Data Analysis in R

Good Programming Practices

Michael E DeWitt Jr

2018-09-20 (updated: 2018-09-21)

Not Every Lecture Can Be Fun

Not Every Lecture Can Be Fun

But it is essential to learn good habits in the beginning

Not Every Lecture Can Be Fun

But it is essential to learn good habits in the beginning

Because bad habits are hard to break

Good Programming Practices

One day you will hand off your project

Good Programming Practices

One day you will hand off your project

You take a new position

Good Programming Practices

One day you will hand off your project

You take a new position

Get promoted

Good Programming Practices

One day you will hand off your project

You take a new position

Get promoted

The future you who hasn't looked at the project in months (years?)

Good Programming Practices

One day you will hand off your project

You take a new position

Get promoted

The future you who hasn't looked at the project in months (years?)

Good project organisation looks good to an employer

Good Programming Practices

One day you will hand off your project

You take a new position

Get promoted

The future you who hasn't looked at the project in months (years?)

Good project organisation looks good to an employer

You don't want to be on call forever!

Applying good practices will save you time

Applying good practices will save you time

- You don't have to re-invent the wheel each time to reopen your work

Applying good practices will save you time

- You don't have to re-invent the wheel each time to reopen your work
- Providing project structure removes some of the "writer's block"

Applying good practices will save you time

- You don't have to re-invent the wheel each time to reopen your work
- Providing project structure removes some of the "writer's block"

> Jobs decided he liked the idea of having a uniform for himself, “both because of its daily convenience (the rationale he claimed) and its ability to convey a signature style,” the excerpt reads. --PC Magazine

The Other Nice Thing

When somone calls you to ask for help

The Other Nice Thing

When somone calls you to ask for help

You can answer everything on the phone because you (and your team) use the same structure!

The RStudio Project Structure

We are going to utilise the RStudio Project Structure

The RStudio Project Structure

We are going to utilise the RStudio Project Structure

This will provide an *anchor* for our project structure

The RStudio Project Structure

We are going to utilise the RStudio Project Structure

This will provide an *anchor* for our project structure

Start a new *directory*

The RStudio Project Structure

We are going to utilise the RStudio Project Structure

This will provide an *anchor* for our project structure

Start a new *directory*

The project will be completely self contained in this directory

The RStudio Project Structure

We are going to utilise the RStudio Project Structure

This will provide an *anchor* for our project structure

Start a new *directory*

The project will be completely self contained in this directory

Allows anyone to take our project folder and run our *exact* same analysis

.Rproj

.RProj is the heart of the R Studio Project

.Rproj

.RProj is the heart of the R Studio Project

Basically a small text configuration file that specifies

- Project working directory (aka where in the filesystem the project exists)
- Allows relative file paths to be used reliably
- General configurations

.Rproj

.RProj is the heart of the R Studio Project

Basically a small text configuration file that specifies

- Project working directory (aka where in the filesystem the project exists)
- Allows relative file paths to be used reliably
- General configurations

We must turn off saving RData file!!!

Building the scaffolding

Start each project with a basic directory structure within your project

```
##               levelName
## 1 data analysis project
## 2  ¦--data
## 3  ¦--figs
## 4  ¦--munge
## 5  ¦--src
## 6  ¦--output
## 7  °--reports
```

Building the scaffolding

Start each project with a basic directory structure within your project

```
##               levelName
## 1 data analysis project
## 2  ¦--data
## 3  ¦--figs
## 4  ¦--munge
## 5  ¦--src
## 6  ¦--output
## 7  °--reports
```

These represent the *juste necessaire* for each project

Enhanced Directory Structure (recommended)

The enhanced structure adds a few more folders which can be helpful for more advanced projects

```
##               levelName
## 1 data analysis project
## 2  ¦--data
## 3  ¦--munge
## 4  ¦--output
## 5  ¦--src
## 6  ¦--figs
## 7  ¦--reports
## 8  ¦--req***
## 9  °--test***
```

Enhanced Directory Structure (recommended)

The enhanced structure adds a few more folders which can be helpful for more advanced projects

```
##               levelName
## 1 data analysis project
## 2  ¦--data
## 3  ¦--munge
## 4  ¦--output
## 5  ¦--src
## 6  ¦--figs
## 7  ¦--reports
## 8  ¦--req***
## 9  °--test***
```

Adding `req` and `test` will become clear as we explore this expanded structure

Starting with a README

```
##                    levelName
## 1  data analysis project
## 2   ¦--data
## 3   ¦--munge
## 4   ¦--output
## 5   ¦--src
## 6   ¦--figs
## 7   ¦--reports
## 8   ¦--req
## 9   ¦--test
## 10  °--README.md/ README.Rmd
```

Your project directory should start with a README file that indicates a few basic components of your project

Starting with a README

```
##                    levelName
## 1  data analysis project
## 2   ¦--data
## 3   ¦--munge
## 4   ¦--output
## 5   ¦--src
## 6   ¦--figs
## 7   ¦--reports
## 8   ¦--req
## 9   ¦--test
## 10  °--README.md/ README.Rmd
```

Your project directory should start with a README file that indicates a few basic components of your project

- **Project Title**
- **Project Team and Contact Information**
- **Purpose**: What is the goal of the project/ research question

The README

Serves as a top level introduction to the project

The README

Serves as a top level introduction to the project

And at the end, you can put your abstract here!

Repository at time of publication

This repository is constantly being updated in response to feedback and inquiries; however, all code will remain entirely reproducible at any point in the commit history.

For full transparency, we wanted to note what the repository looked like before we made additional changes. Thus, the `paper` release is the version of the repository that existed at the original time of publication. You can get this release by downloading it or using `git checkout paper` in your local repository.

Abstract

Background: Quantifying the effect on society of natural disasters is critical for recovery of public health services and infrastructure. The death toll can be difficult to assess in the aftermath of a major disaster. In September 2017, Hurricane Maria caused massive infrastructural damage to Puerto Rico, but its effect on mortality remains contentious.

Methods: Using a representative, stratified sample, we surveyed 3299 randomly chosen households across Puerto Rico to produce an independent estimate of all-cause mortality after the hurricane. Respondents were asked about displacement, infrastructure loss, and causes of death. We calculated excess deaths by comparing our estimated post-hurricane mortality rate with official rates for the same period in 2016.

Results: From the survey data, we estimated a mortality rate of 14.2 deaths (95% confidence interval [CI], 9.8 to 18.9) per 1000

usethis::use_rmd_readme

RStudio has made initiating a README file easy

usethis::use_rmd_readme

RStudio has made initiating a README file easy

```
install.packages("usethis")
library(usethis)
usethis::use_readme_rmd()
```

usethis::use_rmd_readme

RStudio has made initiating a README file easy

```
install.packages("usethis")
library(usethis)
usethis::use_readme_rmd()
```

Use the markdown mark-up language to format the new document

It All Starts With The Data

Rules of Data

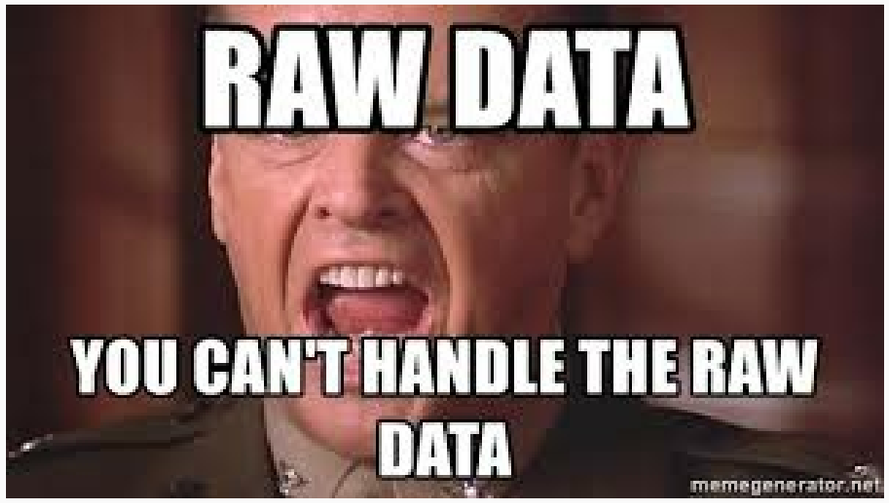
Rule #1 You never alter the raw data

It All Starts With The Data

Rules of Data

Rule #1 You never alter the raw data

Rule #2 You never, ever alter the raw data



data

This is a folder to store whatever raw files you have

Reproducibility starts with data provenance

data

This is a folder to store whatever raw files you have

Reproducibility starts with data provenance

Add a README

Use a README file *within* the data directory to list the facts about your data

- Date of collection
- Who collected it
- Method of acquisition
- Codebook/ Fieldname Descriptions

Munge - Now It's Time to Play

munge - mash until no good

This is the folder to store scripts that take the *raw data* and *clean it*

Munge - Now It's Time to Play

munge - mash until no good

This is the folder to store scripts that take the *raw data* and *clean it*

Then write out the *cleaned data* into the `output` folder for future use

Munge - Now It's Time to Play

munge - mash until no good

This is the folder to store scripts that take the *raw data* and *clean it*

Then write out the *cleaned data* into the `output` folder for future use

As painful as it is, all cleaning should be done programatically

```
clean_df_1 <- df_raw %>%
  mutate(student_id = str_extract_all(id, pattern = "0\\d{7}"))
```

Munge - Now It's Time to Play

munge - mash until no good

This is the folder to store scripts that take the *raw data* and *clean it*

Then write out the *cleaned data* into the `output` folder for future use

As painful as it is, all cleaning should be done programatically

```
clean_df_1 <- df_raw %>%
  mutate(student_id = str_extract_all(id, pattern = "0\\d{7}"))
```

e.g. Your munge code will *always* work if new data overwrote your old data

Outputs

As mentioned, this is the home for any kind of outputed data

- Cleaned data with a descriptive name
- Useful to version using a date (YYYY-MM-DD)

```
write_csv(cleaned_data, paste0("output/", Sys.Date(), "_data_cleaned.csv"))
```

src let's start writing

`src` is the folder to write all of your scripts

src let's start writing

`src` is the folder to write all of your scripts

Start Your Analysis By Running Previous Steps

Typically these scripts will start by running any cleaning scripts with something like

```
lapply(list.files(path = "munge",full.names = T, pattern = ".R"), source)
```

src let's start writing

`src` is the folder to write all of your scripts

Start Your Analysis By Running Previous Steps

Typically these scripts will start by running any cleaning scripts with something like

```
lapply(list.files(path = "munge",full.names = T, pattern = ".R"), source)
```

This command will look for all `.R` files in the `munge` folder and run them

All about reproducibility

Analysis is fun!

Save each analysis as a separate R script

Save a script based on the topic it covers or the question your were exploring

Analysis is fun!

Save each analysis as a separate R script

Save a script based on the topic it covers or the question your were exploring

Good	BAD
EDA.R	test.R
descriptive_stats.R	final.R
linear_regression.R	untitled.R

Parts of a Script

Each script should have the same basic form

Parts of a Script

Each script should have the same basic form

Purpose

The purpose describes what the script is attempting to accomplish/ or question being asked

Comments use the `#` symbol for comments (not read by R)

```
#Purpose: The purpose of this script is to look at a linear relationship between mpg and disp

fit_linear <- lm(mpg ~ disp, data = mtcars)
```

Parts of a Script

Each script should have the same basic form

Purpose

The purpose describes what the script is attempting to accomplish/ or question being asked

Comments use the `#` symbol for comments (not read by R)

```
#Purpose: The purpose of this script is to look at a linear relationship between mpg and disp

fit_linear <- lm(mpg ~ disp, data = mtcars)
```

Libraries

It is useful to outline what libraries are used *at the top* of the script.

```
library(tidyverse)
library(brms)
```

Parts of a Script

Each script should have the same basic form

Section Labels

If you can organise your code into logical sections, then add a section break by pressing

CTRL/CMD + SHIFT + R

```
# Calculating Summaries --------------------------------------------------
```

Parts of a Script

Inline Comments

```
#Generate histogram

ggplot(mtcars, aes(x = wt))+
  geom_histogram()
```

Imagine you are writing for your *future self*

Annotate code based on what you are *trying* to do

General Naming Conventions

Datasets should be named as nouns

General Naming Conventions

Datasets should be named as nouns

Descriptive names are better than non-descriptive names

Object Naming Examples

Good

- student_records
- iris_data
- clean_data_final

Bad

- data
- my_data
- a

Naming Functions

There will come a time when you will need to write your own function

It is advisable to name functions descriptively with **verbs**

```
make_sum <- function( x, y) {
  out <- x + y

  return(out)
}
```

If they start with the same keyword, that can help, too!

Whitespace

Whitespace is your friend

When writing, put spaces after `"(" , "," ` to make the code legible

Indent with **two spaces** when writing a long piece of code or in a block

Whitespace and Punctuation (examples)

Good

```
if(this ==TRUE) {
  median(a, n)
} else{
  mean(a, b, trim = 0.2)
}
```

Bad

```
if(this ==TRUE) {median(a,n)} else{
  mean(a,b,trim=0.2)
}
```

Note that RStudio will correct to this convention if you highlight your code and press

```
CTRL/CMD + SHIFT + A
```

Naming Things

R is **case sensitive**

```
a <- 1
A
```

```
## Error in eval(expr, envir, enclos): object 'A' not found
```

Naming Things

R is **case sensitive**

```
a <- 1
A
```

```
## Error in eval(expr, envir, enclos): object 'A' not found
```

The only rule for names is that an object cannot start with a number

Naming Things

R is **case sensitive**

```
a <- 1
A
```

```
## Error in eval(expr, envir, enclos): object 'A' not found
```

The only rule for names is that an object cannot start with a number

Several conventions exist for naming objects

- snake_case (my prefered)
- camelCase
- dot.case

Naming Things

R is **case sensitive**

```
a <- 1
A
```

```
## Error in eval(expr, envir, enclos): object 'A' not found
```

The only rule for names is that an object cannot start with a number

Several conventions exist for naming objects

- snake_case (my prefered)
- camelCase
- dot.case

Whatever convention you use, stick with it!

Outputing

Generally, you will use your `src` scripts to generate

- figures
- summary tables

Outputing

Generally, you will use your `src` scripts to generate

- figures
- summary tables

Write these out to the `output`

Outputing

Generally, you will use your `src` scripts to generate

- figures
- summary tables

Write these out to the `output`

Treat the output folder as something that *should* be over-written

Reports

Write and maintain your final report in this folder

This should include your `.Rmd` file

Using the project structure you can reference your `figs` and `outputs`

Tests

For more advanced users, it can be beneficial to include a `tests` folder

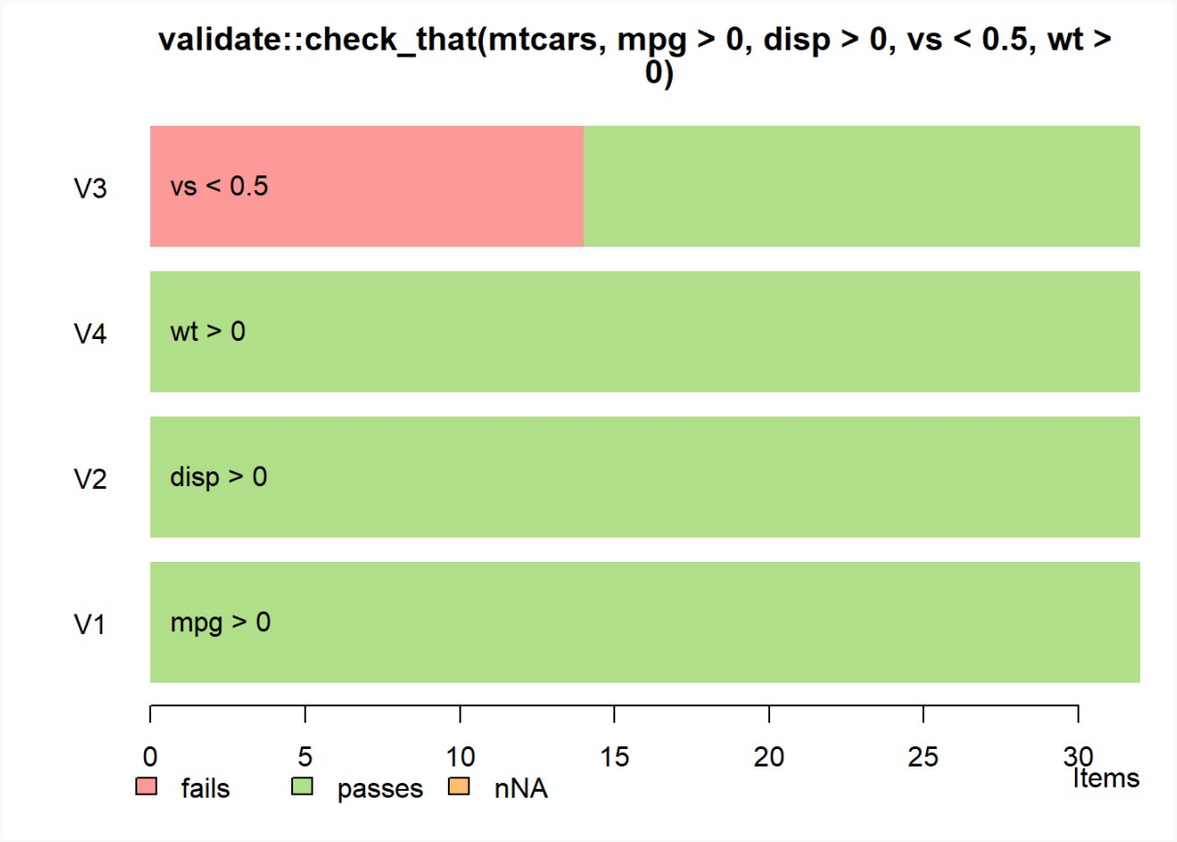
Here unit tests both on your code and your data can be performed

```
library(validate)
validate::check_that(mtcars, mpg >0, disp >0, vs <= 1, wt >0)
```

```
## Object of class 'validation'
## Call:
##     validate::check_that(mtcars, mpg > 0, disp > 0, vs <= 1, wt >
##     0)
##
## Confrontations: 4
## With fails    : 0
## Warnings      : 0
## Errors        : 0
```

Visualising Test Ouputs

```
barplot(validate::check_that(mtcars, mpg >0, disp >0, vs < .5, wt >0))
```



Testing your modeling assumptions

You can use the `testthat` and `validate` functions to test your data and code

```
# Function to check the normality of data using Shapiro-Wilk Test
check_normality <- function(df, alpha = 0.05) {
  p.value <- broom::tidy(shapiro.test(df))$p.value
  if (p.value <= alpha) {
    warning("Imported data is not normal, check QQ Plot")
  }
  else{
    cat("Data passed")
  }
}
```

```
non_normal_data <- rpois(10, 1) # Non-normal data should FAIL
normal_data <- rnorm(100, 54, 1.5) #Normal Data Should PASS

test_that("Function return correct conclusion", {
  expect_that(check_normality(non_normal_data), prints_text("Data passed"))
  expect_output(check_normality(normal_data))
})
```

`req` and your own functions

Eventually you will need to write custom functions

Put these in the `req` folder and you can source them at the beginning of your analysis

Additionally, you can use this folder to list all of the libraries you need

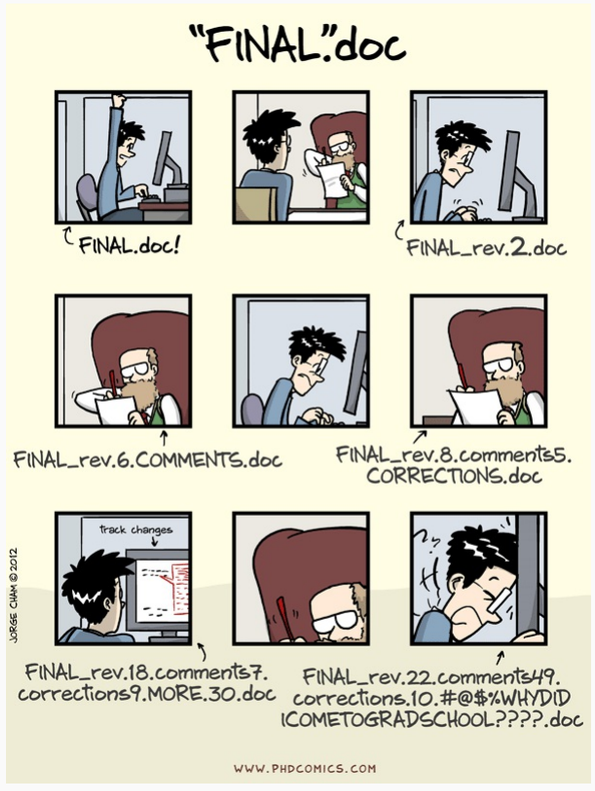
What it looks like

```
# Purpose: Install all required libraries for this project

if(!require("tidyverse", character.only = T)) {
  install.packages("tidyverse")
  library(tidyverse)
} else{
  library(tidyverse)
}

if(!require("AER", character.only = T)) {
  install.packages("AER")
  library(AER)
} else{
  library(AER)
}
```

Our Project Layout

```
##                         levelName
## 1  data analysis project
## 2   ¦--data
## 3   ¦   ¦--raw_data.csv
## 4   ¦   °--README.md/ README.Rmd
## 5   ¦--munge
## 6   ¦   °--import_clean.R
## 7   ¦--output
## 8   ¦   °--cleaned_data.csv
## 9   ¦--src
## 10  ¦   ¦--00_descriptive_stats.R
## 11  ¦   ¦--01_factor_analysis.R
## 12  ¦   °--02_regression_models.R
## 13  ¦--figs
## 14  ¦   ¦--histograms.pdf
## 15  ¦   °--boxplots.pdf
## 16  ¦--reports
## 17  ¦   °--analysis_of_a.Rmd
## 18  ¦--req
## 19  ¦--test
## 20  ¦   °--data_unit_checks.R
## 21  °--README.md/ README.Rmd
```

Now we need to make a change



Git

Git was created by Linus Torvalds (creator of Linux) to help manage the code base of Linux

Git

Git was created by Linus Torvalds (creator of Linux) to help manage the code base of Linux

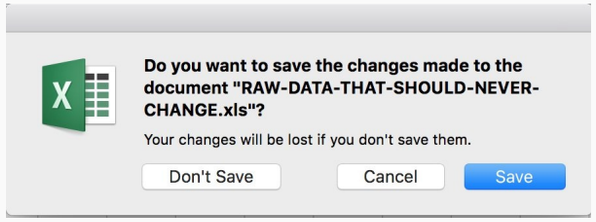
Git is a version control system to track changes in code

Git

Git was created by Linus Torvalds (creator of Linux) to help manage the code base of Linux

Git is a version control system to track changes in code

Provides a history of changes and ability to make deliberate changes to the main code



Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Within your branch you can make changes and the `stage` and `commit` them (make edits) with comments

Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Within your branch you can make changes and the `stage` and `commit` them (make edits) with comments

You can then `push` these changes (stores in memory)

Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Within your branch you can make changes and the `stage` and `commit` them (make edits) with comments

You can then `push` these changes (stores in memory)

If you like the changes you can merge them with the `master` branch

Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Within your branch you can make changes and the `stage` and `commit` them (make edits) with comments

You can then `push` these changes (stores in memory)

If you like the changes you can merge them with the `master` branch

You and your collaborators can then `pull` the new master branch

Git Philosophy

In Git you work on a `branch` of a git repository or *repo* (think version number)

Within your branch you can make changes and the `stage` and `commit` them (make edits) with comments

You can then `push` these changes (stores in memory)

If you like the changes you can merge them with the `master` branch

You and your collaborators can then `pull` the new master branch

If two people try to make changes on the same bit of code you will get a `merge` conflict.

Diff

Version Control = Traceability of Changes

Visibility of differences

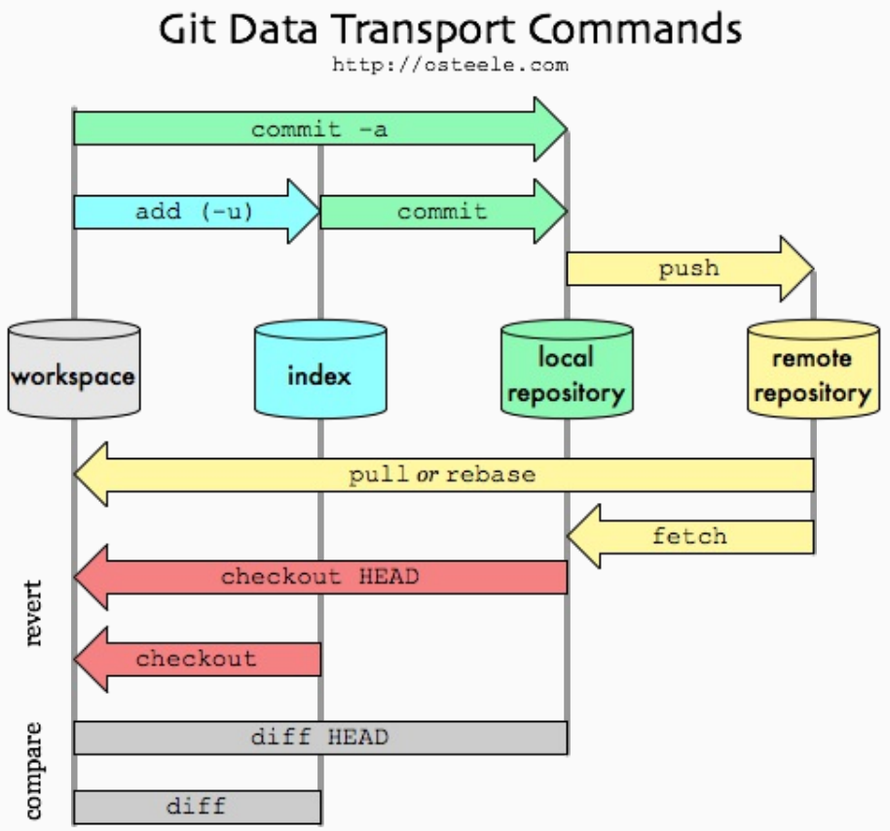
Diff

Version Control = Traceability of Changes

Visibility of differences

```
@@ -3,9 +3,9 @@
 include_once( get_template_directory() . '/lib/init.php' );

 //* Child theme (do not remove)
-define( 'CHILD_THEME_NAME', 'Genesis Sample Theme' );
-define( 'CHILD_THEME_URL', 'http://www.studiopress.com/' );
-define( 'CHILD_THEME_VERSION', '2.1.2' );
+define( 'CHILD_THEME_NAME', 'WPSmith Theme' );
+define( 'CHILD_THEME_URL', 'http://wpsmith.net' );
+define( 'CHILD_THEME_VERSION', '1.0.0' );

 //* Enqueue Google Fonts
 add_action( 'wp_enqueue_scripts', 'genesis_sample_google_fonts' );
```

Git in Pictures



And then there was GitHub

GitHub emerged as an online platform to manage git repositories

And then there was GitHub

GitHub emerged as an online platform to manage git repositories

Hosts your git repository online

And then there was GitHub

GitHub emerged as an online platform to manage git repositories

Hosts your git repository online

Github also provides some handy user interfaces with:

- Issue tracking
- Repo management (merges, etc)
- Sharing repositories
- web hosting

Some Git Commands

starting a repository

```
git init
```

check changes

```
git status
```

stage files

```
git add <file_name>
```

More commands

add a commit message

```
git commit -m "initial commit"
```

push it to the branch

```
git push origin master
```

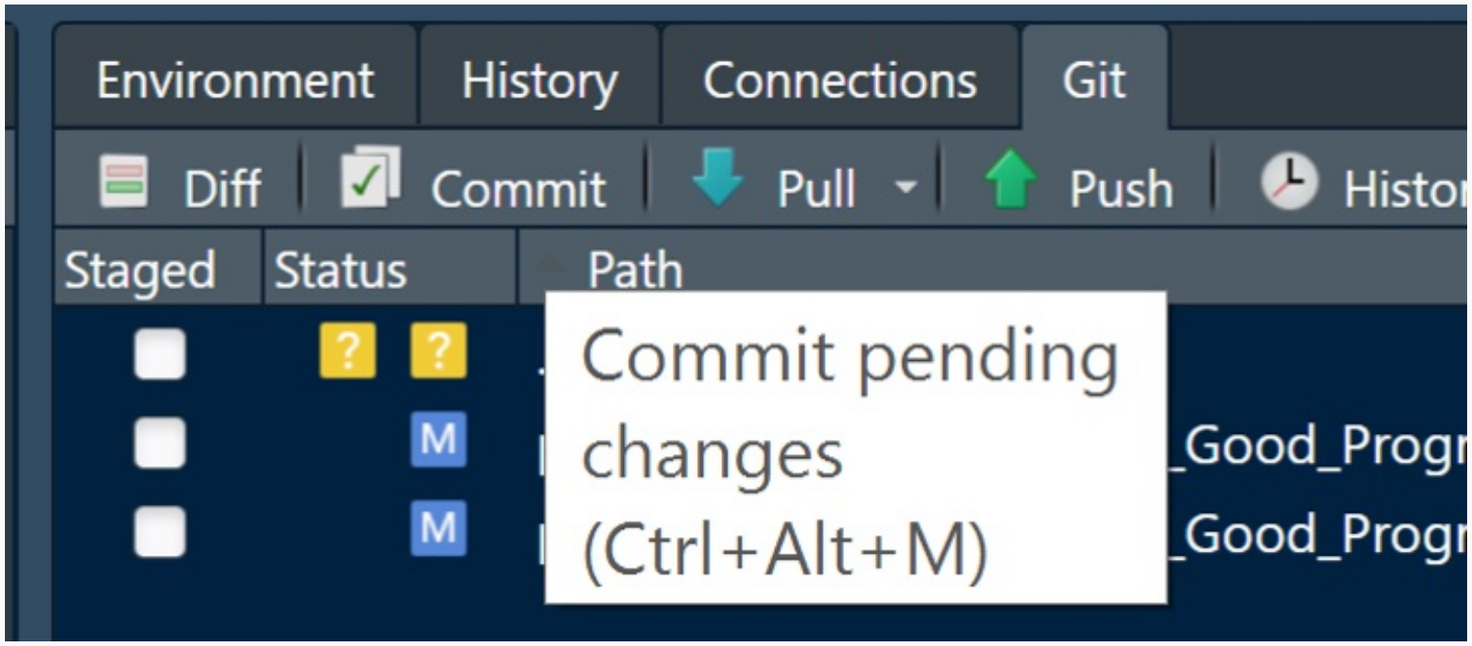
pull the master branch

```
git pull master origin
```

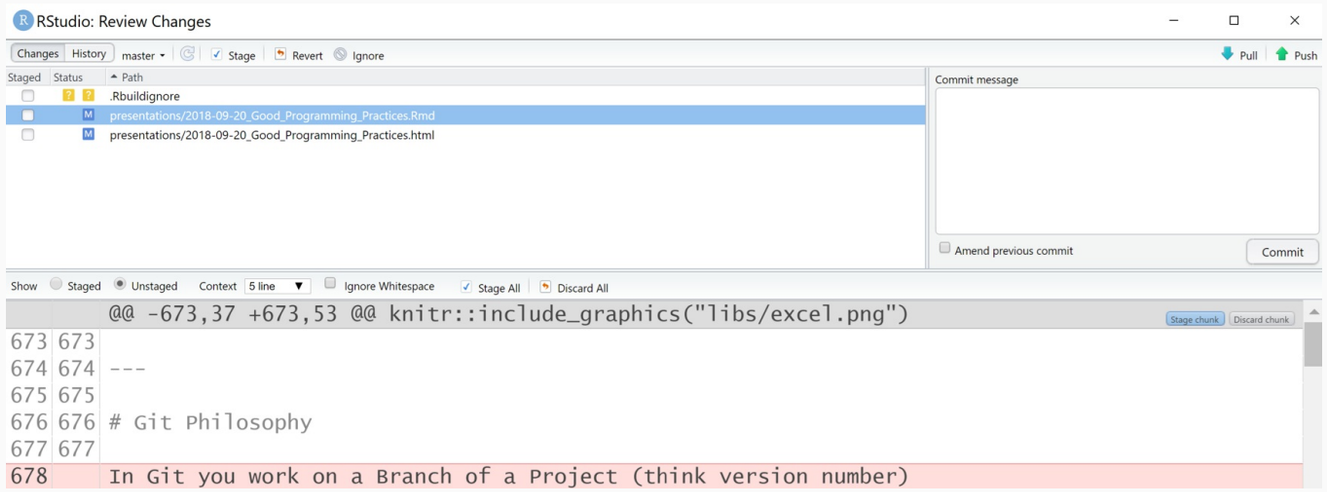
undo to previous commit

```
git revert HEAD
```

Or just use the R gui



The structure is the same as the CL



Recap

Adopt a File System

- Use a standard folder system
- Never alter the raw data

Think Programatically

- Comments and Section Breaks
- README Files
- Design for reproducibility

Version Control

- Use it

It's all for future you!

Now lets play!

https://github.com/medewitt/r_class_demo

Initial Git Configuration

If you haven't done this already we can link your local git to your github

```
git config --global user.name 'Michael DeWitt'
git config --global user.email 'dewittme@wfu.edu'
```

Now we can verify that everything is configured

```
git config --global --list
```

More instructions are located here