables
- Redo writeup
code.R
- set up
- EDA
- Main figure
- Reviewer C
git add
3. Stage New Work
git commit
4. Commit Changes
git push
5. Send Changes to
Remote Repository

git: other use cases

- You are experimenting, or collaborating
- You can make **branches** of your project
- You add and commit to the branch
- You merge changes into the **master** branch

~/>_

R and the tidyverse

```r
library(tidyverse)
```

```
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
```

- ggplot2 ◀ Draw graphs
- tibble ◀ Nicer data tables
- tidyr ◀ Tidy your data
- readr ◀ Get data into R
- purrr ◀ Cool functional programming stuff
- dplyr ◀ Action verbs for manipulating data

R and the tidyverse

```
> install.packages("devtools")
> library(devtools)

> install_github("kjhealy/socviz")
> library(socviz)
```

FOUR THINGS TO KNOW ABOUT R

1: Everything has a Name

```
my_numbers
data
p
```

Some names are forbidden

```
FALSE  TRUE  Inf

for  if   break

function
```

2. Everything is an Object

```
letters
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
"m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

You **create** objects by **assigning** a thing to a name

named thing

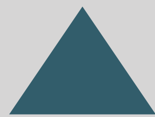
"gets"

this stuff

```
my_numbers <- c(1, 2, 3, 1, 3, 5, 25)
```

You create objects by assigning a thing to a name

```
my_numbers <- c(1, 2, 3, 1, 3, 5, 25)
```



The assignment operator performs the action of creating objects. Use the keyboard shortcut to type it:

`option -` Mac

`alt -` Windows

3. You do things using functions and operators

named thing

"gets"

this stuff

```
my_numbers <- c(1, 2, 3, 1, 3, 5, 25)
```

c() is a function that takes comma-separated numbers or strings and joins them together into a vector

Some operators

Operators	Type
`<-` or `=`	Assignment ("gets")
`+, -, *, /, ^`	Arithmetic
`<, >, <=, >= y, ==, !=`	Relational
`&, &&, |, ||, !`	Logical
`%*%, %in%, %>%`	Special

Functions

take inputs, perform actions, produce outputs

Functions have parentheses
at the end of their name.
This is where the inputs, or **arguments** go.

```
mean()
```

"Input is this object. Get the mean of it."

```
mean(x = my_numbers)
```

Named argument.
Their names are internal to functions.

Functions

take inputs, perform actions, produce outputs

```
mean(my_numbers)
```

If you just write the name of the input,
R assign it to the function's arguments
in the order given.

You can assign a function's output to a named object

```
my_summary <- summary(my_numbers)
```

```
my_sd <- sd(my_numbers)
```

Objects you create exist until you overwrite or delete them

```
rm(my_numbers)
```

```
my_numbers
```

```
my_numbers <- c(1, 2, 3, 1, 3, 5, 25)
```

Things to try on Objects

```
class(my_numbers)

table(my_numbers)
```

Notice that these are functions

```
x <- c(my_numbers, 5)
y <- c(my_numbers, "hello")
```

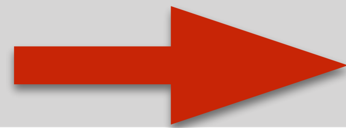
How do x and y differ?

```
mean(c(my_numbers, my_numbers))
```

Functions can be nested, and will be evaluated from the inside out.

The pipe operator %>%

```
mean(my_numbers)
```



```
my_numbers %>% mean()
```

"and then"

```
round(mean(my_numbers))
```

```
my_numbers %>% mean() %>% round()
```

This is very convenient

Objects are of different classes

```
class(my_numbers)
```

Vectors

`numeric`

`character`

`factor`

Arrays

`matrix`

`data.frame`

`tibble`

Models

`lm`

`glm`

4. R will be Frustrating

Syntax

Vocabulary

Concepts

~/>_

LET'S GO ALREADY

MOST DATA ANALYSIS IS CLEANING & RECODING

```r
my_data <- read_csv(file = "data/organdata.csv")
```

Field delimiter is ,

```r
read_csv2(file = "data/my_csv_file.csv")
```

Field delimiter is ;

```r
read_dta(file = "data/my_stata_file.dta")

read_spss(file = "data/my_spss_file.sav")

read_sas(data_file = "<NAME>",
         catalog_file = "<NAME>")

read_table(file = "<NAME>")
```

Structured but not delimited

```r
my_data <- read_csv(file = "data/organdata.csv")
```

Field delimiter is ,

```r
read_csv2(file = "data/my_csv_file.csv")
```

Field delimiter is ;

```r
read_dta(file = "data/my_stata_file.dta")

read_spss(file = "data/my_spss_file.sav")

read_sas(data_file = "<NAME>",
         catalog_file = "<NAME>")

read_table(file = "<NAME>")
```

Structured but not delimited

https://github.com/kjhealy/cq

Congressional Example

dplyr **lets you manipulate tables in a series of steps called a pipeline**

Group the data at the level we want, such as "Religion by Region" or "Authors by Publications by Year".

`group_by()`

Filter or ***Select*** pieces of the data. This gets us the subset of the table we want to work on.

`filter()` rows
`select()` columns

Mutate the data by creating new variables at the *current* level of grouping. Mutating adds new columns to the table.

`mutate()`

Summarize or aggregate the grouped data. This creates new variables at a higher level of grouping. For example we might calculate means with mean() or counts with n(). This results in a smaller, summary table, which we might do more things with if we want.

`summarize()`

Create a **pipeline** of transformations with the **pipe operator**

%>%

```
mean_age <- data %>%
            filter(position == "U.S. Representative") %>%
            group_by(congress) %>%
            summarize(year = first(start_year),
                      mean_age = mean(start_age))

data %>% filter(position == "U.S. Representative",
                party %in% c("Democrat", "Republican")) %>%
   group_by(congress) %>%
   summarize(year = first(start_year), mean_age = mean(start_age)) %>%
   filter(congress == 116)
```

Looking at Congress

The name of the function, and the library it is in.

mean {base} R Documentation

Arithmetic Mean

Description

What it does.

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

The function's name, and in the parentheses the named arguments it expects, in the order it expects them. If an argument has a default value, it is shown. Arguments without default values (e.g. x) must be provided by you.

Arguments

More details on each named argument. This will tell you what class of thing each argument has to be—an object, a number, a data frame, a logical value, etc.

`x` An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

The ellipsis allows other arguments to be passed to and from the function.

Value

What the function returns—i.e., the result of whatever operation or calculation it performs. This can be a single number, as here, or a multi-part object such as a list, a data frame, a plot, or a model.

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Other related functions

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Self-contained examples that you can run at the console. These may use built-in datasets or other R functions.

[Package *base* version 3.4.3 Index]

Visit the package's Index page to look for Demos and Vignettes detailing how it works.