

# Version Control with Git and GitHub

## Winter Institute in Data Science

Ryan T. Moore

2024-01-04

Introducing Git + GitHub

Workflow and Git Commands

Branches

Merging and Rebasing

Pull Requests and Forks

# Introducing Git + GitHub

*“Git is a free and open source distributed version control system”*

– *<https://git-scm.com/>*

*“Git is a free and open source distributed version control system”*

*– <https://git-scm.com/>*

- ▶ Set of command-line tools for *version control*: explicit management of file history

*“Git is a free and open source distributed version control system”*

*– <https://git-scm.com/>*

- ▶ Set of command-line tools for *version control*: explicit management of file history
- ▶ *Distributed*: full codebase (current and history) lives on every developer's machine

*“Git is a free and open source distributed version control system”*

*– <https://git-scm.com/>*

- ▶ Set of command-line tools for *version control*: explicit management of file history
- ▶ *Distributed*: full codebase (current and history) lives on every developer's machine
- ▶ Originally written by Linus Torvalds (Linux)

# GitHub

- ▶ A web-based repository for code



# GitHub

- ▶ A web-based repository for code
- ▶ Utilizes `git` version control system for careful tracking of how files change

# GitHub

- ▶ A web-based repository for code
- ▶ Utilizes `git` version control system for careful tracking of how files change
- ▶ Facilitates collaboration by organizing simultaneous editing, issue tracking, merging, conflict management, ...

# GitHub

- ▶ A web-based repository for code
- ▶ Utilizes `git` version control system for careful tracking of how files change
- ▶ Facilitates collaboration by organizing simultaneous editing, issue tracking, merging, conflict management, ...
- ▶ (“Collaboration” includes your future self)

# GitHub

- ▶ A web-based repository for code
- ▶ Utilizes `git` version control system for careful tracking of how files change
- ▶ Facilitates collaboration by organizing simultaneous editing, issue tracking, merging, conflict management, ...
- ▶ (“Collaboration” includes your future self)
- ▶ Think Dropbox/GDrive, but better, more deliberate.

# GitHub

- ▶ A web-based repository for code
- ▶ Utilizes `git` version control system for careful tracking of how files change
- ▶ Facilitates collaboration by organizing simultaneous editing, issue tracking, merging, conflict management, ...
- ▶ (“Collaboration” includes your future self)
- ▶ Think Dropbox/GDrive, but better, more deliberate.
- ▶ Next steps: `renv`, Containers, Docker, Code Ocean

# Examples

**ryantmoore** / **r-data-science** Private

Unwatch ▾ 1

★ Star 0

🍴 Fork

<> Code

🔔 Issues 5

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

Introductory R for Data Science

Manage topics

📶 267 commits

🌿 1 branch

📦 0 releases

👥 3 contributors

📄 GPL-3.0

Branch: master ▾

New pull request

Create new file

Upload files

Find file


Clone or download



**ryantmoore** Update PS6







Latest commit fd1ff6d 6 days ago

📁 admin	Update PS6	6 days ago
📁 code	Update pkg tests and building	6 days ago
📁 data	Make laws data longer	last year
📁 notes	Update pkg tests and building	6 days ago
📁 ps_labs	ps05 Exam class	18 days ago
📁 quiz	Initialize quiz pkg2	6 days ago
📄 .gitignore	Create full gitignore	9 months ago
📄 LICENSE	Initial commit	2 years ago
📄 README.md	Fix typo	13 days ago
📄 r-data-science.Rproj	Add Rproj file	8 months ago


# Examples

 **ryantmoore** / **blockTools** Private












 Unwatch ▾ 10  ★ S

 Code  Issues 16  Pull requests 1  Projects 0  Insights  Settings

Branch: master ▾ **blockTools** / **blockTools** / Create new file Upload file

 **ryantmoore** Add tarball 0.6-2. Update all /blockTools/ files. Latest commit

..

 <b>R</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>data</b>	Initial
 <b>demo</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>inst</b>	Copying directory blockTools/ from devel to master
 <b>man</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>src</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>CHANGELOG</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>COPYING</b>	Initial
 <b>DESCRIPTION</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>LICENSE</b>	Add tarball 0.6-2. Update all /blockTools/ files.
 <b>NAMESPACE</b>	Add tarball 0.6-2. Update all /blockTools/ files.

15 / 159

# Examples

🔒 [ryantmoore / blockTools](#) Private

👁 Unwatch ▼ 10 ⭐ Star 0 🍴 Fork

<> Code ⓘ Issues 16 📄 Pull requests 1 📁 Projects 0 📊 Insights ⚙ Settings

## Optimal greedy randomly breaks ties. Can we set a seed to get same blocks? #57

[Edit](#)[New](#)

🔔 **Open** ryantmoore opened this issue on Jun 14, 2017 · 3 comments



ryantmoore commented on Jun 14, 2017

+ 🗨 🖋

In `block()`, the optimal-greedy algorithm breaks ties randomly to create blocks. Can we set a seed so that we can replicate the blocks when a lot of ties exist? @keithschnak will this involve passing a new argument to `optgreedy()` in `block()` to influence the underlying C code?



👤 ryantmoore added **enhancement** **question** labels on Jun 14, 2017



👤 ryantmoore assigned **keithschnak** on Jun 14, 2017



📧 keithschnak commented on Jun 14, 2017

+ 🗨 🖋 ✕

I think I can make it pass the seed from R as an argument to the C function. I'll take a look this evening.

...

### Assignees



keithschnak

### Labels

**enhancement**

**question**

### Projects

No one yet

### Milestone

No milestone

### Notifications

16 / 16



## The Motivation

- ▶ Web resources: page, README, issue tracking and assignment

## The Motivation

- ▶ Web resources: page, README, issue tracking and assignment
- ▶ Work simultaneously, large group, without fear of conflicts

## The Motivation

- ▶ Web resources: page, README, issue tracking and assignment
- ▶ Work simultaneously, large group, without fear of conflicts
- ▶ Package sharing w/o CRAN or `tar.gz`

## The Motivation

- ▶ Web resources: page, `README`, issue tracking and assignment
- ▶ Work simultaneously, large group, without fear of conflicts
- ▶ Package sharing w/o CRAN or `tar.gz`
- ▶ Gists: small stand-alone functions/code (e.g., <http://j.mp/2djpFON>)

## The Motivation

- ▶ Web resources: page, README, issue tracking and assignment
- ▶ Work simultaneously, large group, without fear of conflicts
- ▶ Package sharing w/o CRAN or `tar.gz`
- ▶ Gists: small stand-alone functions/code (e.g., <http://j.mp/2djpFON>)
- ▶ Data science jobs: provide GitHub ID

# Alternatives

Git:

- ▶ Mercurial
- ▶ Concurrent Versions System (CVS)
- ▶ Subversion (SVN)
- ▶ ...

# Alternatives

Git:

- ▶ Mercurial
- ▶ Concurrent Versions System (CVS)
- ▶ Subversion (SVN)
- ▶ ...

GitHub:

- ▶ Bitbucket
- ▶ GitLab
- ▶ GitKraken
- ▶ SourceForge
- ▶ ...

## Workflow and Git Commands



# Git + GitHub

The General Workflow:

- ▶ Create/designate/find *repository* to track

# Git + GitHub

The General Workflow:

- ▶ Create/designate/find *repository* to track
- ▶ Make changes to code

## The General Workflow:

- ▶ Create/designate/find *repository* to track
- ▶ Make changes to code
- ▶ *Commit* changes: declare “save this snapshot”

# Git + GitHub

## The General Workflow:

- ▶ Create/designate/find *repository* to track
- ▶ Make changes to code
- ▶ *Commit* changes: declare “save this snapshot”
- ▶ Send commits to GitHub (*push*)

# Workflow and Product

- ▶ Include elements of the work product

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata



# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)
- ▶ Exclude elements of the workflow

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)
- ▶ Exclude elements of the workflow
  - ▶ Directory setting

# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)
- ▶ Exclude elements of the workflow
  - ▶ Directory setting
  - ▶ Package installation

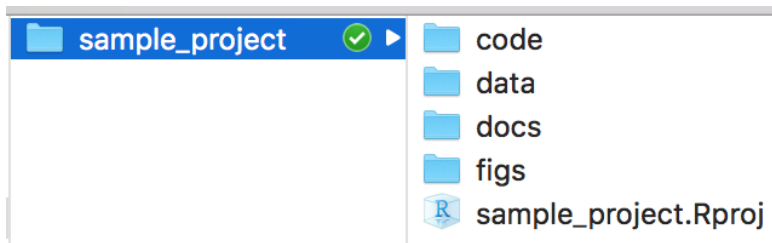
# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)
- ▶ Exclude elements of the workflow
  - ▶ Directory setting
  - ▶ Package installation
- ▶ Exclude sensitive files

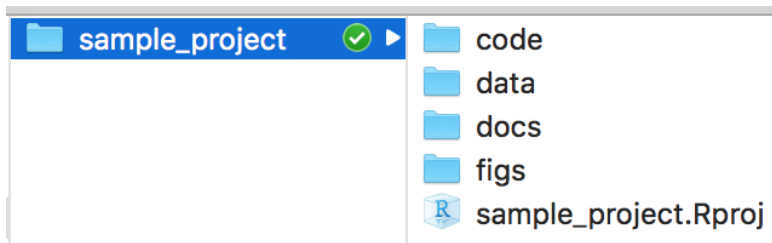
# Workflow and Product

- ▶ Include elements of the work product
  - ▶ Code files
  - ▶ Documentation (how to use your code)
  - ▶ Metadata
  - ▶ Compiled files? (`.pdf`, `.tar.gz`, ...)
- ▶ Exclude elements of the workflow
  - ▶ Directory setting
  - ▶ Package installation
- ▶ Exclude sensitive files
  - ▶ Seriously. This is hard to undo.

# Work Product: Directory Structure



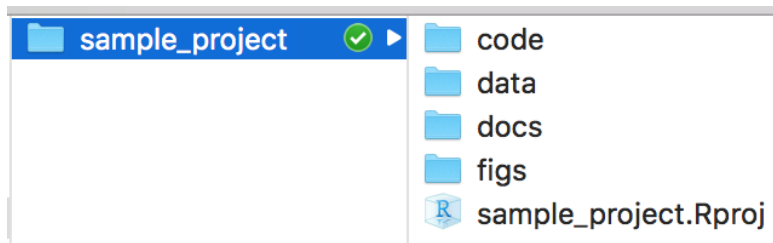
# Work Product: Directory Structure



- ▶ Set up on your machine

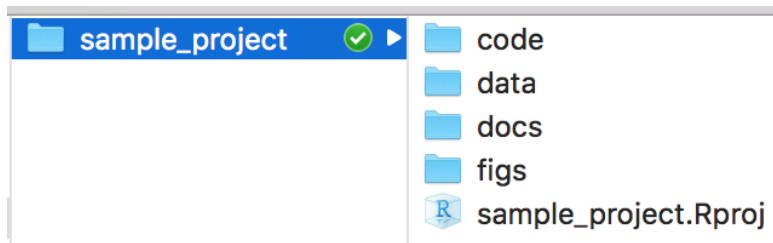


# Work Product: Directory Structure



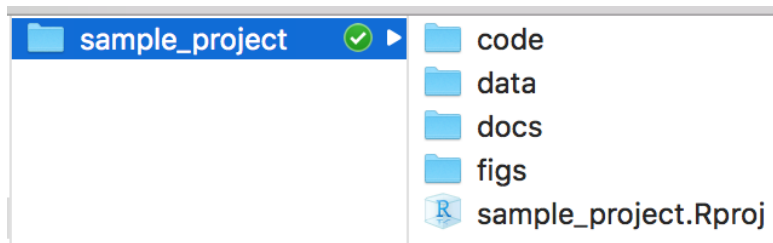
- ▶ Set up on your machine
- ▶ But `git` won't track empty directories

## Work Product: Directory Structure



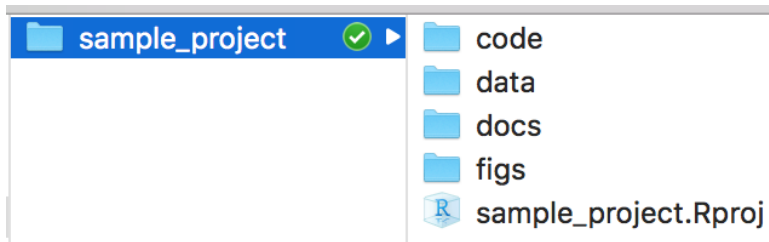
- ▶ Set up on your machine
- ▶ But `git` won't track empty directories
- ▶ Add, track an empty file

# Work Product: Directory Structure

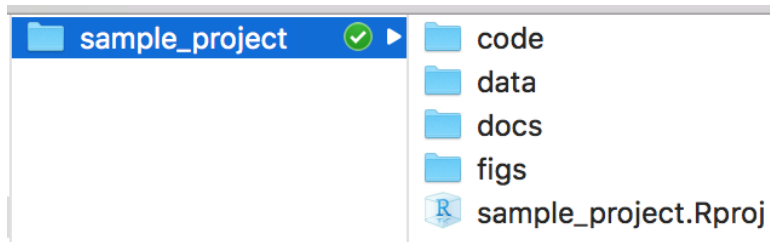


- ▶ Set up on your machine
- ▶ But `git` won't track empty directories
- ▶ Add, track an empty file
  - ▶ (How I make `ps` directories)

## Work Product: Sensitive Data

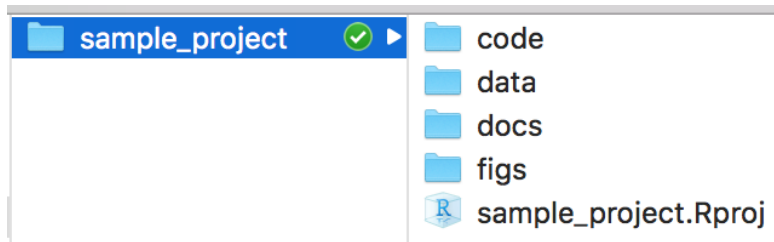


## Work Product: Sensitive Data



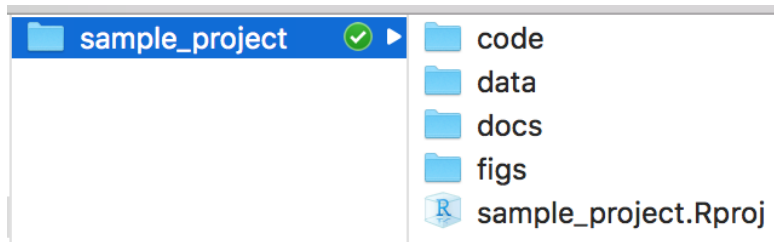
- ▶ Store sensitive data in local `sample_project/data/`

## Work Product: Sensitive Data



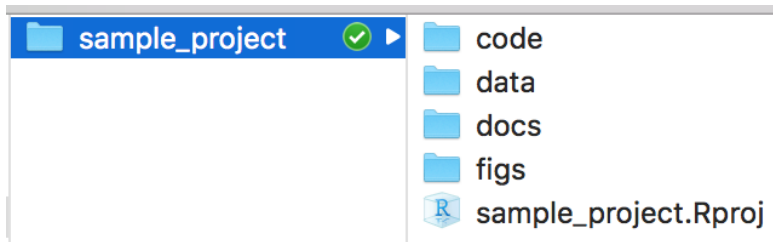
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it

## Work Product: Sensitive Data



- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo

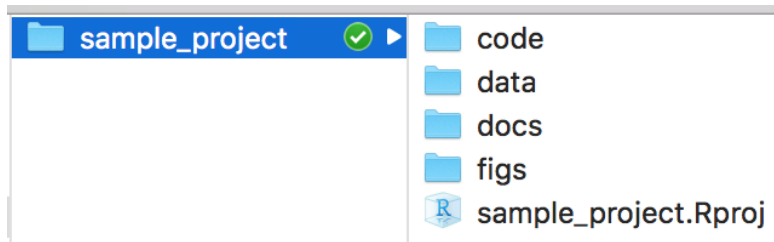
## Work Product: Sensitive Data



- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)

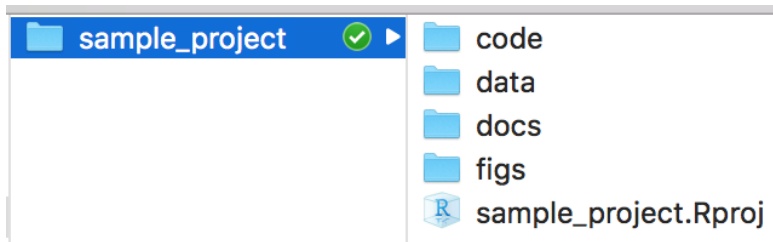


## Work Product: Sensitive Data



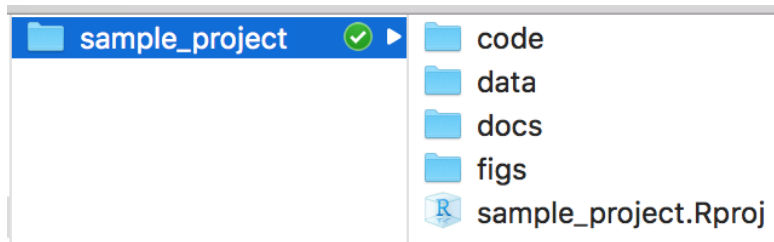
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.

## Work Product: Sensitive Data



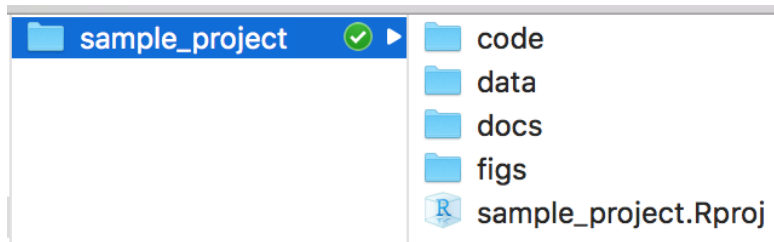
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)

## Work Product: Sensitive Data



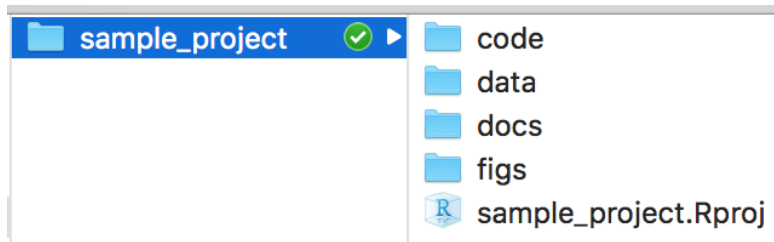
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)
  - ▶ First add to `.gitignore` (avoid future problem)

## Work Product: Sensitive Data



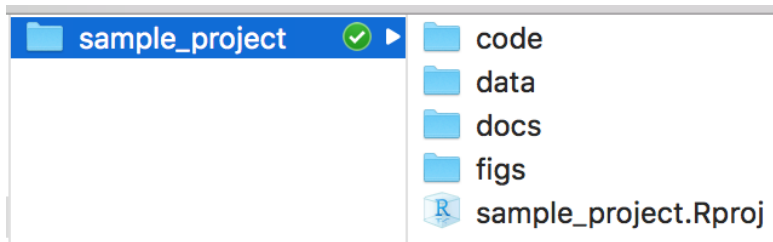
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)
  - ▶ First add to `.gitignore` (avoid future problem)
  - ▶ Then remove sensitive file from history's dirty commits

## Work Product: Sensitive Data



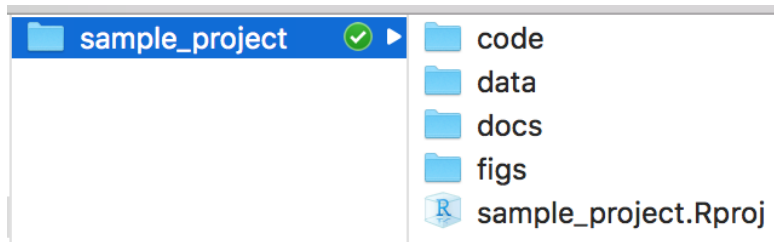
- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)
  - ▶ First add to `.gitignore` (avoid future problem)
  - ▶ Then remove sensitive file from history's dirty commits
    - ▶ `git-filter-branch`

## Work Product: Sensitive Data



- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)
  - ▶ First add to `.gitignore` (avoid future problem)
  - ▶ Then remove sensitive file from history's dirty commits
    - ▶ `git-filter-branch`
    - ▶ (Or `bfg` from BFG Repo Cleaner)

## Work Product: Sensitive Data



- ▶ Store sensitive data in local `sample_project/data/`
- ▶ But do not `git` track it
- ▶ Assume everything in public, even if “private” repo
  - ▶ federal laws (HIPAA, FERPA, etc.)
  - ▶ agreements with clients, employers, funders, partners, etc.
- ▶ Difficult to remove (the point of version control...)
  - ▶ First add to `.gitignore` (avoid future problem)
  - ▶ Then remove sensitive file from history's dirty commits
    - ▶ `git-filter-branch`
    - ▶ (Or `bfg` from BFG Repo Cleaner)
  - ▶ Repeat for every branch

## Work Product and .gitignore

To **not track**, list in .gitignore file.

You can gitignore

- ▶ a specific file



## Work Product and .gitignore

To **not track**, list in .gitignore file.

You can gitignore

- ▶ a specific file
- ▶ an entire file type

## Work Product and .gitignore

To **not track**, list in .gitignore file.

You can gitignore

- ▶ a specific file
- ▶ an entire file type
- ▶ an entire directory

## Work Product and .gitignore

To **not track**, list in .gitignore file.

You can gitignore

- ▶ a specific file
- ▶ an entire file type
- ▶ an entire directory
- ▶ a file type within a directory

## Work Product and `.gitignore`

To **not track**, list in `.gitignore` file.

You can `gitignore`

- ▶ a specific file
- ▶ an entire file type
- ▶ an entire directory
- ▶ a file type within a directory
- ▶ ...

## Work Product and .gitignore

My .gitignore workflow:

- ▶ Initialize repo w/ .gitignore, choose R
- ▶ Immediately update via  
<https://www.gitignore.io>

## Work Product and .gitignore

My .gitignore workflow:

- ▶ Initialize repo w/ .gitignore, choose R
- ▶ Immediately update via <https://www.gitignore.io>
- ▶ R
- ▶ L<sup>A</sup>T<sub>E</sub>X
- ▶ T<sub>E</sub>X
- ▶ Python
- ▶ Data files, directories
- ▶ ...

How should I `git`?

There are many ways to `git`.

## How should I `git`?

There are many ways to `git`.

Be familiar w/ command line `git`, even if you GUI.



## How should I `git`?

There are many ways to `git`.

Be familiar w/ command line `git`, even if you GUI.

Each GUI defines its own “sync”, but `git` is more specific.

## How should I `git`?

There are many ways to `git`.

Be familiar w/ command line `git`, even if you GUI.

Each GUI defines its own “sync”, but `git` is more specific.

If “sync” fails, was it `push`, `fetch`, `pull`, `merge`, ...?

# How should I `git`?

There are many ways to `git`.

Be familiar w/ command line `git`, even if you GUI.

Each GUI defines its own “sync”, but `git` is more specific.

If “sync” fails, was it `push`, `fetch`, `pull`, `merge`, ...?

- ▶ GitHub's GUI
- ▶ GitKraken
- ▶ Tower
- ▶ RStudio
- ▶ ...

## Some Command Line basics

Where to find the command line?

- ▶ Stand-alone programs:
  - ▶ MacOS **iTerm2**, Terminal ...
  - ▶ Windows **Cmder**, Git BASH, PowerShell
- ▶ RStudio Terminal
  - ▶ (next to Console)
  - ▶ (why not? Workflow.)
  - ▶ (Multiple windows, Cmd-tab, file mngmnt w/o RStudio)

## Some Command Line basics

- ▶ `ls`: list files/dirs
- ▶ `pwd`: print working dir
- ▶ `mkdir subdir`: make new subdir
- ▶ `cd subdir`: change working dir (to `subdir`)
- ▶ `cd ..`: change working dir (to one above)
- ▶ `cp file.R file_copy.R`: copy file
- ▶ `mv file.R subdir/file.R`: move file
- ▶ `rm file.R`: delete file
- ▶ `touch file.R`: create new file
- ▶ `open file.R`: open extant file  
(Win: `file.R` + Enter)
- ▶ `cat file.R`: print contents of file
- ▶ `man ls`: help file for `ls` (e.g.)

## Let's Practice

Using only the command line,

1. Navigate to your Desktop
2. Make a directory called `cl_dir`
3. Navigate to `cl_dir`
4. Create an empty file here called `empty.txt`
5. Open `empty.txt`
6. Add a line of text; save the file
7. Change the filename to `notempty.txt`
8. Navigate up to the Desktop
9. Print contents of `notempty.txt`
10. List the files in `Desktop/cl_dir`
11. Delete `notempty.txt`

## Some Command Line basics

This is how I navigate files/directories.

## Some Command Line basics

This is how I navigate files/directories.

Git uses similar commands, prefaced with `git`.



## Some Command Line intermediates

- ▶ `ps -u <username>`: view running processes
- ▶ `top`: view CPU hogs
- ▶ `kill <pid>`: kill process (given ID)
- ▶ `mail`
- ▶ `cal`

Some help

GitHub's Git Cheat Sheet:  
<http://j.mp/2Y5HklD>

## Creating a new repository

- ▶ On GitHub.com:  
Profile > Repositories > New
- ▶ Name (`mytest`)
- ▶ Description (brief descr)
- ▶ README (yes, initialize it)
- ▶ `.gitignore`  
(yes, choose R, then [www.gitignore.io](http://www.gitignore.io))
- ▶ license (yes, select one)

# Contributing to a repository

On web directly:

- ▶ Click on **README**, pencil icon. Edit the `.md` file.

# Contributing to a repository

On web directly:

- ▶ Click on **README**, pencil icon. Edit the `.md` file.
- ▶ Preview changes

# Contributing to a repository

On web directly:

- ▶ Click on **README**, pencil icon. Edit the `.md` file.
- ▶ Preview changes
- ▶ Commit

# Contributing to a repository

On web directly:

- ▶ Click on **README**, pencil icon. Edit the `.md` file.
- ▶ Preview changes
- ▶ Commit

## Contributing to a repository

On web directly:

- ▶ Click on `README`, pencil icon. Edit the `.md` file.
- ▶ Preview changes
- ▶ Commit

`README.md` is “GitHub-flavored markdown”

Like `.qmd`, but not identical.



# Contributing to a repository

On web directly:

- ▶ Update `.gitignore`: Don't ignore `.Rproj` files

# Contributing to a repository

On web directly:

- ▶ Update `.gitignore`: Don't ignore `.Rproj` files
- ▶ Edit file, Preview changes

# Contributing to a repository

On web directly:

- ▶ Update `.gitignore`: Don't ignore `.Rproj` files
- ▶ Edit file, Preview changes
- ▶ Commit

# Contributing to a repository

On web directly:

- ▶ Upload files
- ▶ Commit

# Contributing to a repository

Note: each commit is *complete* and *minimal*.

- ▶ Solve a problem, make an addition
- ▶ Addresses a **single** issue

# Contributing to a repository

Note: each commit is *complete* and *minimal*.

- ▶ Solve a problem, make an addition
- ▶ Addresses a **single** issue

Different problem? Different commit.

# Contributing to a repository

Using local version:

- ▶ Clone repo

# Contributing to a repository

Using local version:

- ▶ Clone repo
- ▶ Edit files directly



# Contributing to a repository

Using local version:

- ▶ Clone repo
- ▶ Edit files directly
- ▶ Send changes to GitHub

# Contributing to a repository

Using local version:

- ▶ Clone repo
- ▶ Edit files directly
- ▶ Send changes to GitHub

# Contributing to a repository

Using local version:

- ▶ Clone repo
- ▶ Edit files directly
- ▶ Send changes to GitHub

```
git status
```

```
git add
```

```
git commit -m "Commit Msg"
```

```
git push
```

## Contributing to a repository

Using local version:

- ▶ Clone repo
- ▶ Edit files directly
- ▶ Send changes to GitHub

```
git status
```

```
git add
```

```
git commit -m "Commit Msg"
```

```
git push
```

Workflow: commit, commit, commit, ..., push

In Case of Emergency

# In Case of Emergency



## Cloning extant repository

```
git clone git@github.com:<username>/<reponame>.git
```

# Workflow Commands

```
git status
```



# Workflow Commands

```
git status
```

Neurotically.

## Workflow Commands

```
git status
```

Neurotically.

```
git status
```

 will suggest what to do next.

## Workflow Commands

When I start,

```
git fetch
```

to bring pushed changes to my local version.

## Workflow Commands

When I start,

```
git fetch
```

to bring pushed changes to my local version.

If needed,

```
git pull
```

to merge version on GitHub into mine.

# Workflow Commands

Make changes.

## Workflow Commands

Make changes. Then git:

```
git add <file>
```

```
git commit -m "Commit msg"
```

```
git push
```

## Clone an extant repository

At terminal prompt, `pwd` and `cd` to a dir  
(Desktop, e.g.).

## Clone an extant repository

At terminal prompt, `pwd` and `cd` to a dir (Desktop, e.g.).

Then,

```
git clone git@github.com:<yourusername>/mytest.git
```

and `/mytest/` will appear in the dir.



## Clone an extant repository

At terminal prompt, `pwd` and `cd` to a dir (Desktop, e.g.).

Then,

```
git clone git@github.com:<yourusername>/mytest.git
```

and `/mytest/` will appear in the dir.

Now, edit `README` a bit.

## Clone an extant repository

At terminal prompt, `pwd` and `cd` to a dir (Desktop, e.g.).

Then,

```
git clone git@github.com:<yourusername>/mytest.git
```

and `/mytest/` will appear in the dir.

Now, edit `README` a bit.

Then, at terminal

```
git status
```

```
git commit -m "Commit Msg"
```

```
git push
```

## Delete the local version

- ▶ Delete the local folders
- ▶ (Note: no `git` here, so truth unaffected.)
- ▶ Reclone

## Remove a file from future commits

► `git rm ps06/rtm.R`

## Remove a file from future commits

▶ `git rm ps06/rtm.R`

(Repeat: *future* commits)

# Branches

## Branches

During Git, you are always on *branch* of codebase.

## Branches

During Git, you are always on *branch* of codebase.

By default, create and are on the **main** branch.



## Branches

During Git, you are always on *branch* of codebase.

By default, create and are on the **main** branch.

Create other branches to make changes, commit them, etc., **without** touching the **main** branch.

## Branches

During Git, you are always on *branch* of codebase.

By default, create and are on the `main` branch.

Create other branches to make changes, commit them, etc., **without** touching the `main` branch.

Then, recombine work on the branch back into `main` branch.

## Branches

During Git, you are always on *branch* of codebase.

By default, create and are on the `main` branch.

Create other branches to make changes, commit them, etc., **without** touching the `main` branch.

Then, recombine work on the branch back into `main` branch.

Goal: `main` always works.

# Branching Workflow

- ▶ Create branch

# Branching Workflow

- ▶ Create branch
- ▶ Move to that branch

# Branching Workflow

- ▶ Create branch
- ▶ Move to that branch
- ▶ Make edits to code

# Branching Workflow

- ▶ Create branch
- ▶ Move to that branch
- ▶ Make edits to code
- ▶ Commit and push

# Branching Workflow

- ▶ Create branch
- ▶ Move to that branch
- ▶ Make edits to code
- ▶ Commit and push
- ▶ Issue pull request at [GitHub.com](https://github.com)



# Branching Workflow

- ▶ Create branch
- ▶ Move to that branch
- ▶ Make edits to code
- ▶ Commit and push
- ▶ Issue pull request at [GitHub.com](https://github.com)
- ▶ Someone reviews pull request, merges your branch in, deletes it

# Branching Workflow

▶ `git branch bugFix`

# Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`

# Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`
- ▶ Make edits to code

## Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`
- ▶ Make edits to code
- ▶ `git add`, `git commit`, `git push`

## Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`
- ▶ Make edits to code
- ▶ `git add`, `git commit`, `git push`
- ▶ (`git status` keeps me on track)

## Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`
- ▶ Make edits to code
- ▶ `git add`, `git commit`, `git push`
- ▶ (`git status` keeps me on track)
- ▶ `git checkout main` to return

## Branching Workflow

- ▶ `git branch bugFix`
- ▶ `git checkout bugFix`
- ▶ Make edits to code
- ▶ `git add`, `git commit`, `git push`
- ▶ (`git status` keeps me on track)
- ▶ `git checkout main` to return
- ▶ Eventually, `git merge bugFix`



# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

- ▶ a *remote*: non-local version of repo

# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

- ▶ a *remote*: non-local version of repo
- ▶ **origin**: standard name of your GitHub remote

# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

- ▶ a *remote*: non-local version of repo
- ▶ **origin**: standard name of your GitHub remote
- ▶ **upstream**: source of your clone (usually **origin**)

# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

- ▶ a *remote*: non-local version of repo
- ▶ **origin**: standard name of your GitHub remote
- ▶ **upstream**: source of your clone (usually **origin**)
- ▶ **main**: standard name of main branch

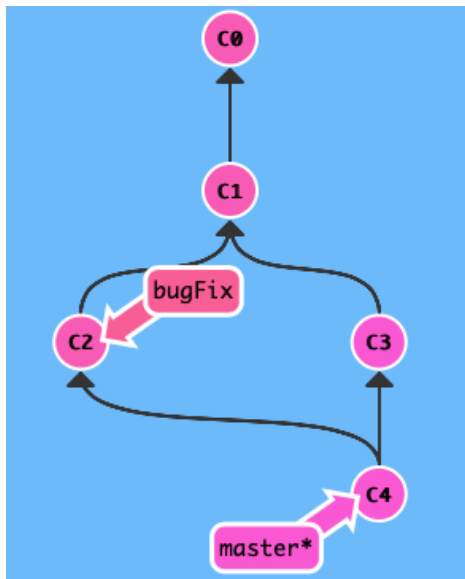
# Terminology for Branches, Forks, Commits

Recall: *distributed* version control.

- ▶ a *remote*: non-local version of repo
- ▶ **origin**: standard name of your GitHub remote
- ▶ **upstream**: source of your clone (usually **origin**)
- ▶ **main**: standard name of main branch
- ▶ **HEAD**: most recent commit on **main** branch

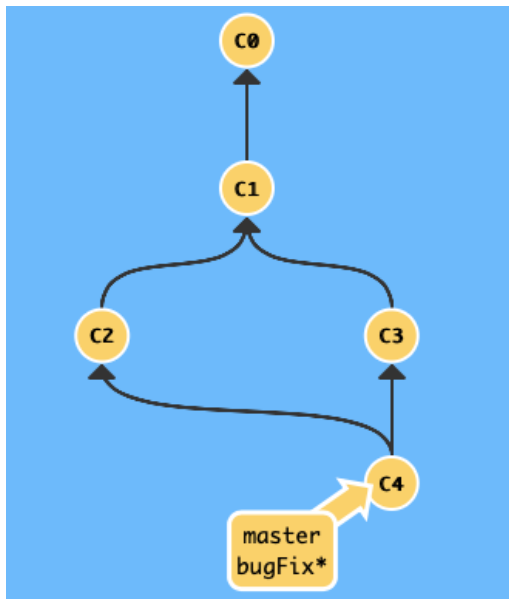
# Merging and Rebasing

## Merging





# Merging



# Rebasing

Rebasing: another way to combine **main** and **subbranch**.

Rebase creates a linear (unbranched) history of commits.

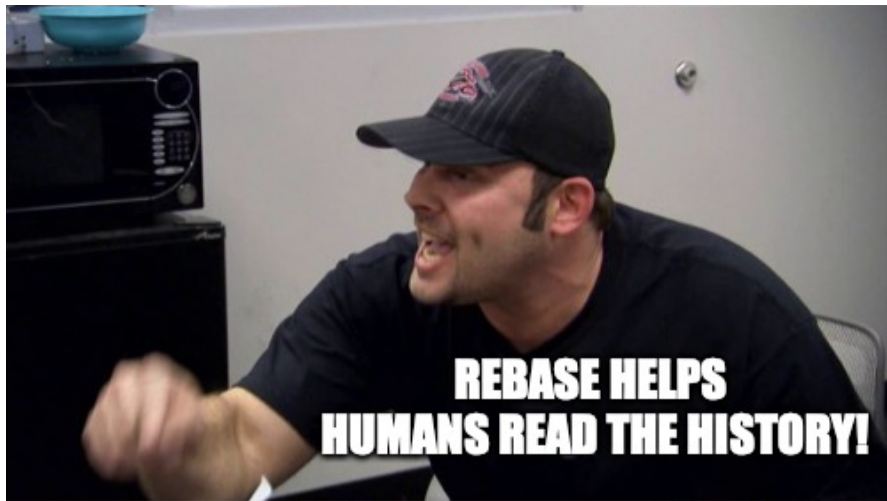
# Rebasing

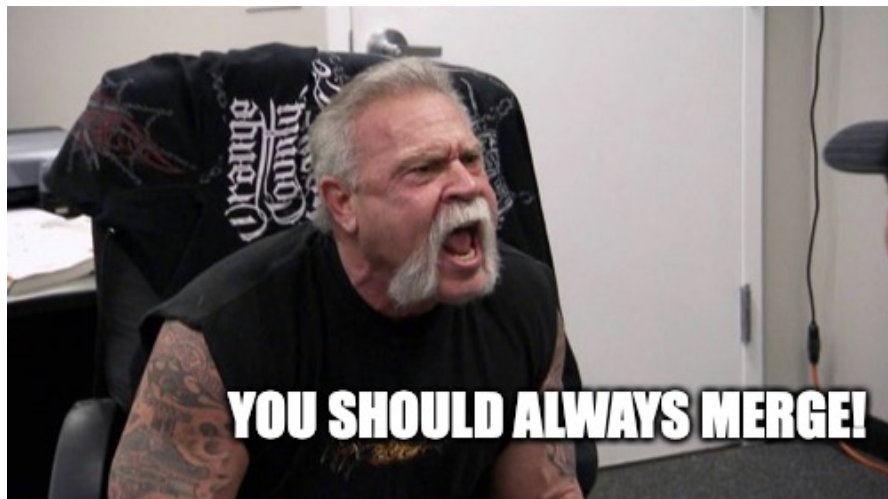
Rebasing: another way to combine **main** and **subbranch**.

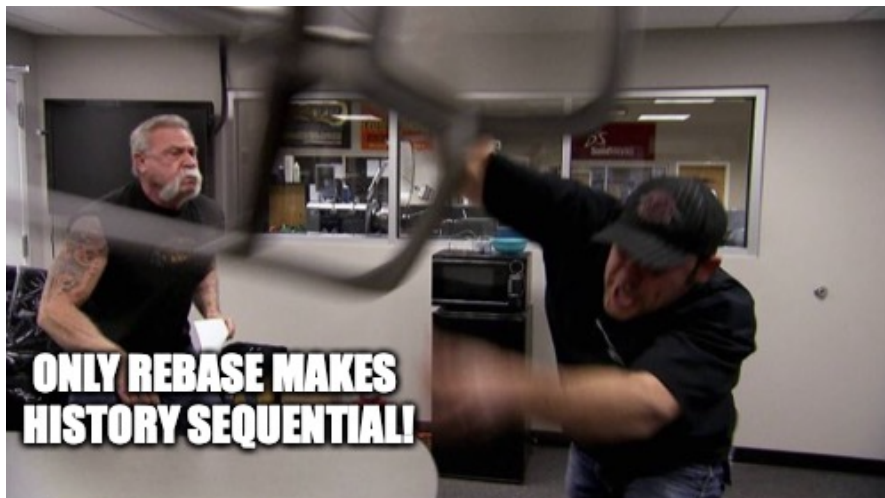
Rebase creates a linear (unbranched) history of commits.

This is a matter of some controversy.













## How to Merge

From `main` branch,

```
git merge subbranch
```

will merge the work done on `subbranch` into the `main` branch.

## How to Rebase

From `subbranch`,

```
git rebase main
```

will add work of `subbranch` as a downstream commit of `main`.

## How to Rebase

From `subbranch`,

```
git rebase main
```

will add work of `subbranch` as a downstream commit of `main`.

But then, update `main` by moving to `main`, then rebasing:

```
git checkout main
```

```
git rebase subbranch
```

## How to Rebase

From `subbranch`,

```
git rebase main
```

will add work of `subbranch` as a downstream commit of `main`.

But then, update `main` by moving to `main`, then rebasing:

```
git checkout main  
git rebase subbranch
```

Now, branches are in sync, same commit.

To learn branching,

<https://learngitbranching.js.org>

- ▶ Complete Intro Sequence 1-3 (*Intro*, *Branching*, and *Merging*)
- ▶ (Bonus: Get through level 4, *Rebasing*)
- ▶ Read every message terminal, in terminal, and file list each step.

## Pull Requests and Forks

## Pull Requests

Issues, focused on branches and merging.

# Pull Requests

Issues, focused on branches and merging.

Three components:

- ▶ Conversation
- ▶ Commits
- ▶ Diffs



# Forking

*Fork*: your *copy* of a repo you don't control

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo
- ▶ Stay current with canonical version

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo
- ▶ Stay current with canonical version
- ▶ Create branch

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo
- ▶ Stay current with canonical version
- ▶ Create branch
- ▶ Edit

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo
- ▶ Stay current with canonical version
- ▶ Create branch
- ▶ Edit
- ▶ Issue pull request

# Forking

*Fork*: your *copy* of a repo you don't control

- ▶ Clone repo
- ▶ Stay current with canonical version
- ▶ Create branch
- ▶ Edit
- ▶ Issue pull request
- ▶ (Then, later pushes update pull request)