

Version Control with Git

Modern Plain Text Social Science

Week 06

Kieran Healy

kieran.healy@duke.edu

September 2026

Plain Text Credo Again



3 rules:

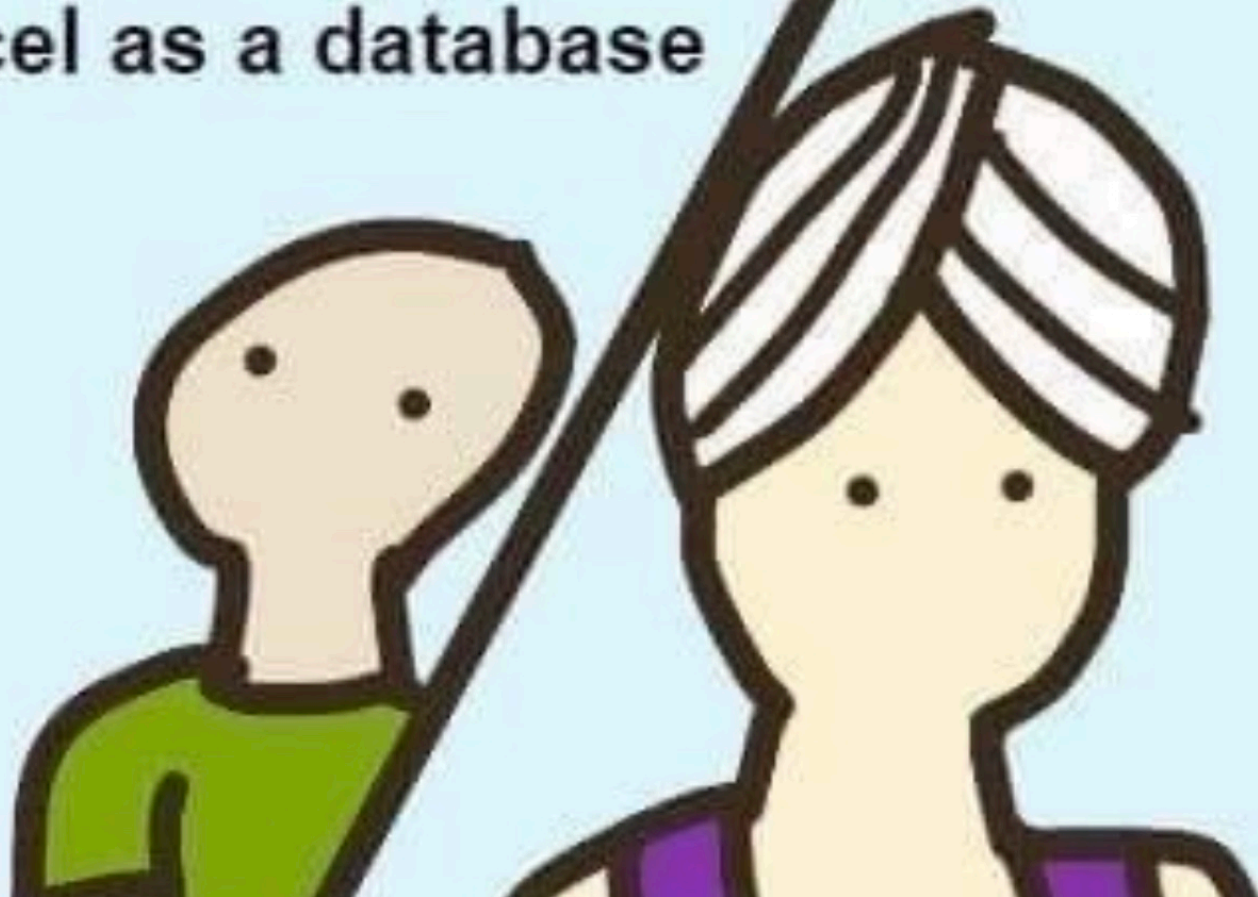
-no wishing for death

-no falling in love

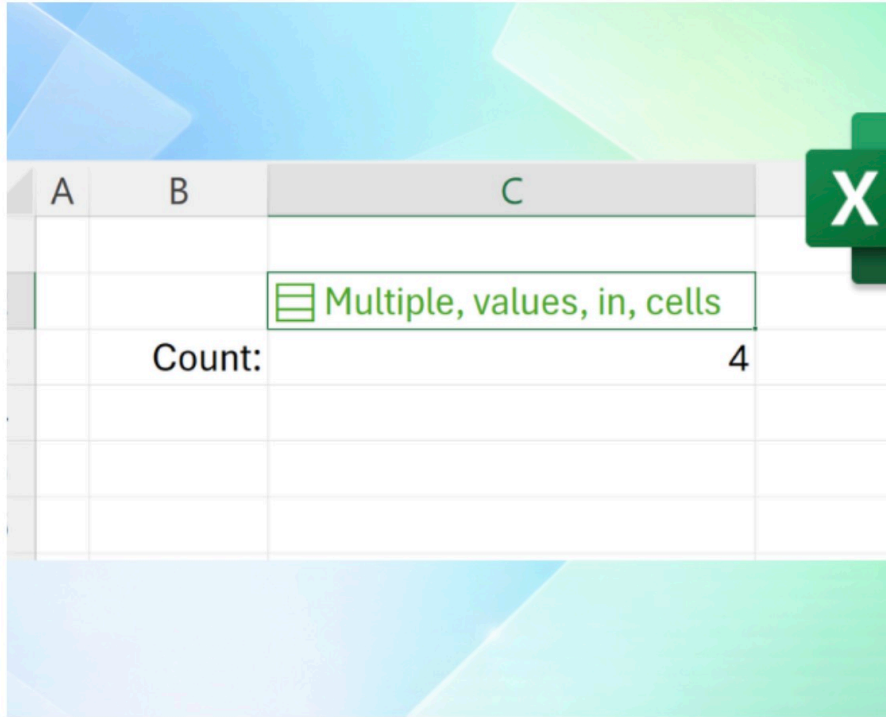
-no bring back dead people

I want to use
Excel as a database

There are 4 rules



Recent news



EXCEL BLOG 3 MIN READ

Excel now supports multiple values in a single cell

Although, in fairness ...

```
gapminder ▷  
  group_by(continent, year) ▷  
  nest()
```

```
# A tibble: 60 × 3  
# Groups:   continent, year [60]  
  continent year data  
  <fct>      <int> <list>  
1 Asia      1952 <tibble [33 × 4]>  
2 Asia      1957 <tibble [33 × 4]>  
3 Asia      1962 <tibble [33 × 4]>  
4 Asia      1967 <tibble [33 × 4]>  
5 Asia      1972 <tibble [33 × 4]>  
6 Asia      1977 <tibble [33 × 4]>  
7 Asia      1982 <tibble [33 × 4]>  
8 Asia      1987 <tibble [33 × 4]>  
9 Asia      1992 <tibble [33 × 4]>  
10 Asia     1997 <tibble [33 × 4]>  
# i 50 more rows
```

(Next week, when we do some stuff on iteration with the `map()` family of functions we'll see why we might want something like this.)

Keep Backups

You Need Backups That Are

You Need Backups That Are

Secure

You Need Backups That Are

Secure

Automated

You Need Backups That Are

Secure

Automated

Redundant

You Need Backups That Are

Secure

Automated

Redundant

Remote

These are Not Backups

These are Not Backups

These are Not Backups

Your computer

These are Not Backups

Your computer

Your old computer

These are Not Backups

Your computer

Your old computer

An attachment you emailed to yourself

These are Not Backups

Your computer

Your old computer

An attachment you emailed to yourself

A shared folder called “Important Stuff” that you never update

These are Not Backups

Your computer

Your old computer

An attachment you emailed to yourself

A shared folder called “Important Stuff” that you never update

A USB Drive you updated twice and forget where it is maybe it is in the drawer in the kitchen oh god oh god i hope so please oh please oh god i am so screwed

Not Nearly Good Enough

These services are good for a lot of things, and some of them are *like* backups in *some* ways, but they are also **not really backups**.

Google Drive

Dropbox

Box

Git/GitHub

Pay for a Reputable Service

E.g., **Backblaze**.

If ensuring that you have **access to months or years of your work** in the event your laptop gets stolen, is left behind in an airport, “just stops working”, is accidentally dropped in the bath, or is found submerged in a foot of water after a hurricane¹ is **not** worth nine bucks a month to you, perhaps reconsider your personal risk tolerance, the implicit value you place on your time, and your career choices.

“When it comes to losing your data, the universe tends toward maximum irony. Don’t push it.” — Jamie Zawinski

1. All things that have happened to people I know personally.

Remember

Secure

Automated

Redundant

Remote

Knowing what you did

**“To remember
everything is a
form of madness.”**

Brian Friel, *Translations*, Act III.

We need to keep track

We need to keep track

When you get down to it a “research project” is a folder on your computer that contains a bunch of files and subfolders.

We need to track things *at the level of our project*.

We need to keep track

When you get down to it a “research project” is a folder on your computer that contains a bunch of files and subfolders.

We need to track things *at the level of our project*.

There’s a long history of software that can keep track of how files in projects have changed, and who did things to which files. A tool like this is called a *Version Control System* or a *Revision Control System*.

We need to keep track

When you get down to it a “research project” is a folder on your computer that contains a bunch of files and subfolders.

We need to track things *at the level of our project*.

There’s a long history of software that can keep track of how files in projects have changed, and who did things to which files. A tool like this is called a *Version Control System* or a *Revision Control System*.

Modern solutions are sometimes called *Distributed Version Control Systems* because of some additional capabilities they have.

Problems a (D)VCS Solves

Track changes you make to project files.

Do this in terms of or at the level of the project as a whole.

Keep track of who made which changes to a file and when.

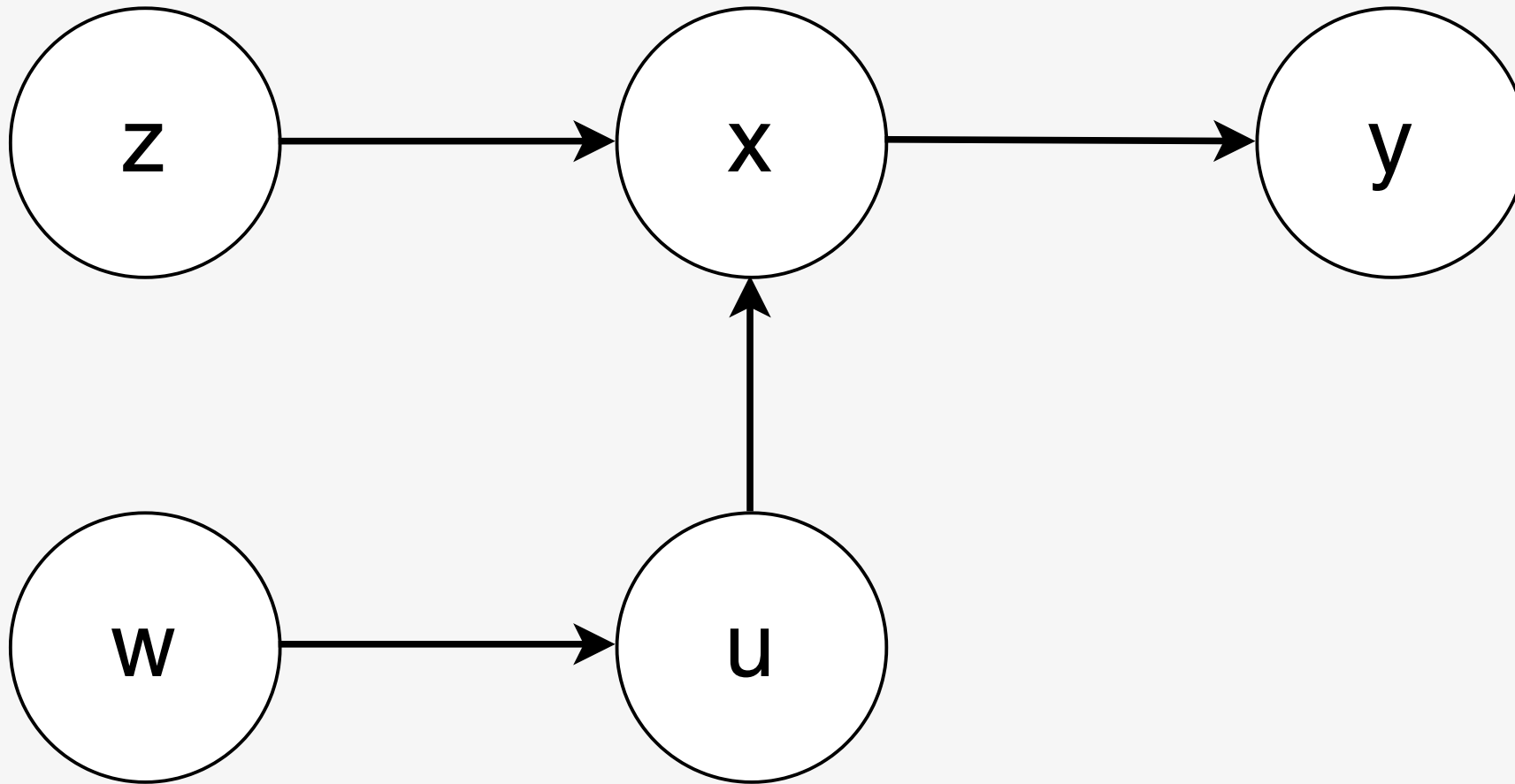
Jump back to any previous point to see the state of the project then.

Manage new content or revisions by breaking them into branches or offshoots that can also be tracked.

Merge these offshoots and branches back into the main line of the project as needed.

Figure out when a change that breaks something later got introduced earlier.

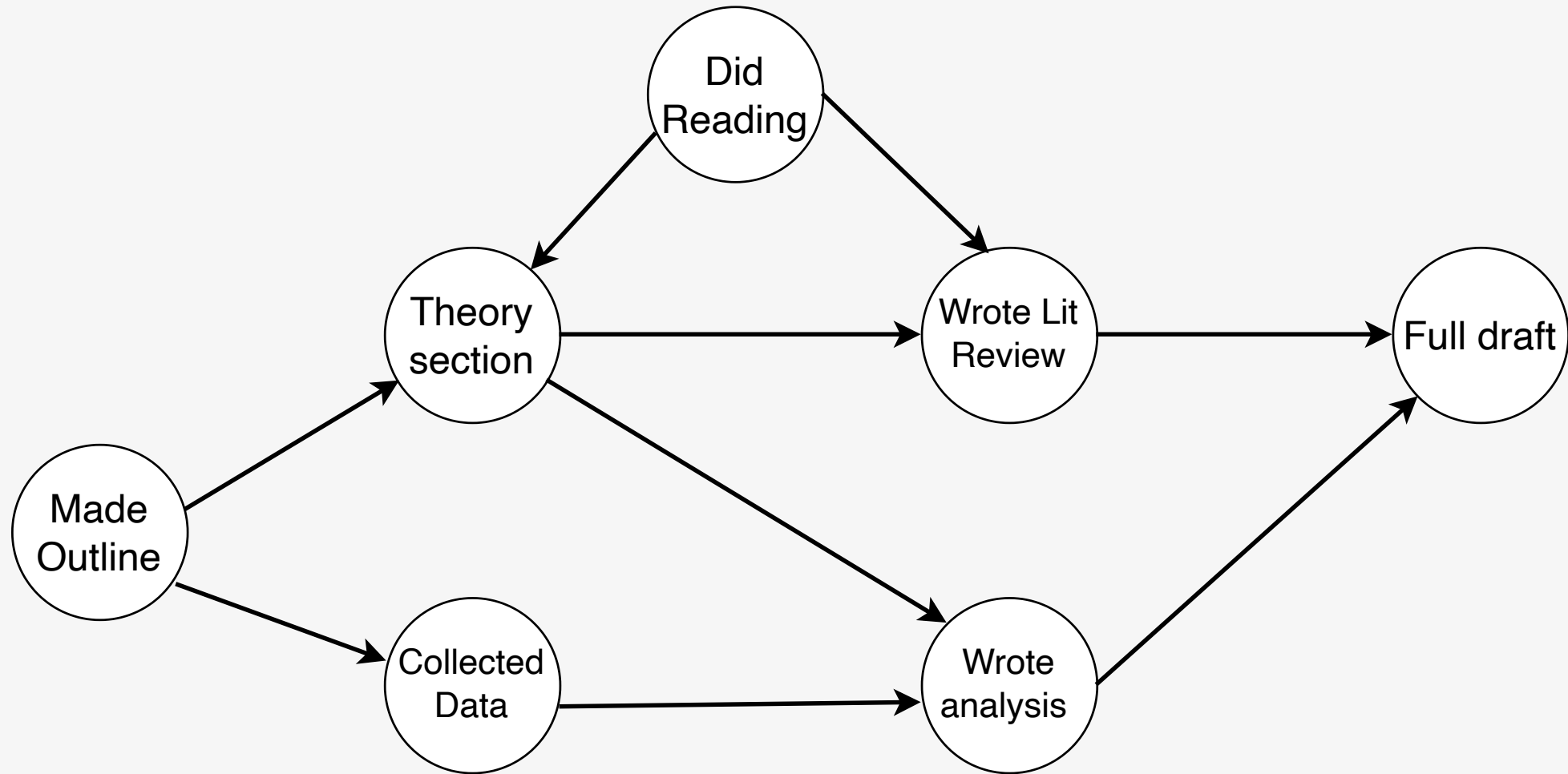
A VCS is a DAG



A causal diagram as a directed acyclic graph

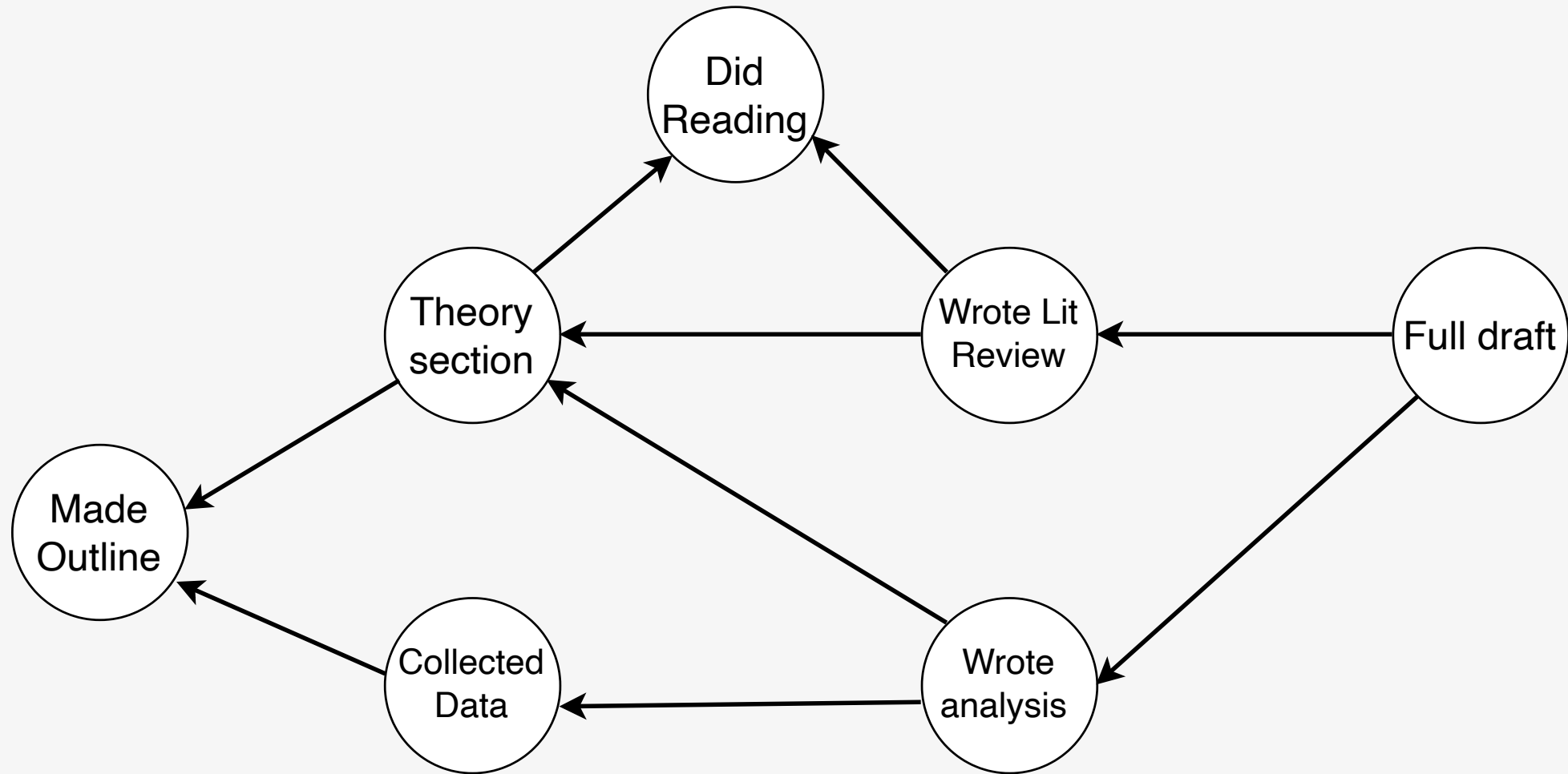
You are going to see a *lot* of diagrams like this in papers, talks, and seminars you're required to take.

A VCS is a DAG



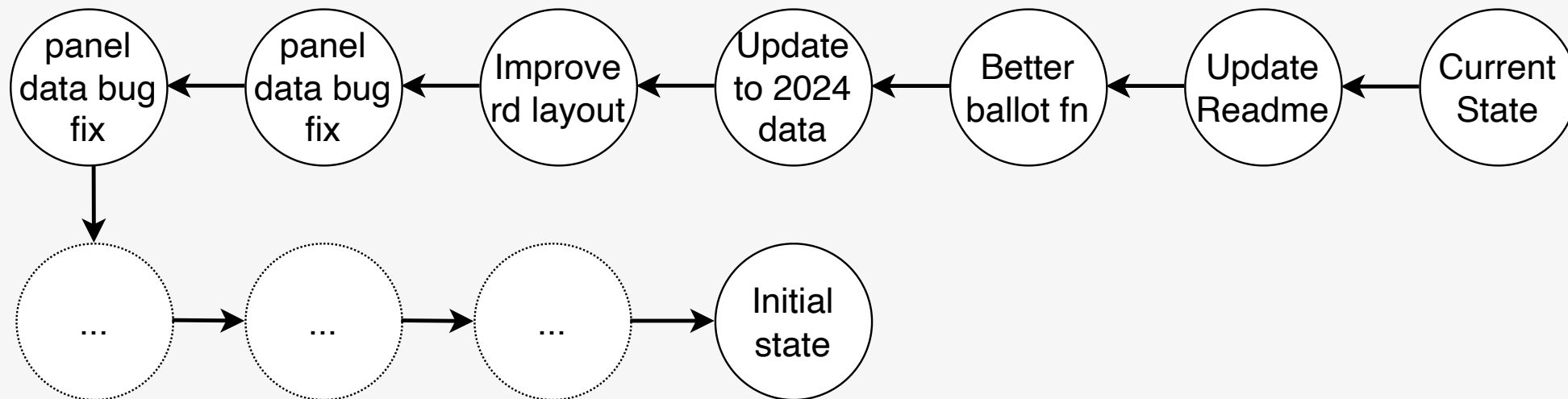
Causal history of an imaginary paper where the arrow means 'led to'

A VCS is a DAG



Causal history of an imaginary paper where the arrow means 'depends on'

A VCS is a DAG



History of mostly linear changes to an imaginary project

Which brings us to ...

```
769d48c main vartib origin/main citation
c913191 README
dea0aa2 News
9ed92cf 24r3 2024R3
827d7f0 Update links to GSS pdfs
473d1ee News and citation updates
d72a3a3 Update to GSS 1972-2024 R2, Oct 2025
4722b7e Update inline help sshot
3176dd9 Remove global var in gss_which_years
3582197 Package theme
9667656 link to gssrdoc
0424bb6 Fix gss_get_yr for 2024
acc8f77 Fix link to gssrdoc
2080bda 2024 Remove ref to gss_dict
0c0e07a Update r-cmd-check action
dae6ead Update pkgdown action
ed90704 2024 update
541cd79 Change theme
ae6e5cf Remove preferably template
114767b Update README
688cde0 update workflow action
3858413 Update docs
20239d5 panelfix origin/panelfix Panel data bug fixes
ea9202a Fix typo
8bc20ec Fix ref to documentation
0437063 Bump citation version
9a91513 Typos
275f463 Version bump and runiverse badges
2124f59 Remove refs to gss_dict
d9b5b8e nodocs origin/nodocs Experimental nodoc build
25642bc Better ballots obj
```

A portion of the git log for the gssr package

Git

Git is the industry-standard DVCS. There are others, but it's the main one.

It's very powerful.

It's used to run the software development projects of gigantic corporations that must coordinate huge projects with contributions by hundreds or thousands of developers.

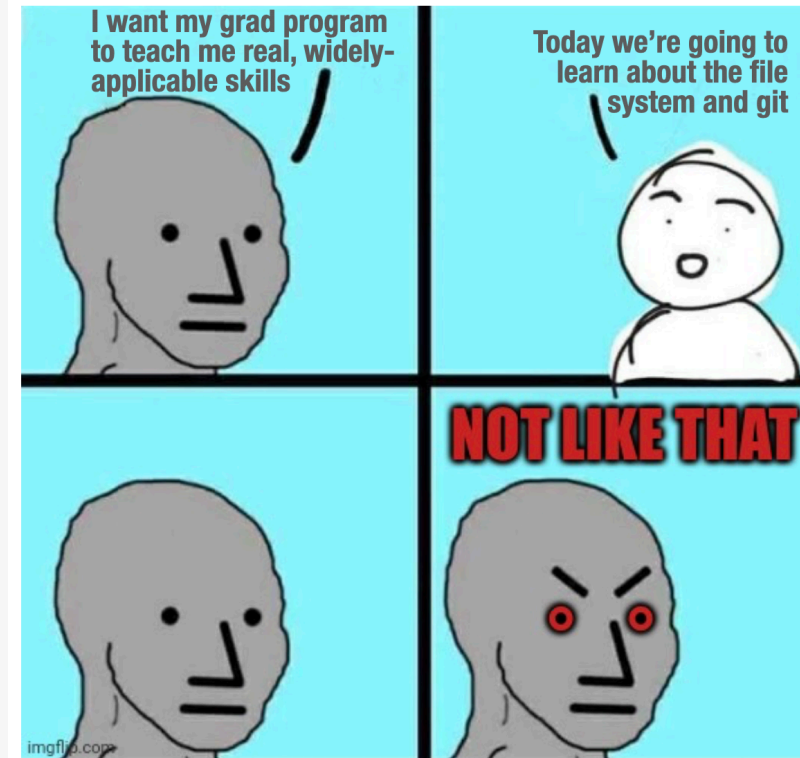
It makes professionals and amateurs alike cry on the regular. But there are a bunch of cry-minimizing ways of using and interacting with it.

Git

We can start by using a *very small fraction* of git's functionality. This can be enough to get by for a long time and is *way* better than nothing.

As projects get more complex—with multiple contributors to multiple tasks—then git offers a way of working that can help bring order to this complexity, and leave a record in its wake.

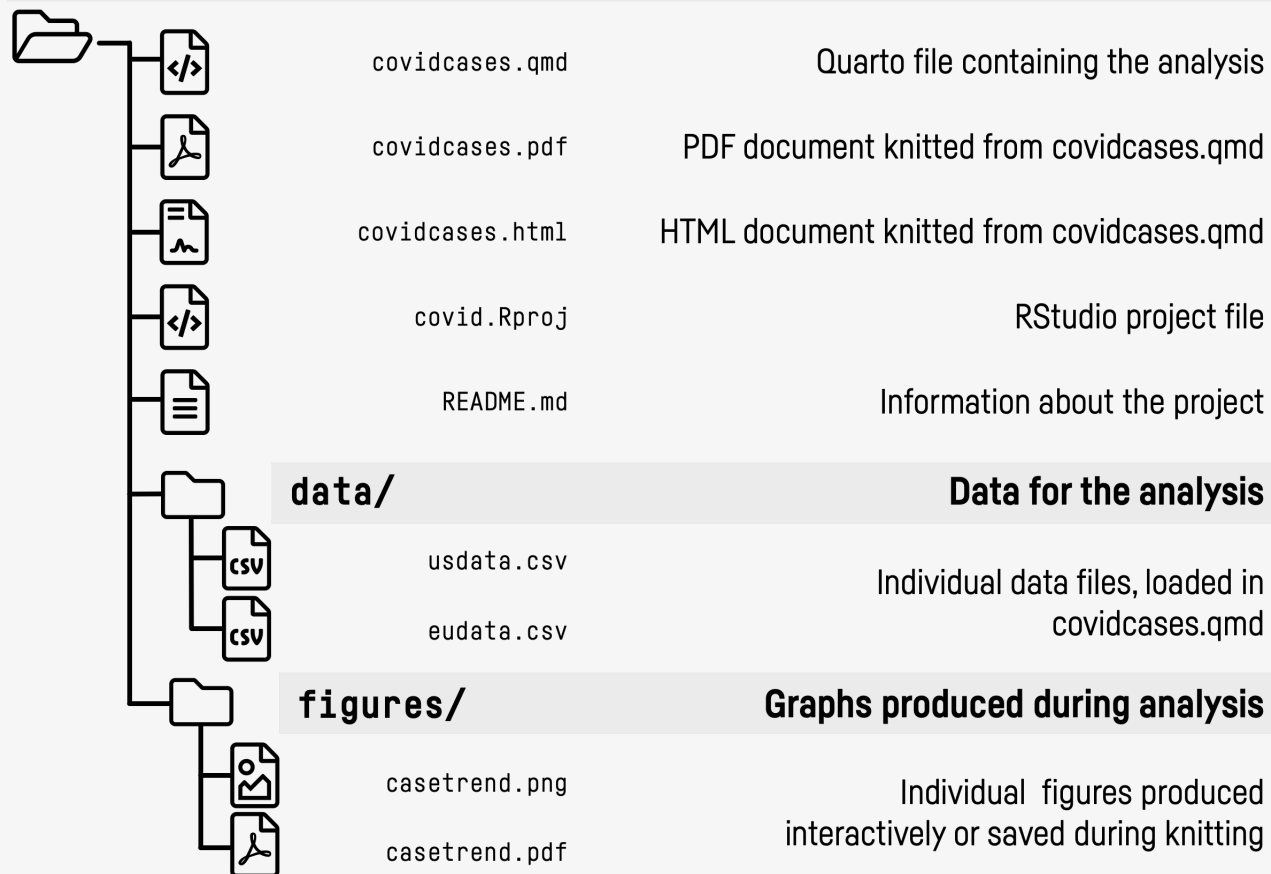
Again, Git is the standard for basically *everything* that gets done in the world of technical computing. All software development is done in this way, or something functionally the same as it.



Git as a Logbook

Our sample project

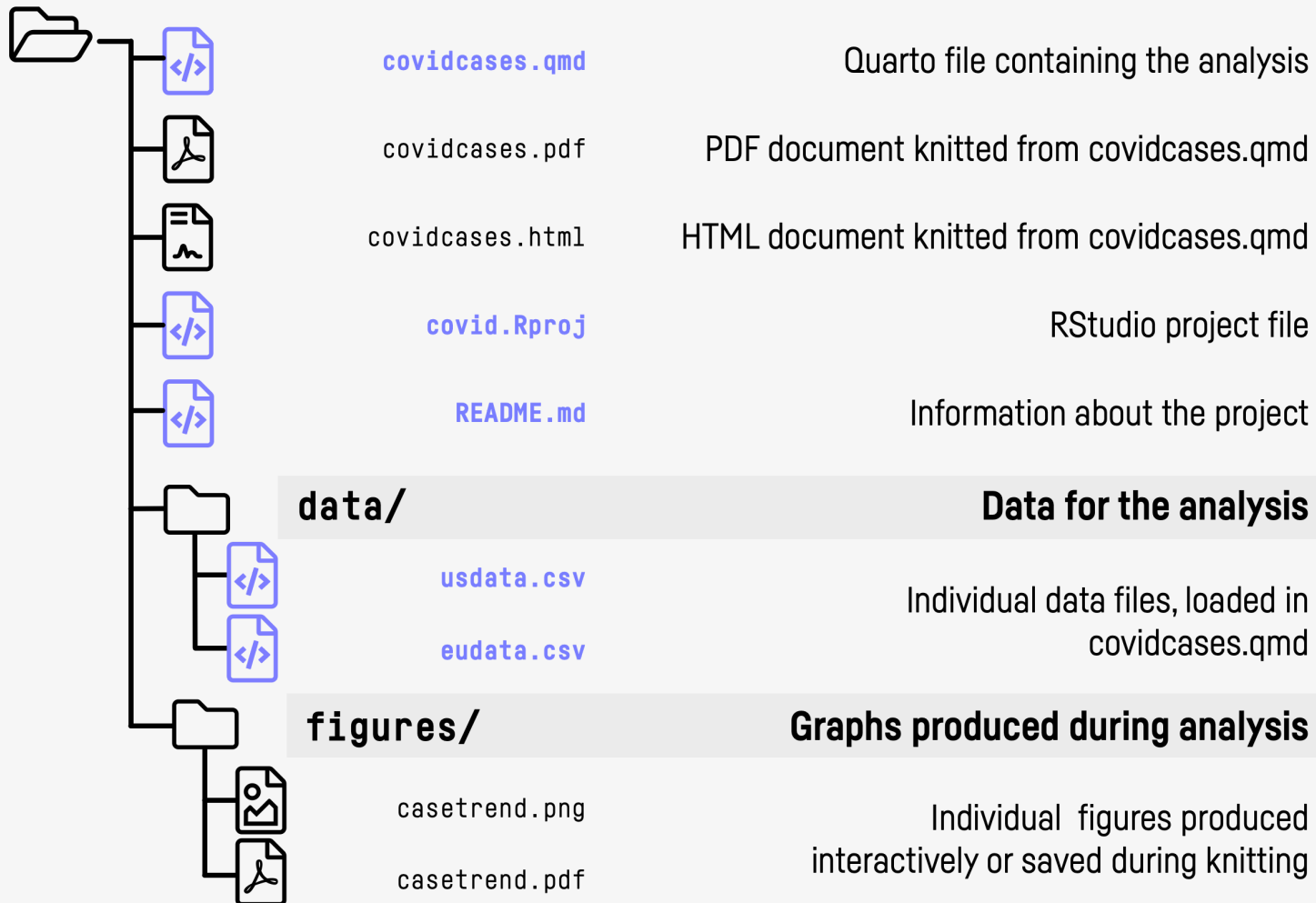
covid-project/



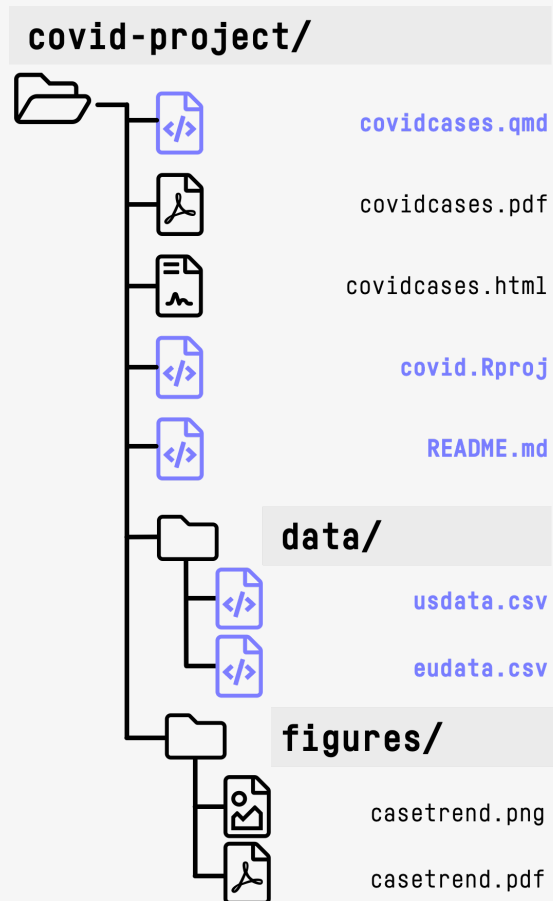
This project can be found at <https://github.com/kjhealy/covid-project>

What **must** we track?

covid-project/



This is our local folder



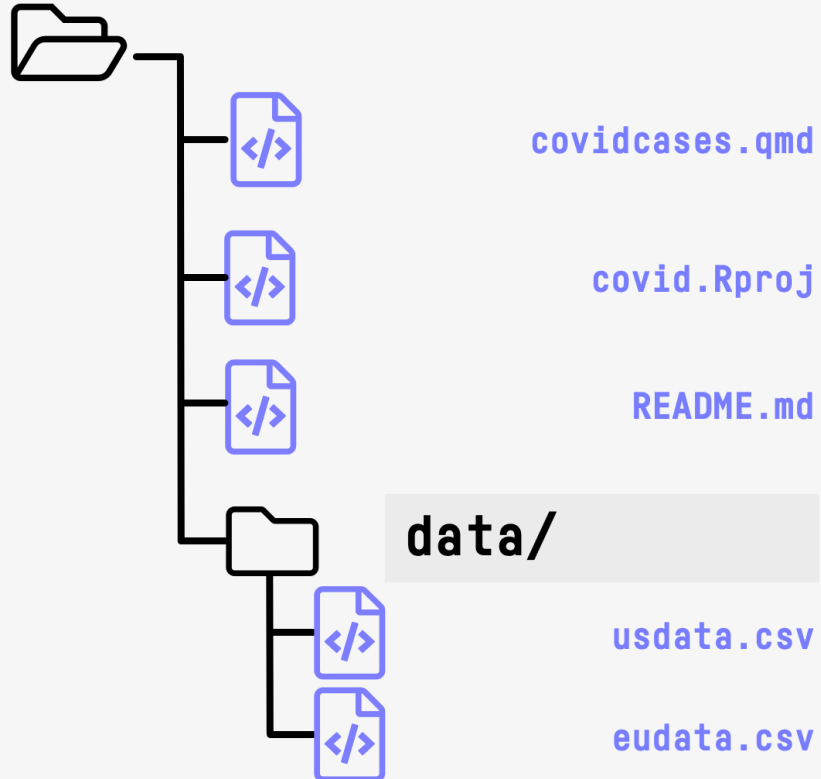
Lives on our laptop.

Just a regular folder with files.

At any moment will have stuff we need to track, and outputs and other things we don't need to track.

This is what we need to track

covid-project/



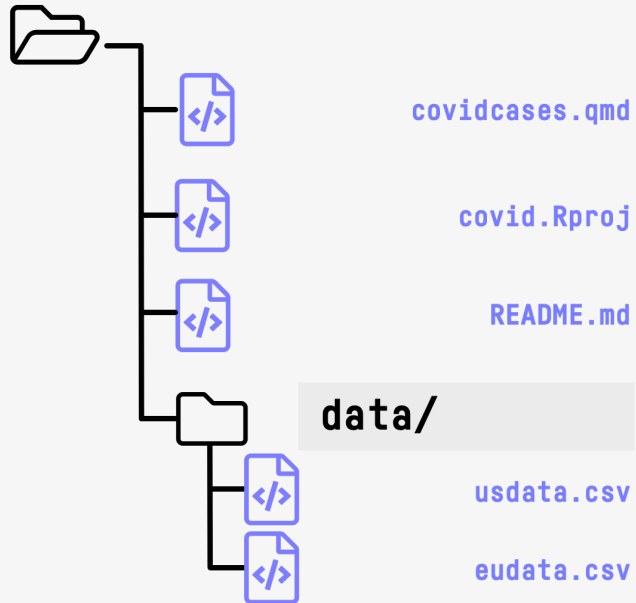
Sidenote: It's true that, even though they are generated by code and not 'real', for figures and tables (and final outputs like papers) we will often want to know which version of the code and data they were produced with.

We'll come back to this in more detail later on. We can write an R function to stamp a version identifier on the output.

Initialise a repository

Local Folder

covid-project/



Our record of changes and edits to files is called a *repository* or repo.

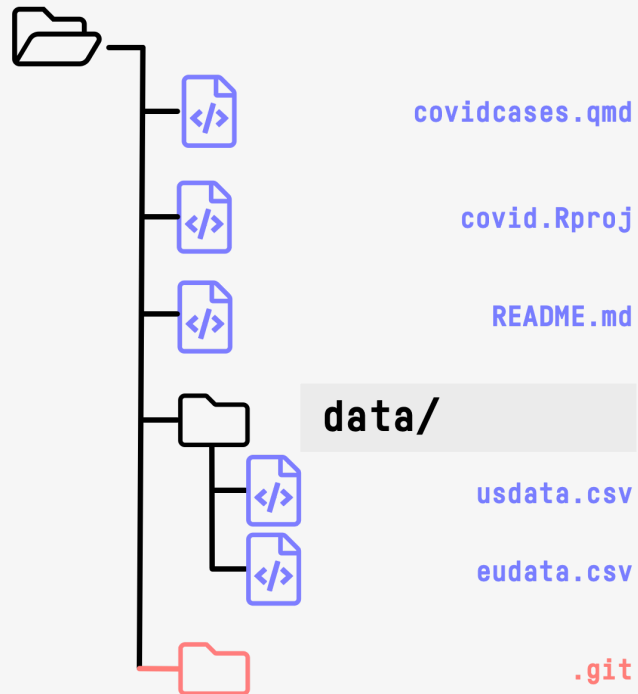
It has to exist somewhere.

With **git**, it exists as a “hidden” folder that has a record of all the changes. It’s stored inside the project. To create it, we *initialise* the repo.

Initialise a repository

Local Folder

covid-project/



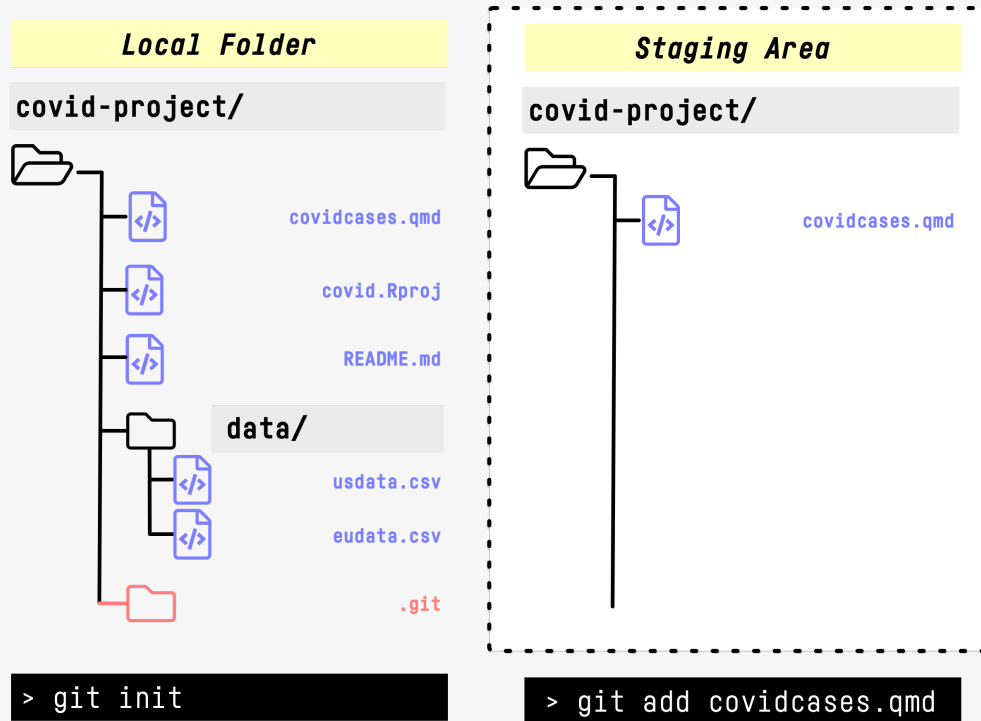
```
> git init
```

The `.git` folder is “hidden” because its name begins with a dot.

Inside it contains its own file and folder hierarchy for keeping track of changes to our project.

Now we need to tell git what to track.

Staging a file

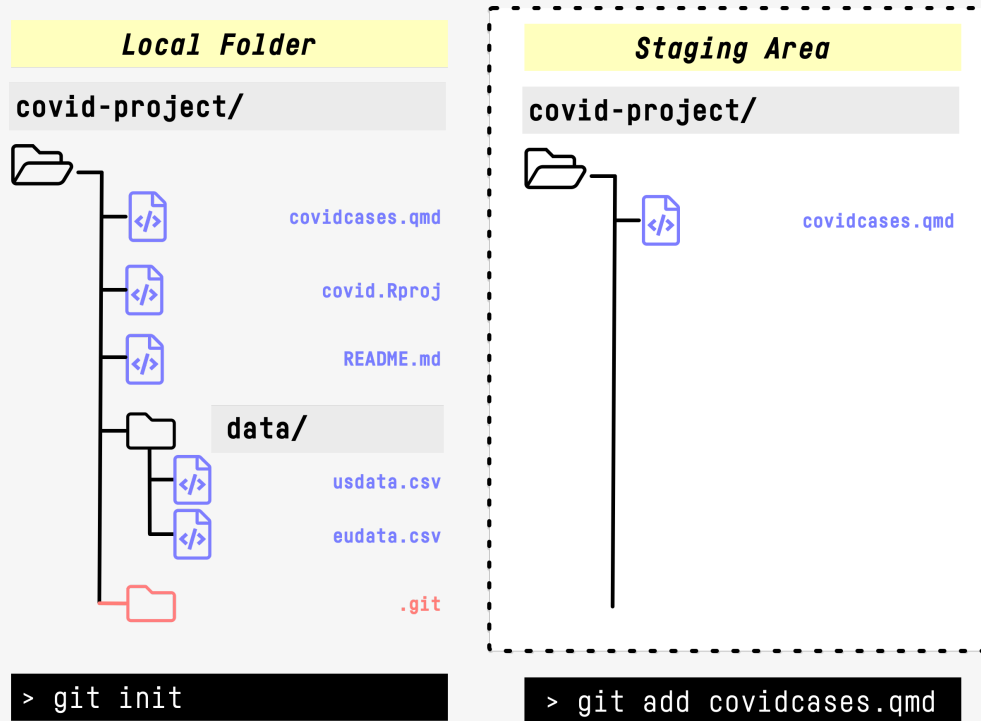


The first step is to “stage” a file with the command `git add <filename>`

This adds the file to an index of things git is going to manage.

Git keeps this index inside the `.git` folder.

Staging a file



No changes are recorded at this point.

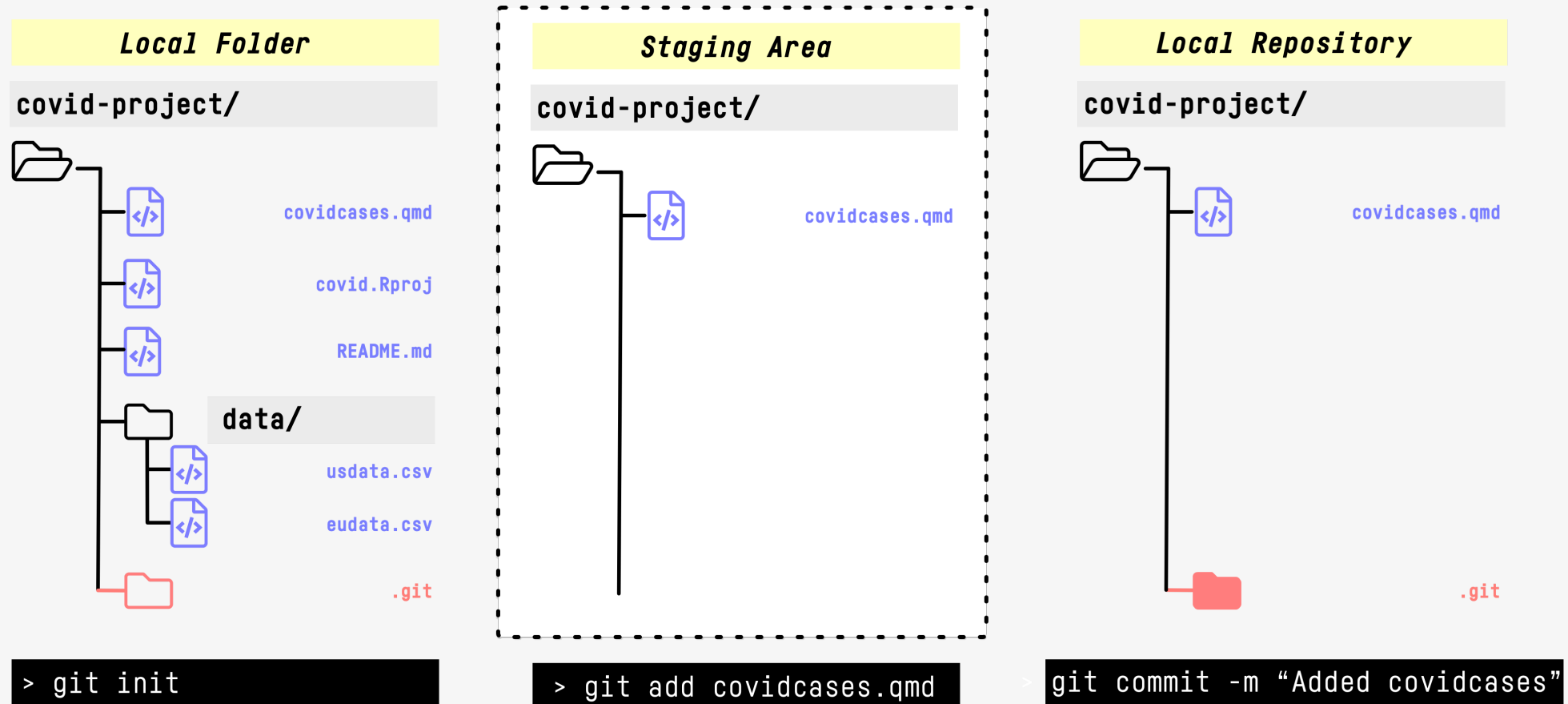
Nothing changes in our local folder.

We've just added a file to a list of items to be tracked.

As the name suggests, conceptually the staging area is preparatory.

... Preparatory to what?

Committing a file



Committing a file makes a record of it in our local repository.

Committing a file



.git

```
[main (root-commit) f43f6f1] Added covidcases  
1 file changed, 11 insertions(+)  
create mode 100644 covidcases.qmd
```

```
> git log  
  
commit f43f6f10ba5cd1a57430c032c8c14f056b154e9b [HEAD -> main]  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 17:34:08 2023 -0400  
  
    Added covidcases
```

The 64-bit hexadecimal number is an **SHA Hash**.

It's derived from the file(s) being committed and it uniquely identifies the changes associated with that commit.

The message added with **-m** is for our benefit. It should be terse and informative about what's new, fixed, or changed in the commit.

Committing a file



`.git`

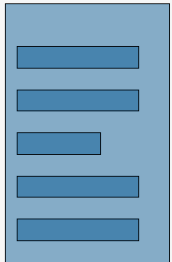
```
> git log
```

```
commit f43f6f10ba5cd1a57430c032c8c14f056b154e9b [HEAD -> main]
```

```
Author: Kieran Healy <kjhealy@gmail.com>
```

```
Date: Mon Sep 18 17:34:08 2023 -0400
```

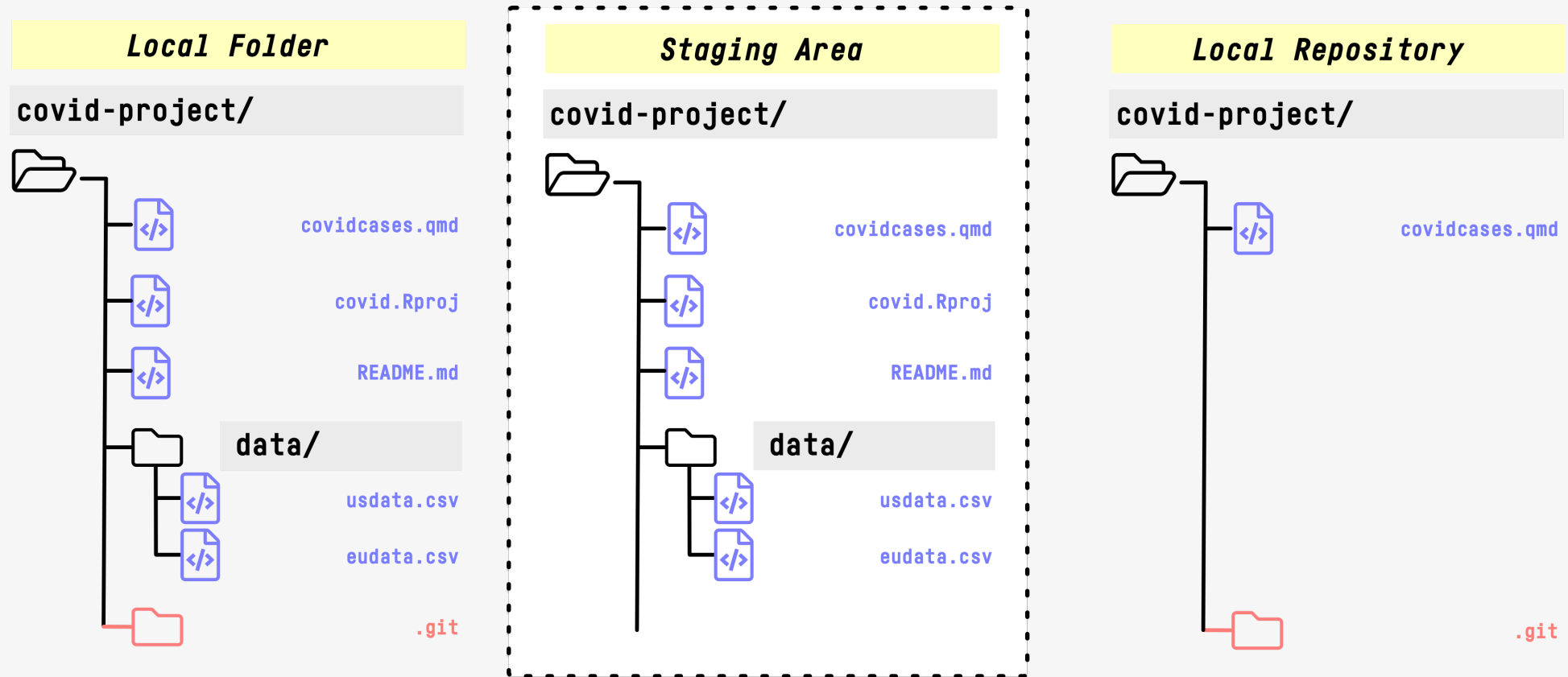
```
Added covidcases
```



First commit contains
the actual file or files

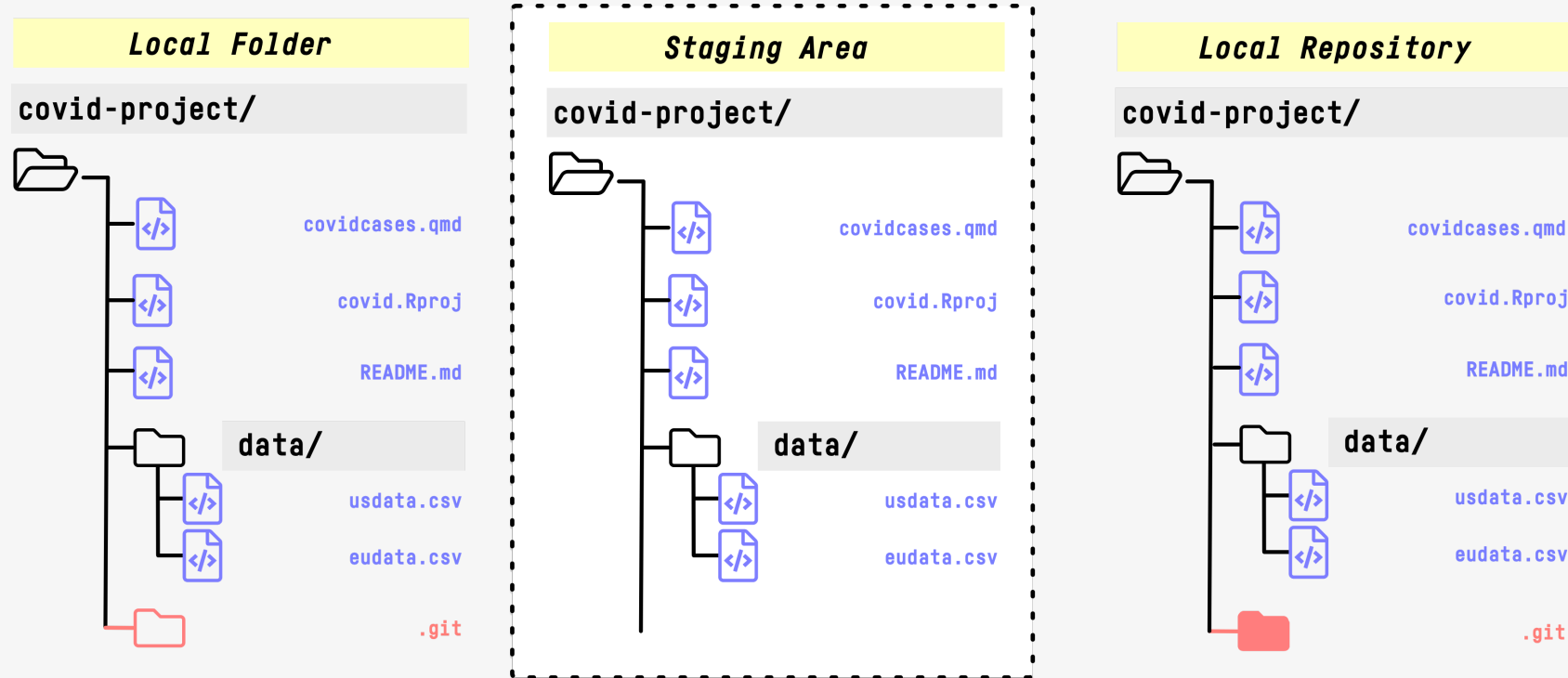
Inside the `.git` folder, a compressed version of the file is created.

Staging several files ...



```
> git add covid.Rproj  
> git add README.md  
> git add data/*.csv
```

... and committing them



```
> git add covid.Rproj  
> git add README.md  
> git add data/*.csv
```

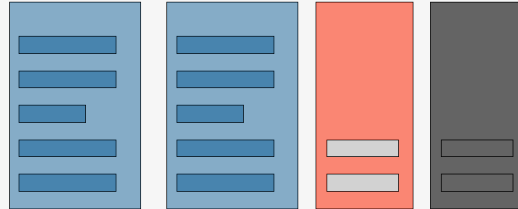
```
> git commit -m "Data and Rproj files"
```

Committing several files



.git

```
[main ef07721] Data and Rproj files
4 files changed, 63514 insertions(+)
create mode 100644 README.md
create mode 100644 covid.Rproj
create mode 100644 data/eudata.csv
create mode 100644 data/usdata.csv
```



More files added, with information about their paths, etc.

```
> git log

commit ef0772106fe4d6f10ee62afd3c951ff14b16efb1 [HEAD -> main]
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 17:38:55 2023 -0400

    Data and Rproj files

commit f43f6f10ba5cd1a57430c032c8c14f056b154e9b
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 17:34:08 2023 -0400

    Added covidcases
```

Our log has two entries now

Remember ...

The *Local Folder* is our project as we (and the file system) can see it.

The *Local Repository* lives inside the special `.git` folder that sits at the root of our project. Conceptually we think of it as its own separate thing that git is managing for us.

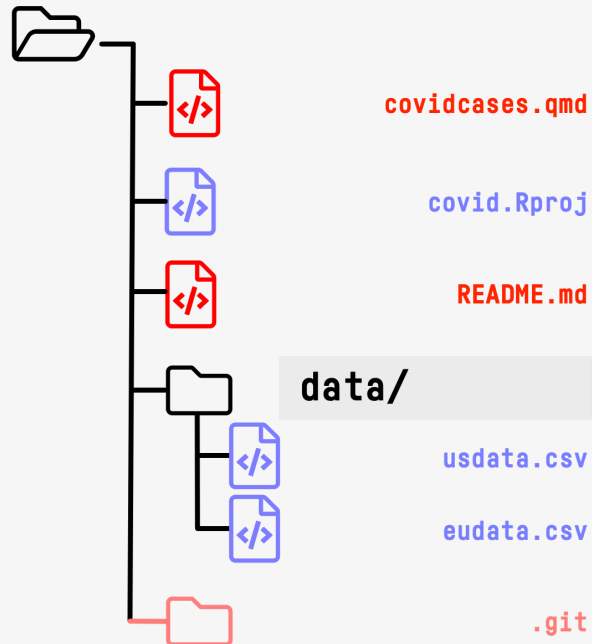
We can move files and folders in and out of our local folder, edit them, and so on, just as we normally would.

But for git to track and manage any of these files or changes to them we need to *add* and *commit* them to the local repository.

Editing and committing

Local Folder

covid-project/



*[Further edits to
covidcases.qmd & README.md]*

```
> git status
```

On branch main

Changes not staged for commit:

[use "git add <file>..." to update what will be committed]

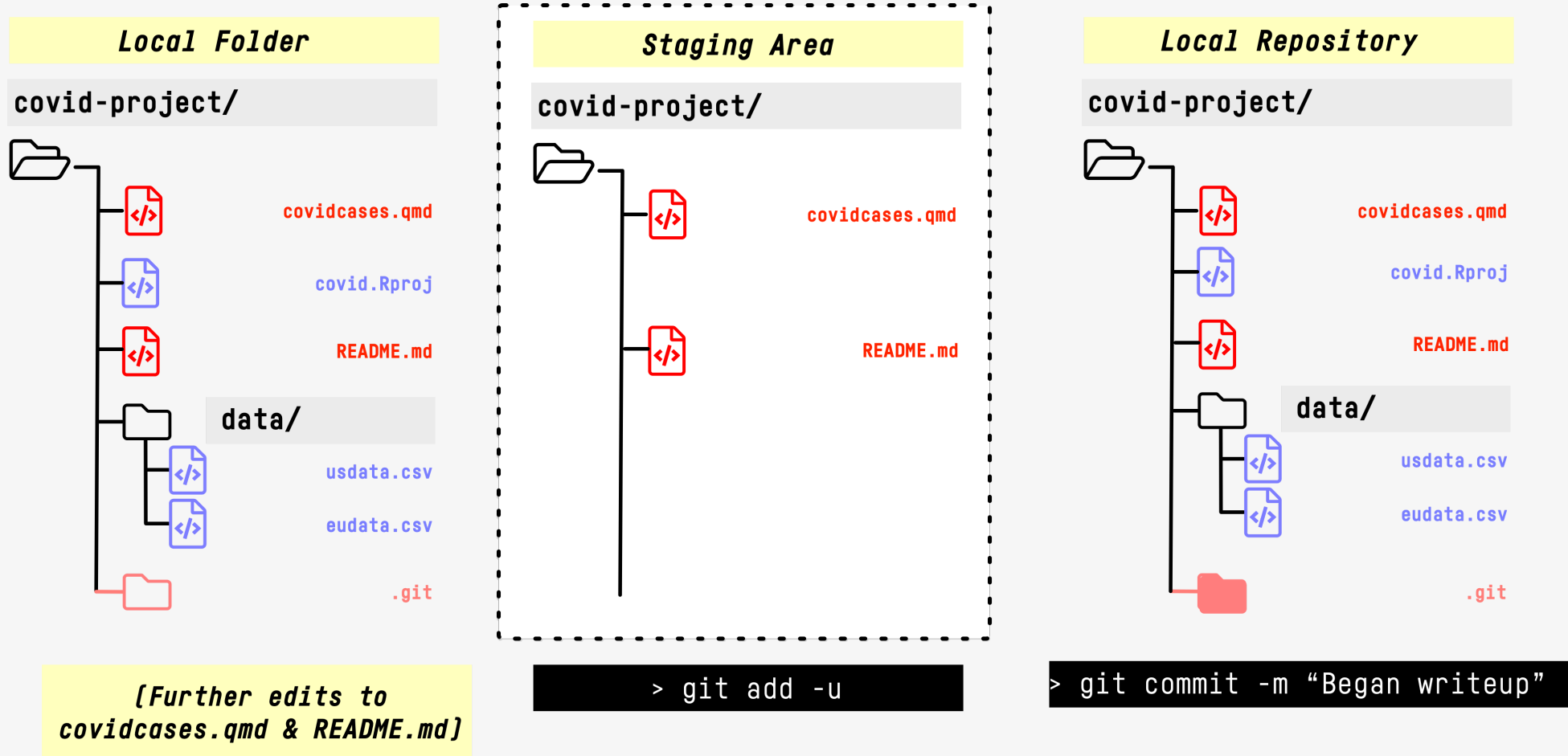
[use "git restore <file>..." to discard changes in working directory]

modified: README.md

modified: covidcases.qmd

no changes added to commit [use "git add" and/or "git commit -a"]

Editing and committing



Storing the changes



`.git`

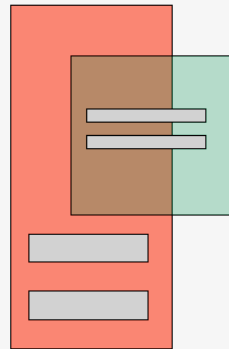
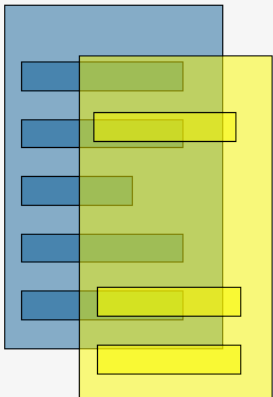
```
> git log -1
```

```
commit 6df6d6b10b127e87115dceac06c869150b67e800 [HEAD -> main]
```

```
Author: Kieran Healy <kjhealy@gmail.com>
```

```
Date: Mon Sep 18 17:49:33 2023 -0400
```

```
Began writeup
```



Subsequent changes
stored efficiently

Remember (again) ...

The *Local Folder* is our project as we (and the file system) can see it.

The *Local Repository* lives inside the special `.git` folder that sits at the root of our project. Conceptually we think of it as its own separate thing that git is managing for us.

We can move files and folders in and out of our local folder, edit them, and so on, just as we normally would.

But for git to track and manage any of these files or changes to them we need to *add* and *commit* them to the local repository.

Git as a Log Book

Working this way, we treat git as managing the files we want to track.

Physically these changes are managed in the `.git` folder inside the project. Again, think of it as a separate entity, the *local repository*.

Every time we decide to record changes to a tracked file in the project we do `git add <filename>` and `git commit -m "Message"`

We write `git add -u` to stage any changes to every *already*-tracked file.

Newly-created files that we want to stage and commit are not picked up by `git add -u`

They must be added by name, either one at a time or with shell-style paths and globs (wildcards).

GitHub

Keeping a remote Log

We have a local repository, but ...

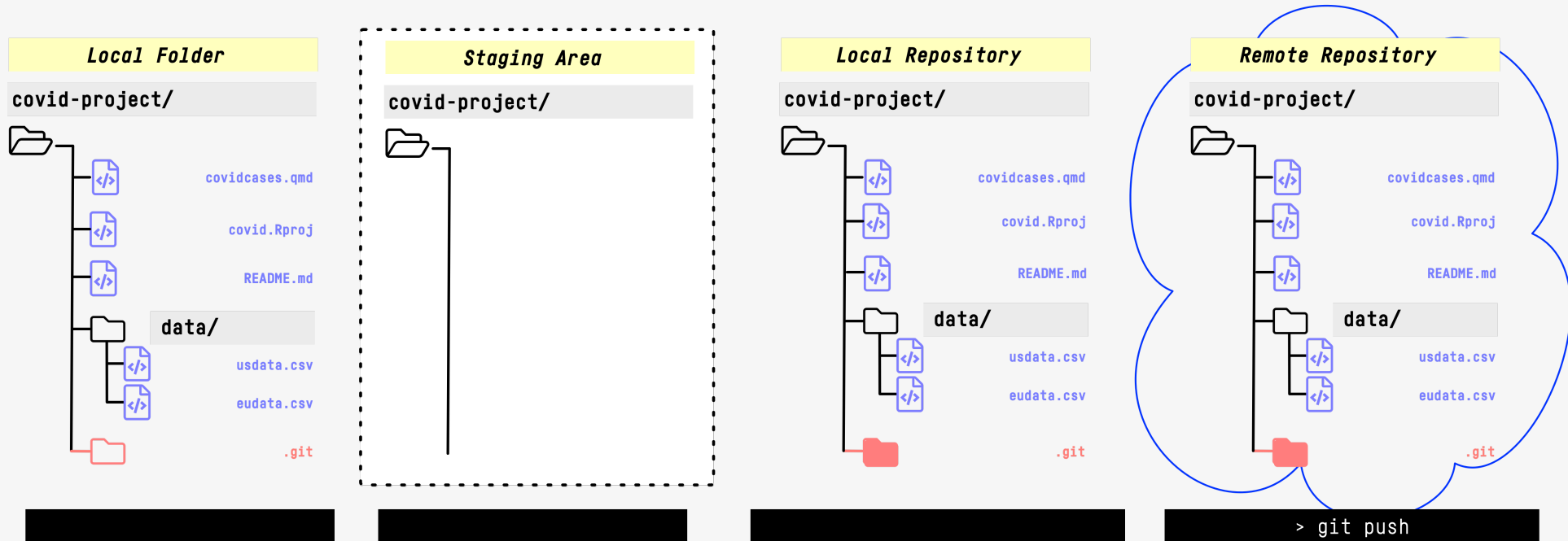
We can also keep copies of it wherever we like.

In [the Cloud](#), for example ...

There's no such
thing as **the Cloud**
It's just someone
else's computer

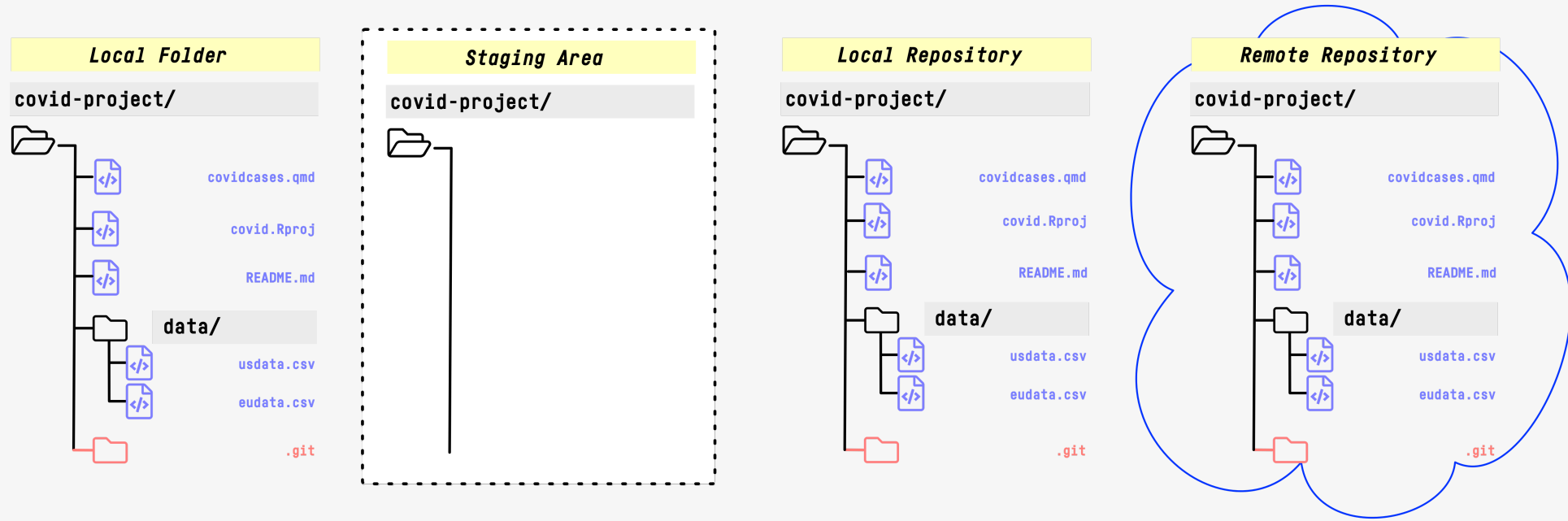
Probably a computer in a big shed in rural North Carolina or Texas or California, etc

Keeping a remote copy



We proceed as before but then **push** changes to GitHub or wherever our remote repository is located.

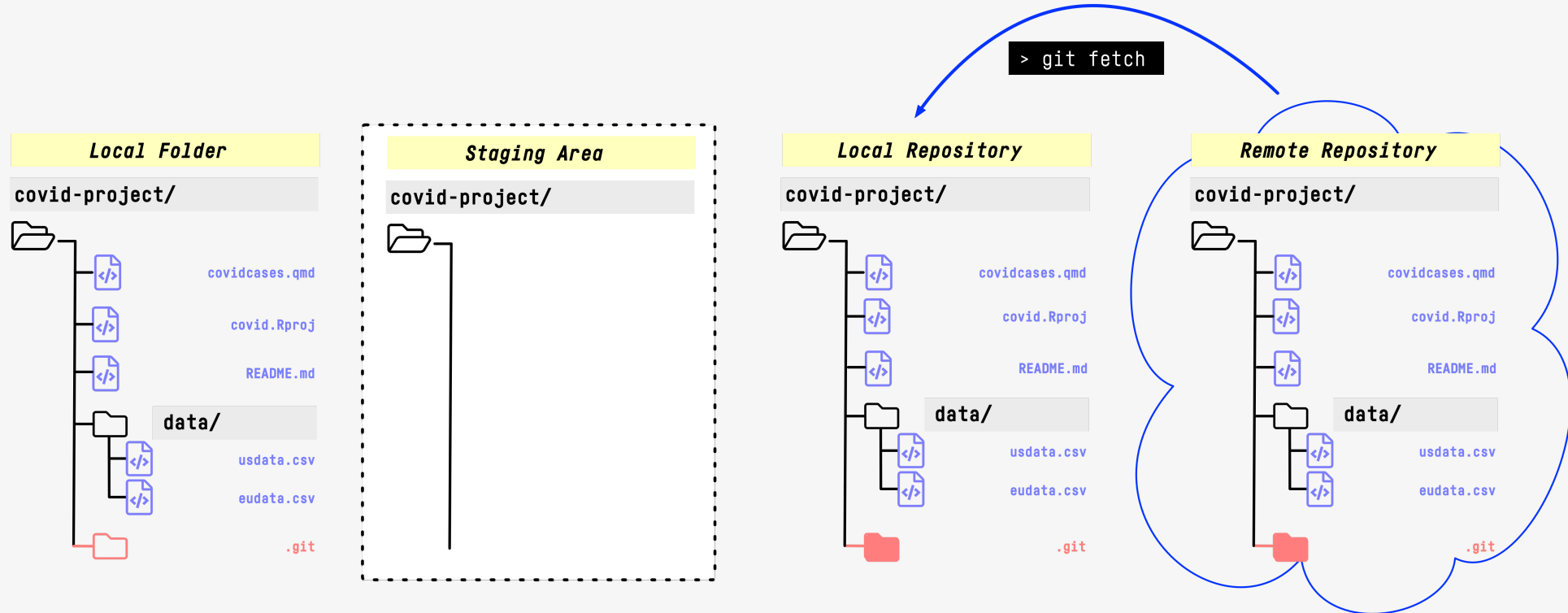
Workflow: GitHub Round Trip



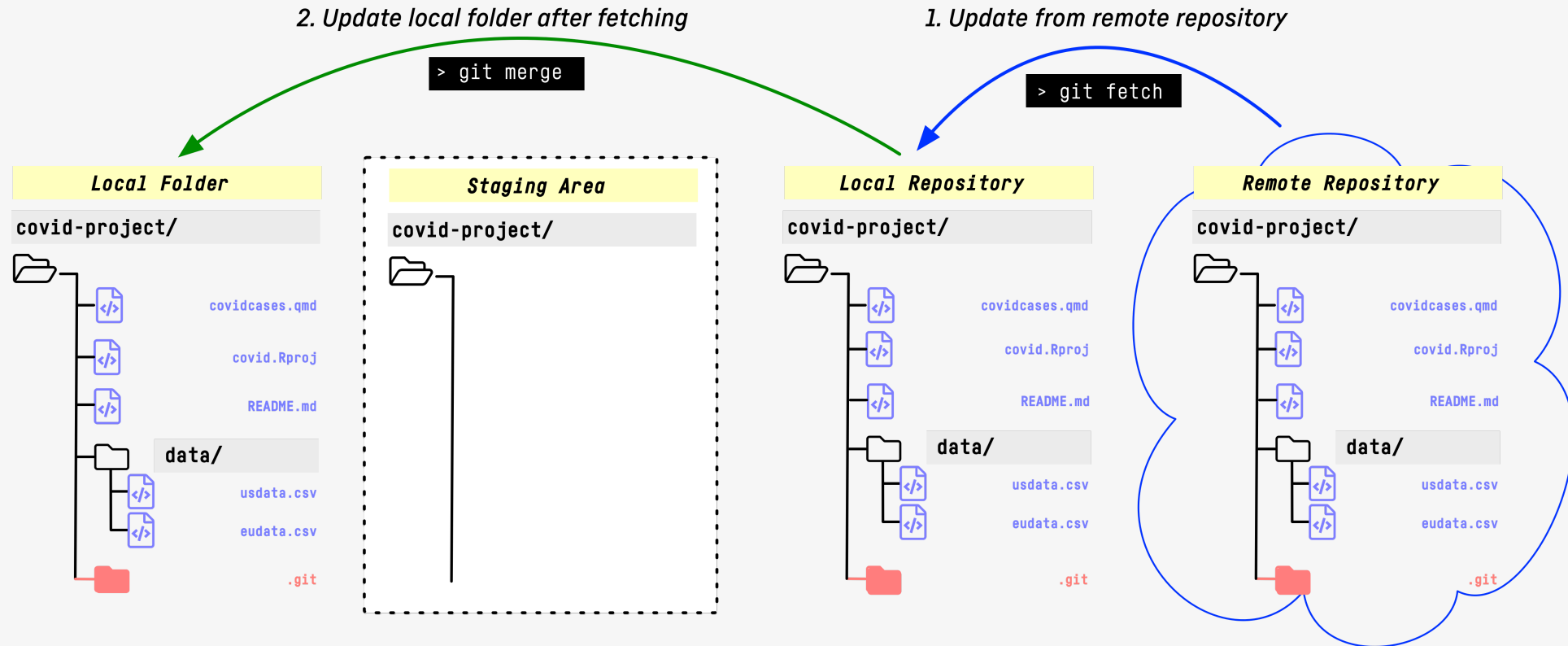
You should already have a GitHub account set up thanks to **Happy Git with R**

Workflow: GitHub Round Trip

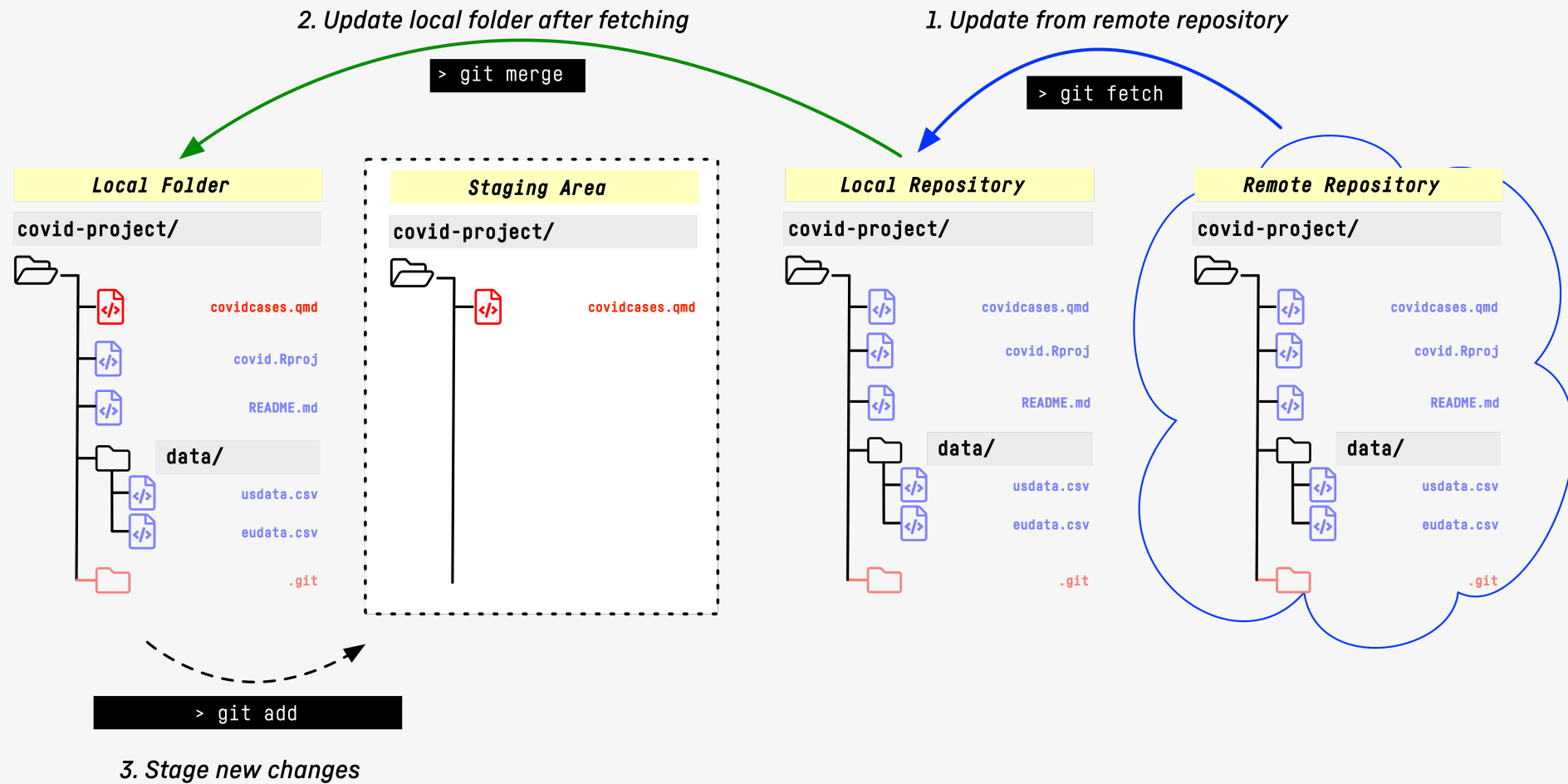
1. Update from Remote Repository



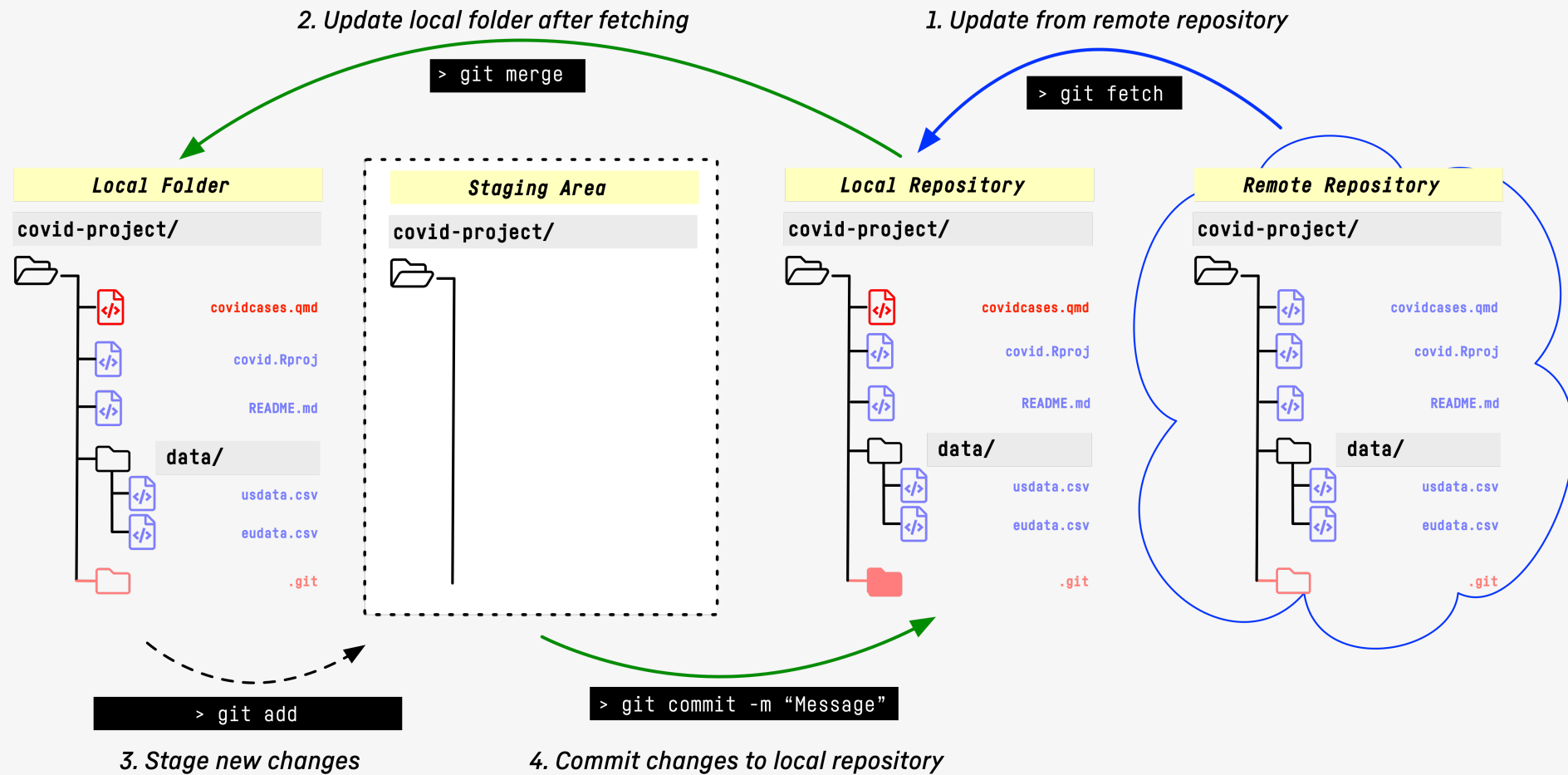
Workflow: GitHub Round Trip



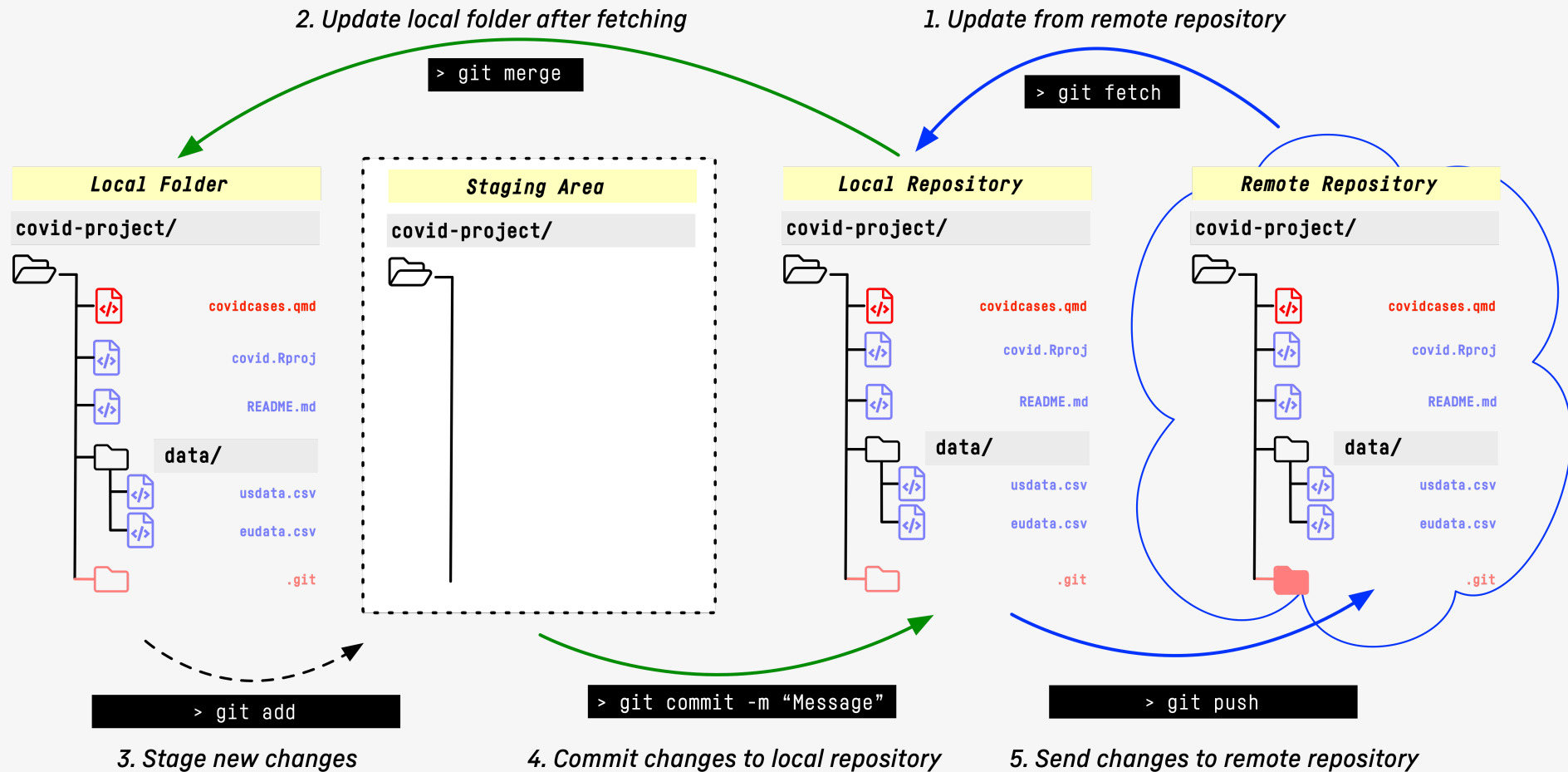
Workflow: GitHub Round Trip



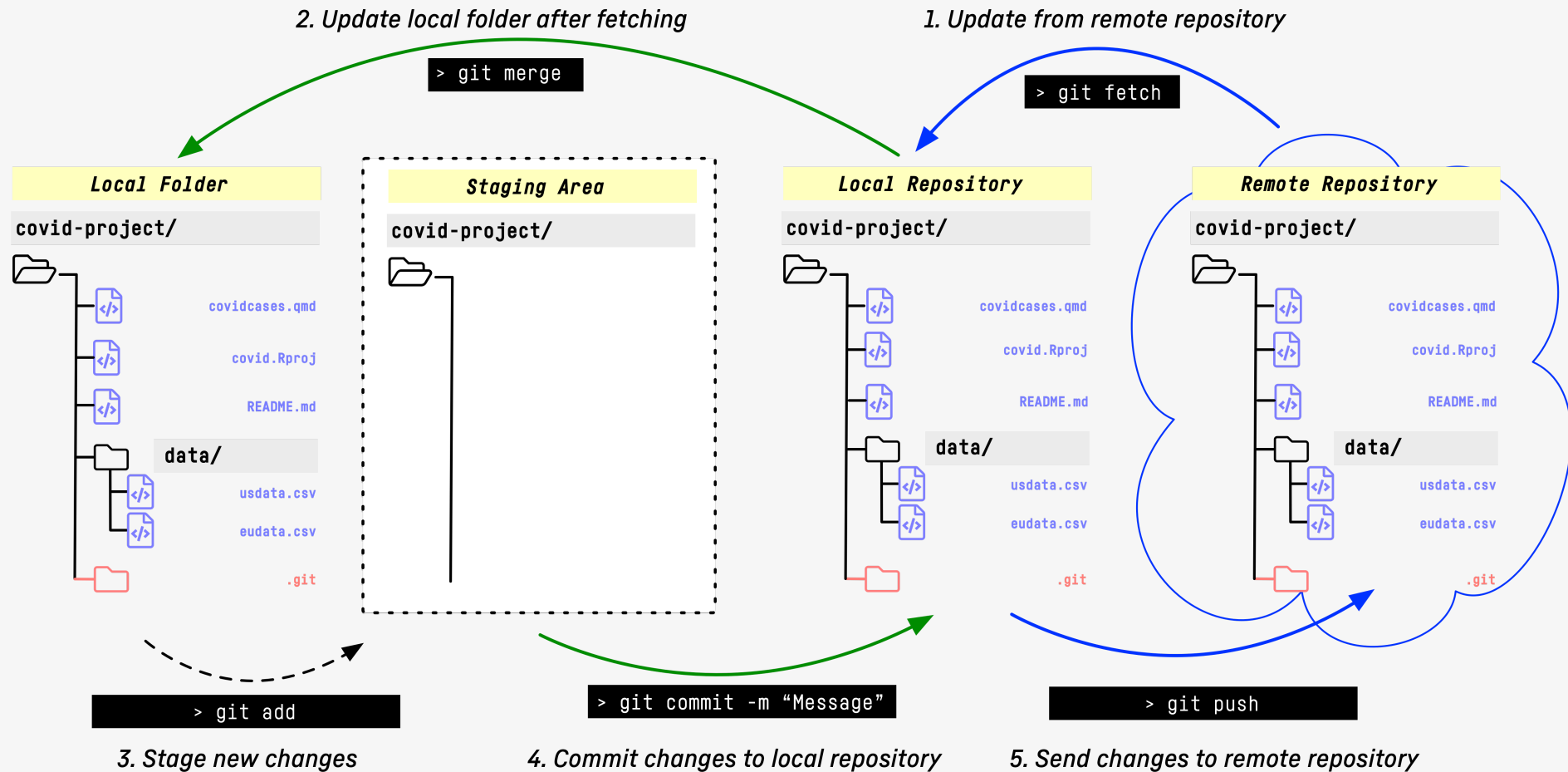
Workflow: GitHub Round Trip



Workflow: GitHub Round Trip

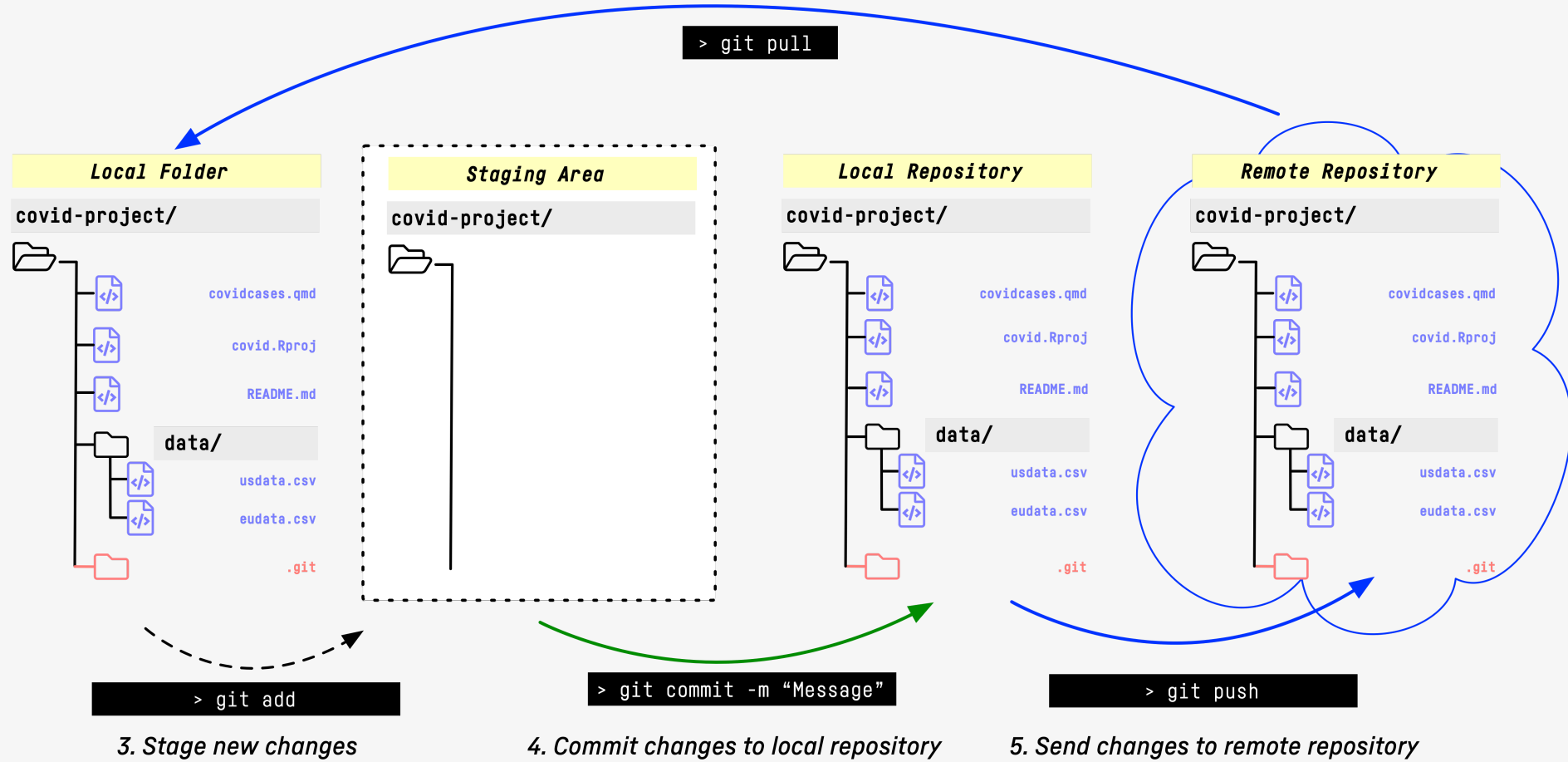


Workflow: GitHub Round Trip



git pull

1 & 2 at once: Update from remote repository and immediately merge any changes



Cloning

Cloning a Repo

What if you want to work with someone else's project?

The screenshot shows the GitHub interface for the 'covid-project' repository by user 'kjhealy'. The repository is public and has 1 branch (main), 0 tags, and 6 commits. The file list includes 'data' (Data and Rproj files), 'README.md' (Began writeup), 'covid.Rproj' (Data and Rproj files), and 'covidcases.qmd' (Load the US data). The README section is titled 'Sample project' and contains a list of bullet points: 'Used for teaching Git', 'Sources of files in data/ are proximate from the covdata package and originally from the New York Times state-level Covid tracker (for usdata.csv) and the European Centers for Disease Control Covid tracker (for eudata.csv).', and 'Do not use this file for any real covid analysis ...'. The right sidebar shows the 'About' section with no description, website, or topics provided, and the 'Releases' and 'Packages' sections with no published items.

covid-project Public

main 2 Branches 0 Tags

Go to file Add file Code

kjhealy Load the US data 06677f4 · last year 6 Commits

data	Data and Rproj files	last year
README.md	Began writeup	last year
covid.Rproj	Data and Rproj files	last year
covidcases.qmd	Load the US data	last year

README

Sample project

- Used for teaching Git
- Sources of files in `data/` are proximate from the `covdata` package and originally from the New York Times state-level Covid tracker (for `usdata.csv`) and the European Centers for Disease Control Covid tracker (for `eudata.csv`).
- Do not use this file for any real covid analysis ...

About

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks

Releases

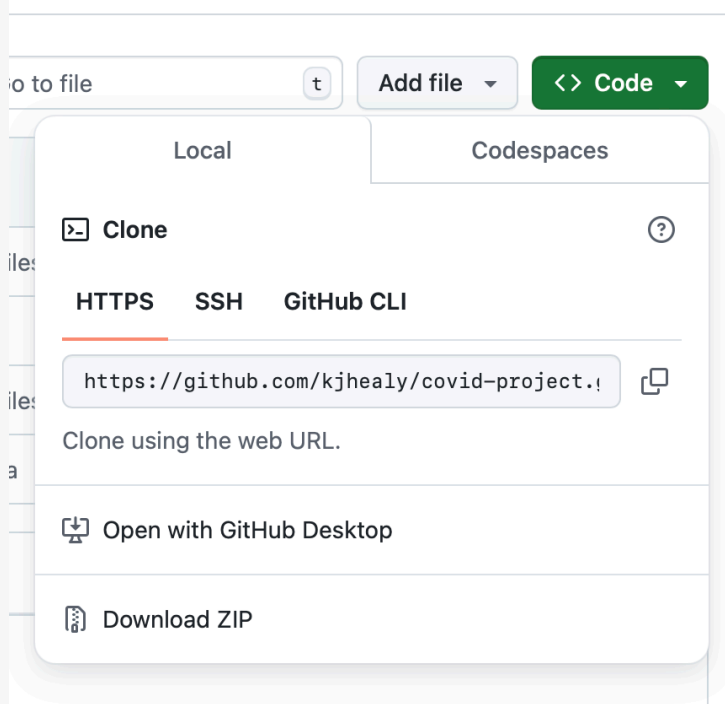
No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Toy Covid Project on GitHub

Cloning a Repo



You can use git or a git client to clone via
HTTPS or SSH

HTTPS + a PAT token is easiest

There's also the `gh` command line tool, which
is for using `git` specifically with GitHub

Cloning a Repo

`git clone` puts the repo in your current folder.

```
> cd Documents/data/
> git clone https://github.com/kjhealy/covid-project.git
Cloning into 'covid-project' ...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 23 (delta 8), reused 22 (delta 7), pack-reused 0 (from 0)
Receiving objects: 100% (23/23), 577.21 KiB | 5.77 MiB/s, done.
Resolving deltas: 100% (8/8), done.

> cd covid-project/
> ls
README.md      covid.Rproj    covidcases.qmd data
```

Cloning a Repo

Now you have access to the full history and any branches, etc.

```
› git log --oneline
06677f4 (HEAD → main, origin/main, origin/HEAD) Load the US data
4a77b2f (origin/figtest) Fixed the plot
912aed7 Checking ggplot code
6df6d6b Began writeup
ef07721 Data and Rproj files
f43f6f1 Added covidcases
```

Checking Changes

Use `git diff` to compare your working folder with the most recent commit (the default), or to compare changes between two commits.

```
idlewild ~/D/covid-project git:main > git diff 6df6d6b 912aed7
```

```
Δ covidcases.qmd
```

```
• 10: library(tidyverse)
```

```
10  ```\n11\n12  ## Read in the data
```

```
10  ```\n11\n12  ## Read in the data\n13\n14\n15  ## Figtest\n16\n17  ```{r}\n18  mtcars |>\n19    ggplot(mapping = aes(x = hp,\n20                        y = mpg)) +\n21    geom_point() +\n22    geom_smooth()\n23  ```\n24
```

```
idlewild ~/D/covid-project git:main > 
```

Project State

Here's the state of this course's project as I write this:

```
idlewild ~/D/c/mptc git:main › git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file> ..." to update what will be committed)
  (use "git restore <file> ..." to discard changes in working directory)
    modified:   slides/05-slides.qmd
    modified:   slides/06-slides.qmd

Untracked files:
  (use "git add <file> ..." to include in what will be committed)
    assets/04-git/04-github-clone.png
    assets/04-git/04-github-covidproject.png
    data/dfstrat.csv

no changes added to commit (use "git add" and/or "git commit -a")
```

Branches

Our project again

```
> git log
commit 6df6d6b10b127e87115dceac06c869150b67e800 (HEAD → main, origin/main)
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 17:49:33 2023 -0400
```

Began writeup

```
commit ef0772106fe4d6f10ee62afd3c951ff14b16efb1
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 17:38:55 2023 -0400
```

Data and Rproj files

```
commit f43f6f10ba5cd1a57430c032c8c14f056b154e9b
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 17:34:08 2023 -0400
```

Added covidcases

Our project again

Individual commits can be referenced by their **SHA** hash.

Usually the first few bits are sufficient to ID the commit in a single project.

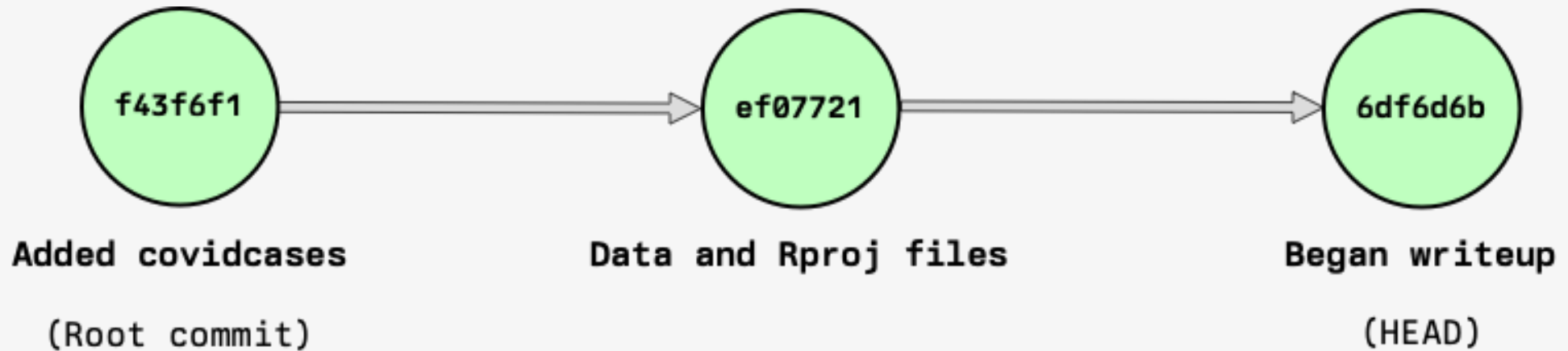
```
> git log --oneline  
  
6df6d6b (HEAD → main, origin/main) Began writeup  
ef07721 Data and Rproj files  
f43f6f1 Added covidcases
```

The current commit is called the **HEAD**

Our project again

The SHA or commit id always refers to the same code.

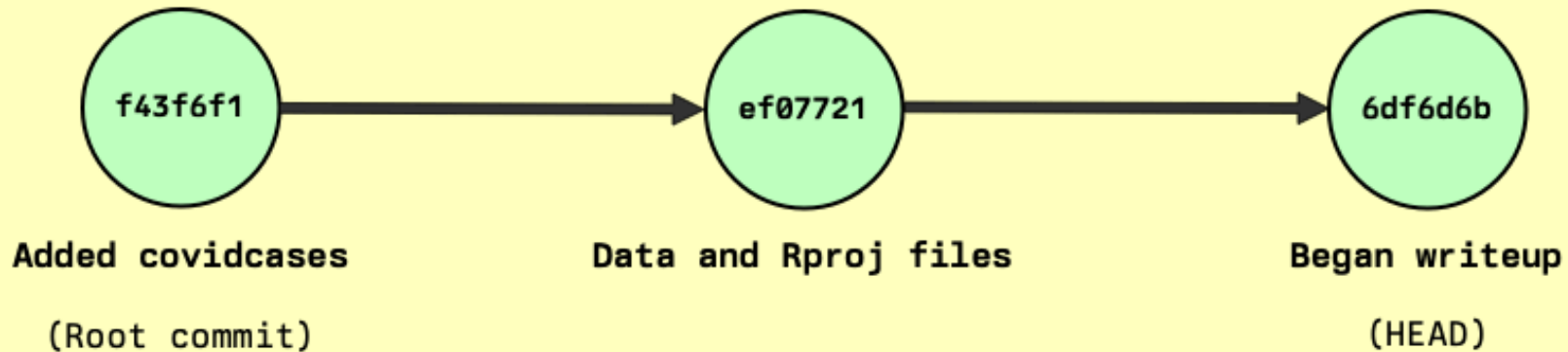
Zooming out from the various files changed in each case to the level of commits, our project looks like this:



We're on a branch

By default we're on the `main` branch

main Branch



We can create branches

```
> git branch figtest  
> git checkout figtest
```

```
Switched to branch 'figtest'
```

We make some edits in covidcases.qmd

```
mtcars ▷  
  ggplot(mapping = aes(x = hp,  
                        y = mpg)) +  
  geom_point() +  
  geom_smooth()
```

What's our status?

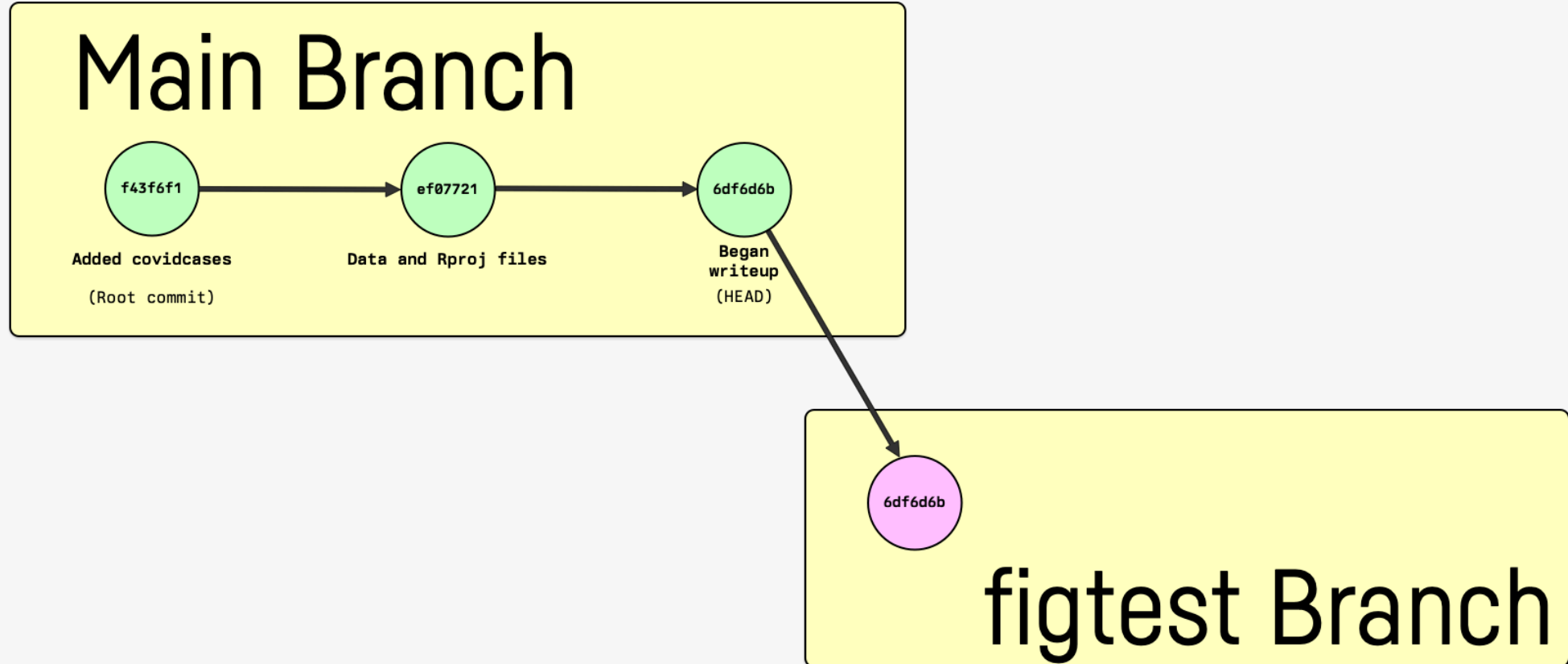
```
# Back at the shell here
> git status
On branch figtest
Changes not staged for commit:
  (use "git add <file> ..." to update what will be committed)
  (use "git restore <file> ..." to discard changes in working directory)
    modified:   covidcases.qmd

no changes added to commit (use "git add" and/or "git commit -a")
```

Commit changes on the new branch

```
> git add covidcases.qmd  
> git commit -m "Checking ggplot code"  
  
[figtest 912aed7] Checking ggplot code  
1 file changed, 12 insertions(+)
```

Schematically



Look at the whole log

```
> git log --abbrev-commit  
commit 912aed7 (HEAD → figtest)  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 20:25:42 2023 -0400
```

Checking ggplot code

```
commit 6df6d6b (origin/main, main)  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 17:49:33 2023 -0400
```

Began writeup

```
commit ef07721  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 17:38:55 2023 -0400
```

Data and Rproj files

```
commit f43f6f1  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 17:34:08 2023 -0400
```

Added covidcases

Status?

```
git status  
On branch figtest  
nothing to commit, working tree clean
```

Make some more changes ...

```
> git log --abbrev-commit

commit 4a77b2f (HEAD → figtest)
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 20:33:57 2023 -0400

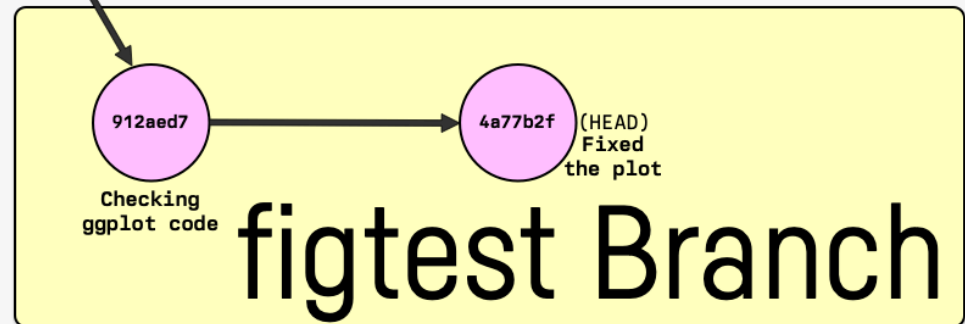
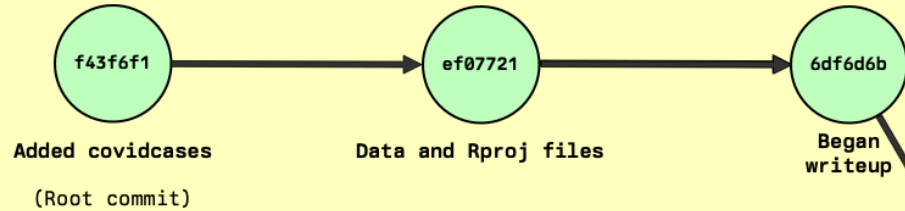
    Fixed the plot

commit 912aed7
Author: Kieran Healy <kjhealy@gmail.com>
Date:   Mon Sep 18 20:25:42 2023 -0400

    Checking ggplot code
```

Schematically

main Branch



Merge it back in to **main**

```
> git checkout main  
Switched to branch 'main'
```

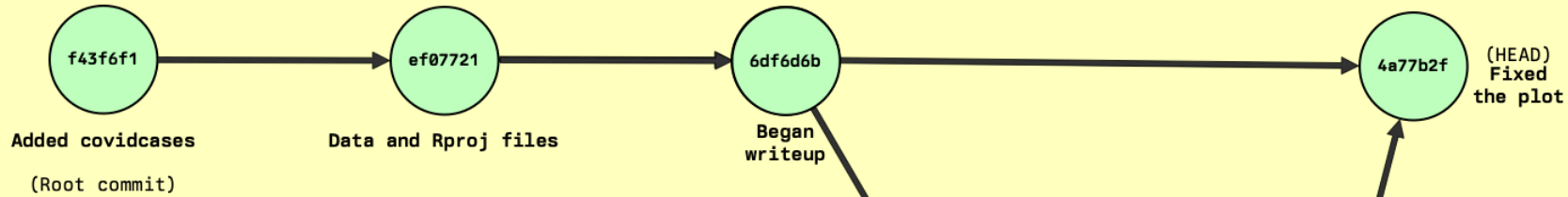
None of the code we wrote in the last two commits is on this branch.

```
> git merge figtest  
  
Updating 6df6d6b..4a77b2f  
Fast-forward  
 covidcases.qmd | 13 ++++++++  
 1 file changed, 13 insertions(+)
```

And now it is.

Schematically

main Branch



figtest Branch

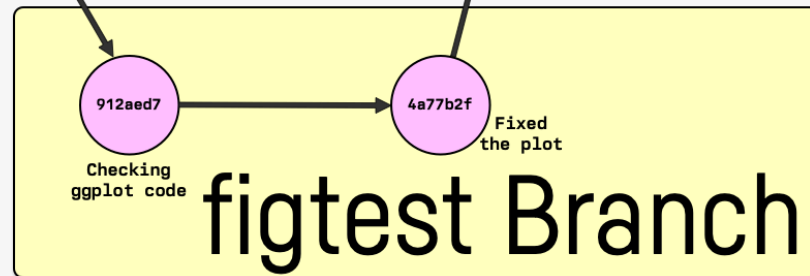
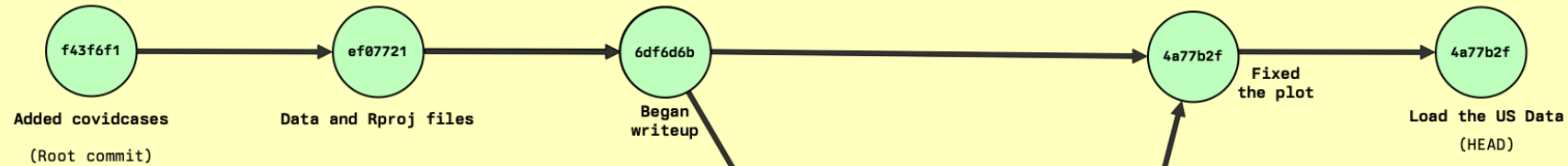
Write some more

Make some more edits and commit those changes.

```
## Read in the data  
df ← read_csv(here::here("data", "uscases.csv"))
```

Schematically

main Branch



Where are we?

```
> git log --abbrev-commit
```

```
commit 06677f4 (HEAD → main, origin/main)  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 20:45:10 2023 -0400
```

Load the US data

```
commit 4a77b2f (origin/figtest, figtest)  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 20:33:57 2023 -0400
```

Fixed the plot

```
commit 912aed7  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 20:25:42 2023 -0400
```

Checking ggplot code

```
commit 6df6d6b  
Author: Kieran Healy <kjhealy@gmail.com>  
Date:   Mon Sep 18 17:49:33 2023 -0400
```

Summary

It's a log book

For single person projects you can go a *long* way with a very basic git workflow.

Essentially do your work then `add`, `commit`, and `push` the results.

At a minimum you will be left with a record of your project's development where you can revisit *any* commit and see what the state of the project was at that point.

It's a testbed

A little discipline with branching will let you experiment with new lines of work or code without getting lost or losing the ability to revert to what was there before, all in the same project folder.

Successful lines of inquiry can be merged into the main branch, unsuccessful ones kept for the record or discarded altogether.

It's a collaboration tool

The combination of multiple copies of the repository on the one hand and the act of branching and merging on the other generalizes to working with multiple contributors to a repository and managing all of their work.

Again, Git requires a bit of discipline about this—a team should have rules about branch naming, task prioritization, bug fix procedures, etc.

Services like GitHub provide most of their value by offering ways to manage these tasks in the context of a git-based way of working.

It's often a pain

Underneath Git is kind of a mess.

Various design features make it easy to be confused or misled.

Start with the Sylor-Miller & Evans zine.