INSTITUTE FOR DEFENSE ANALYSES

Introduction to Git

Dr. John T. Haman, Project Leader

Dr. Curtis G. Miller

April 2022

IDA Document NS D-33021

Log: H 2022-000108

INSTITUTE FOR DEFENSE ANALYSES
730 East Glebe Road
Alexandria, Virginia 22305

The Institute for Defense Analyses is a nonprofit corporation that operates three Federally Funded Research and Development Centers. Its mission is to answer the most challenging U.S. security and science policy questions with objective analysis, leveraging extraordinary scientific, technical, and analytic expertise.

About This Publication
This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-19-D-0001, Task BD-09-229990, "TestSci App," for the Office of the Director, Operational Test and Evaluation. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

Acknowledgments
The IDA Technical Review Committee was chaired by Dr. V. Bram Lillard and consisted of Dr. Matthew R. Avery, Dr. Juan D. Remolina Gonzalez, and Dr. Thomas H. Johnson from the Operational Evaluation Division.

For more information:
Dr. John T. Haman, Project Leader
jhaman@ida.org • (703) 845-2132

V. Bram Lillard, Director, Operational Evaluation Division
vlillard@ida.org • (703) 845-2230

Rigorous Analysis | Trusted Expertise | Service to the Nation

INSTITUTE FOR DEFENSE ANALYSES

IDA Document NS D-33021

Introduction to Git

Dr. John T. Haman, Project Leader

Dr. Curtis G. Miller

Executive Summary

Version control software manages, archives, and (optionally) distributes different versions of files. Git is the most popular program for version control and serves as the backbone for websites such as Github, Bitbucket, and others.

In this mini-tutorial we will introduce basics of version control in general, and Git in particular. We focus on two aspects of version control: maintaining a history of a project, and maintaining multiple simultaneous versions. Effective history management keeps a project clean and means that previous versions can be stored should they prove useful while formalizing the history-management process. Maintaining simultaneous file versions facilitates collaboration and the creation of similar products.

Participants will be able to get started using Git right away. The tutorial covers the basics, including creating a repository, storing it externally on GitHub, tracking changes via commits, sharing those changes, getting others' changes, recovering old versions of a project, branching a project into multiple (simultaneous) versions, and merging versions together.

We show how to use Git from the command line and also how online tools provided by GitHub can help with the process and understanding a repository's features. We also describe some Git and reproducible research best practices and demonstrate them through a notional project.

We also explain what role Git plays in a reproducible research context. Git itself will not make research reproducible, but by systematically tracking the history of a project, Git shows the relationship between chosen analytical methods, data used, and the results and conclusions. All of these factors often change as a project evolves over time, and Git tracks that evolution.

Introduction to Git

Curtis Miller

April 27, 2022

Institute for Defense Analyses
730 East Glebe Road ● Alexandria, Virginia 22305

Introduction

We will see basic Git usage and concepts

- Getting Started
- Repos
- Tracking Data
- Branches
- Conclusion

Git is a popular and powerful open-source, version control software system

- Version control software manages changes in files as files are edited over time and by different users
- With Git, users can track changes made to files and back up old versions of files
- When multiple variants of a product need to be managed or different people make changes, Git tracks the differences across branches

Getting Started

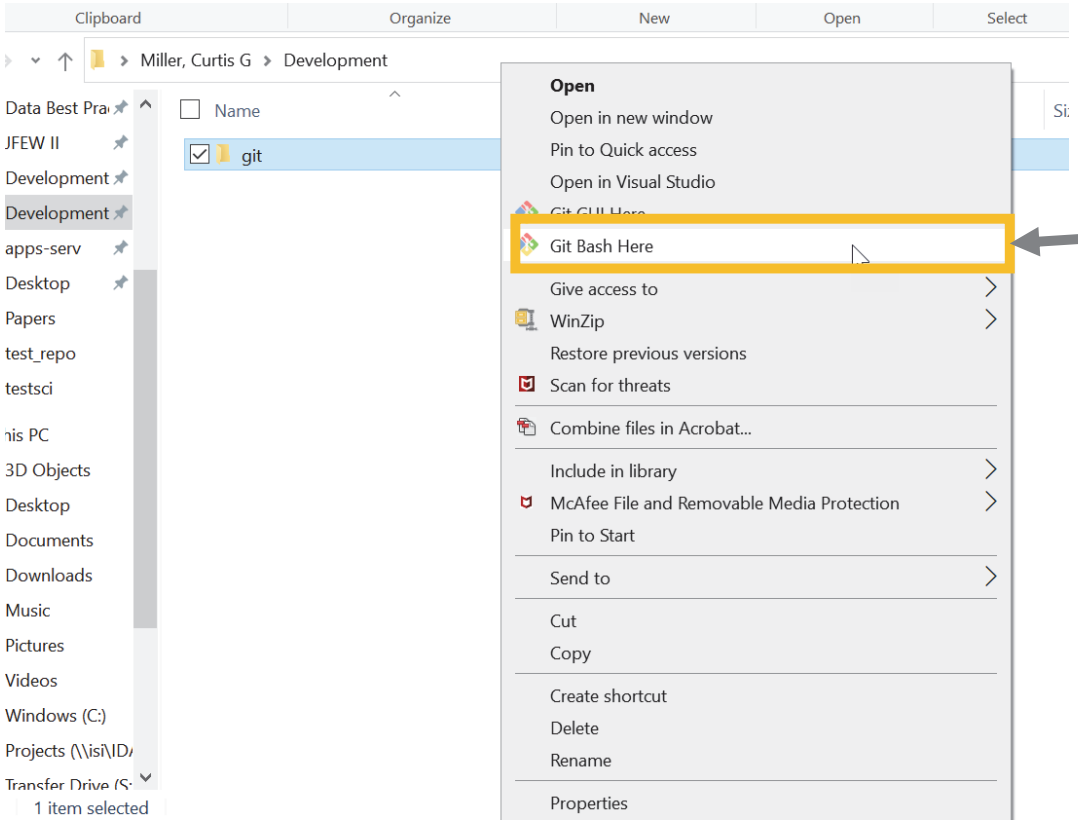
Git is free, open source, and the de facto version control system

- Visit http://git-scm.com/ to download a copy of Git for your operating system (or use your operating system's package manager, if available)
- Follow the instructions of the installer to get set up
- We will be working in a Windows environment for this tutorial

I'm going to assume you have Git installed now.

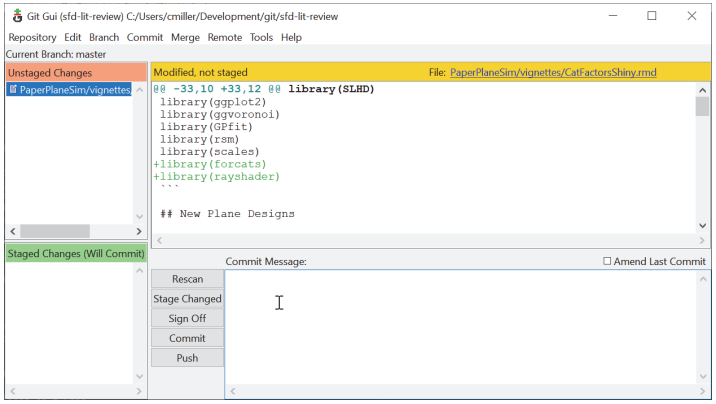
GPL: General Public License

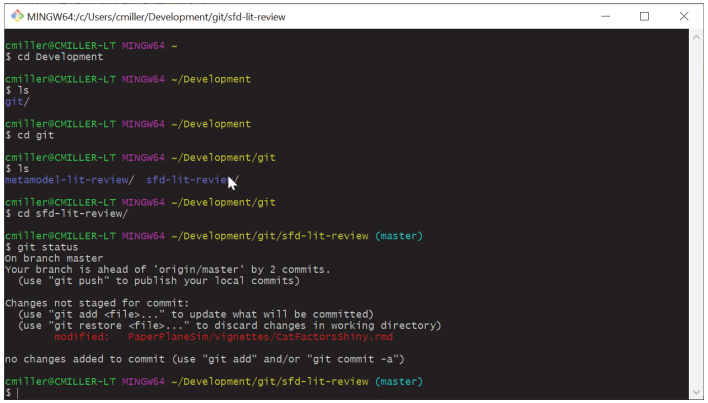
You can open a Bash terminal to issue Git commands via a right-click on a directory on Windows



On Windows: click the folder, then right-click for this menu, then click Git Bash for a Bash terminal, or Git GUI for a graphical interface

Git on Windows provides a GUI and a Bash command line

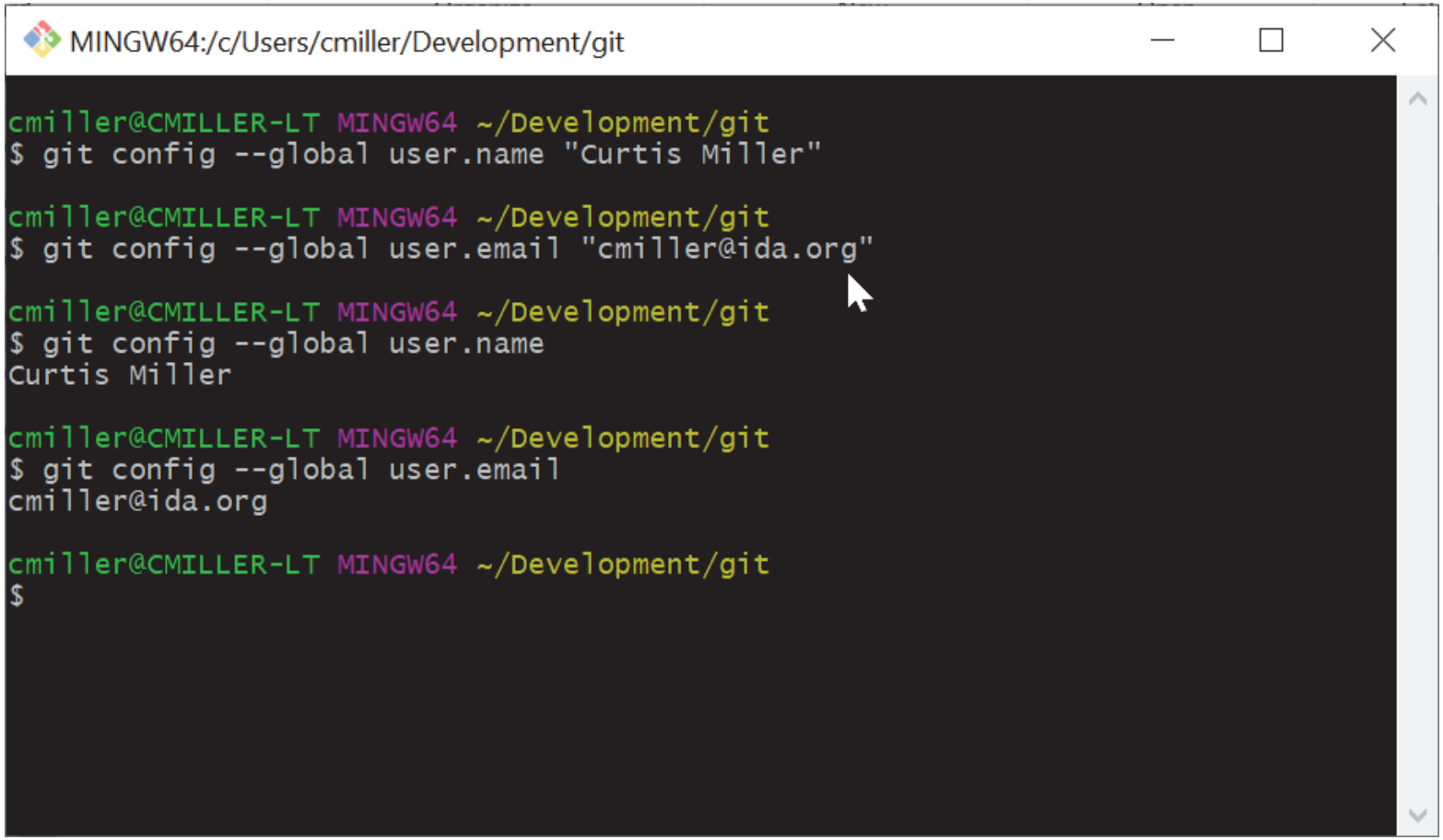




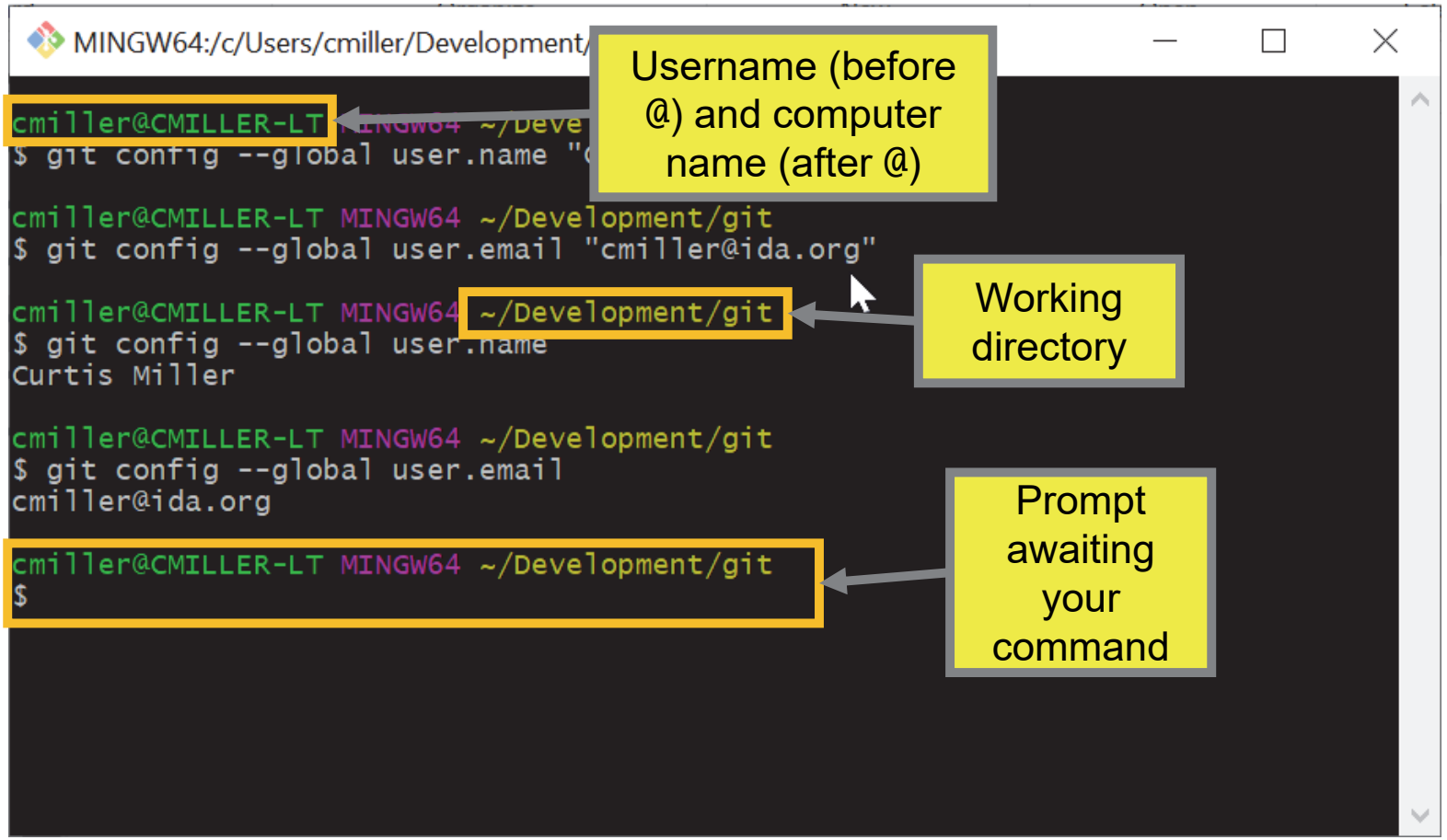
- The Windows Git installer provides both a GUI and a Bash terminal emulator for using Git
- While the GUI may seem beginner friendly, it serves mostly as a front-end to the command line interface
- Full utilization of Git's power is easiest when using the command line
- Git concepts are easier to explain when using the command line

GUI: Graphical User Interface

Set up initial user parameters using `git-config`



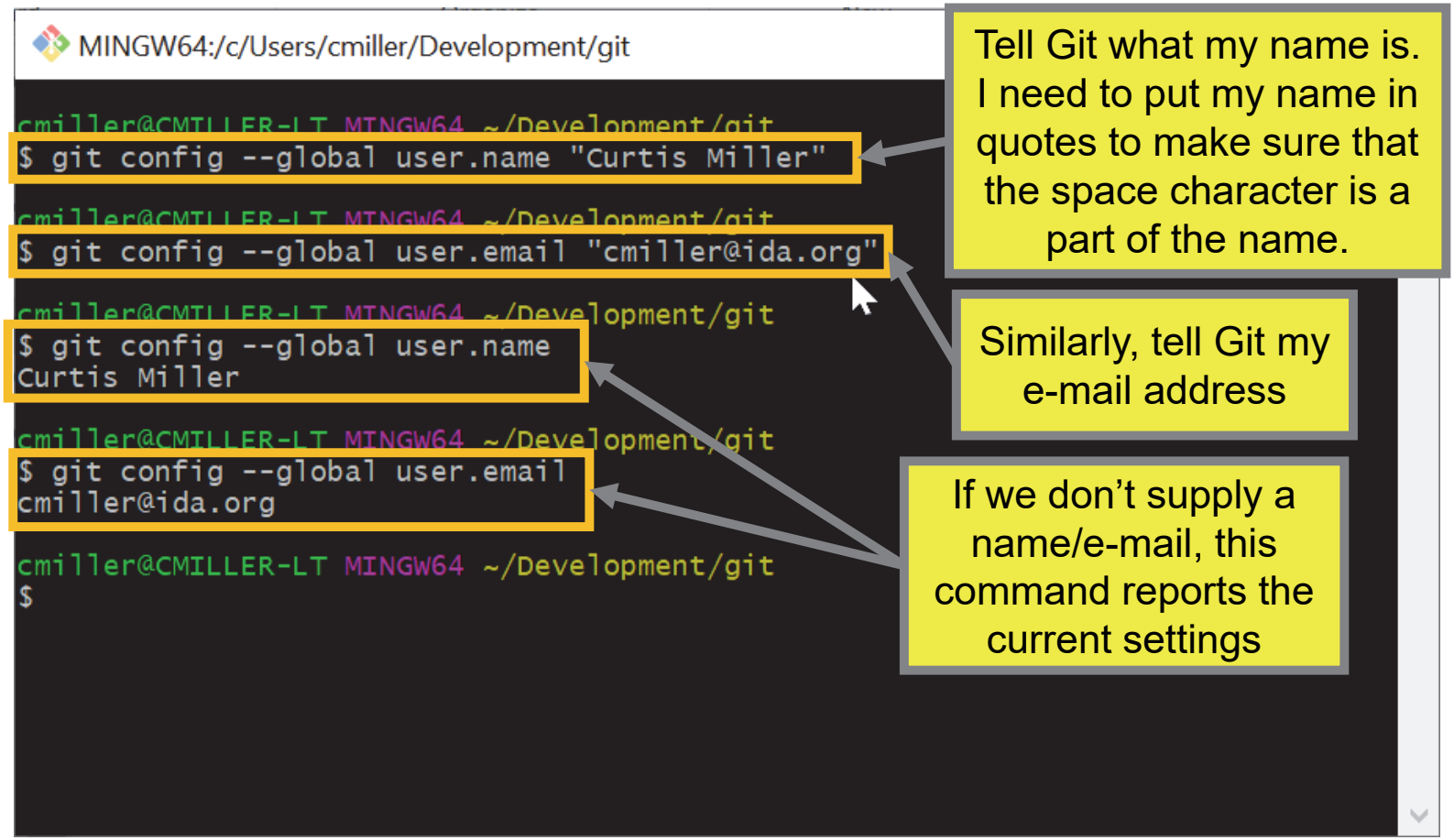
We can determine a lot from a Bash prompt



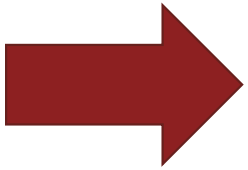
Bash is *not* Git; it is a shell (like PowerShell on Windows) and the default shell of Linux. We are talking about Git today, not Bash, though you may learn a little about Bash in the process.

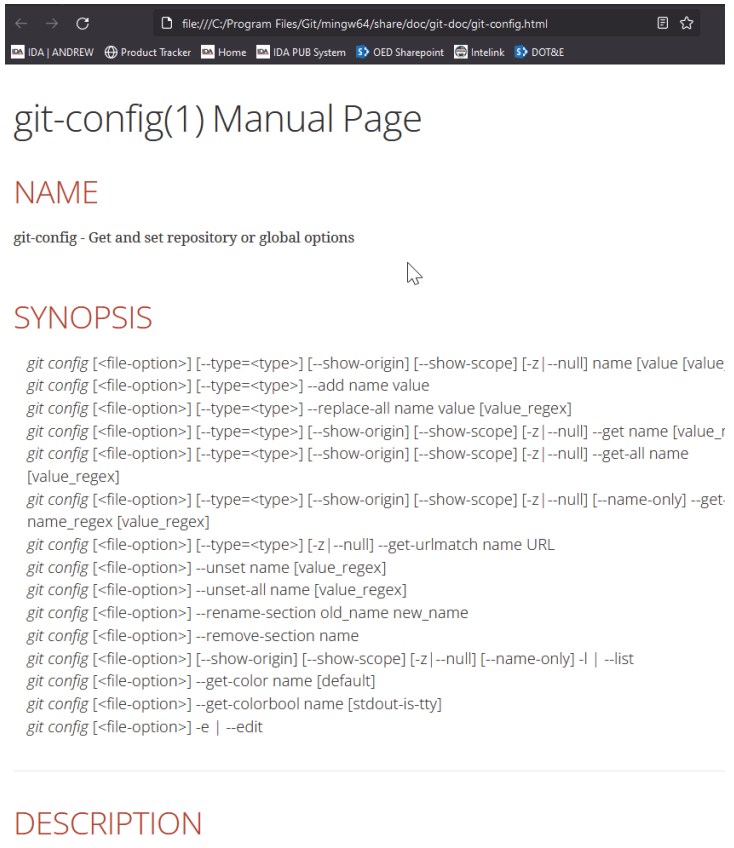
We are using Git via Bash.

We can configure Git to know who we are; this information will be useful when others look at the Git history and see your work



“How interesting! How can I learn more about how this command works?”



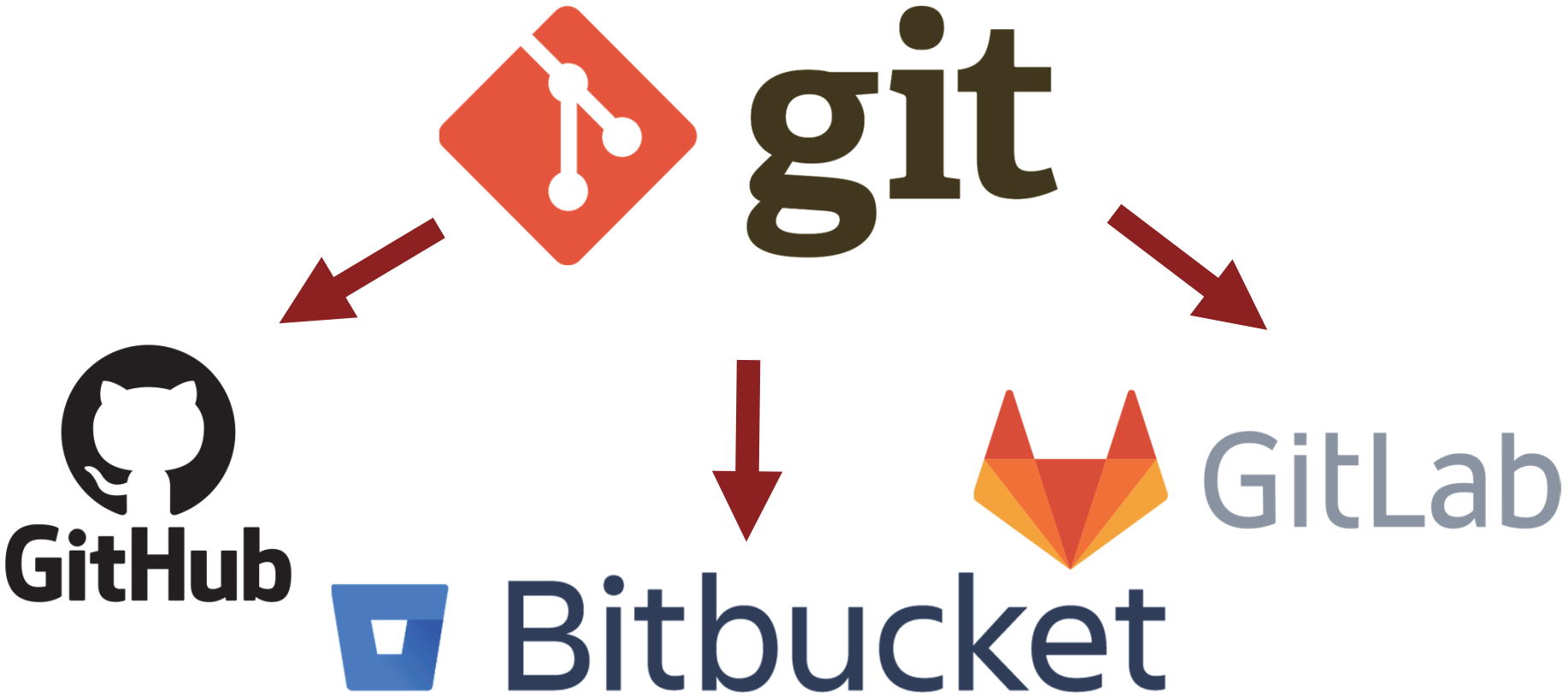


“How interesting! How can I learn more about how this command works?”

Every Git command has a manual page accessible via the --help option

On Windows, the manual page will be opened in a browser window; try to look for examples for what you want to do

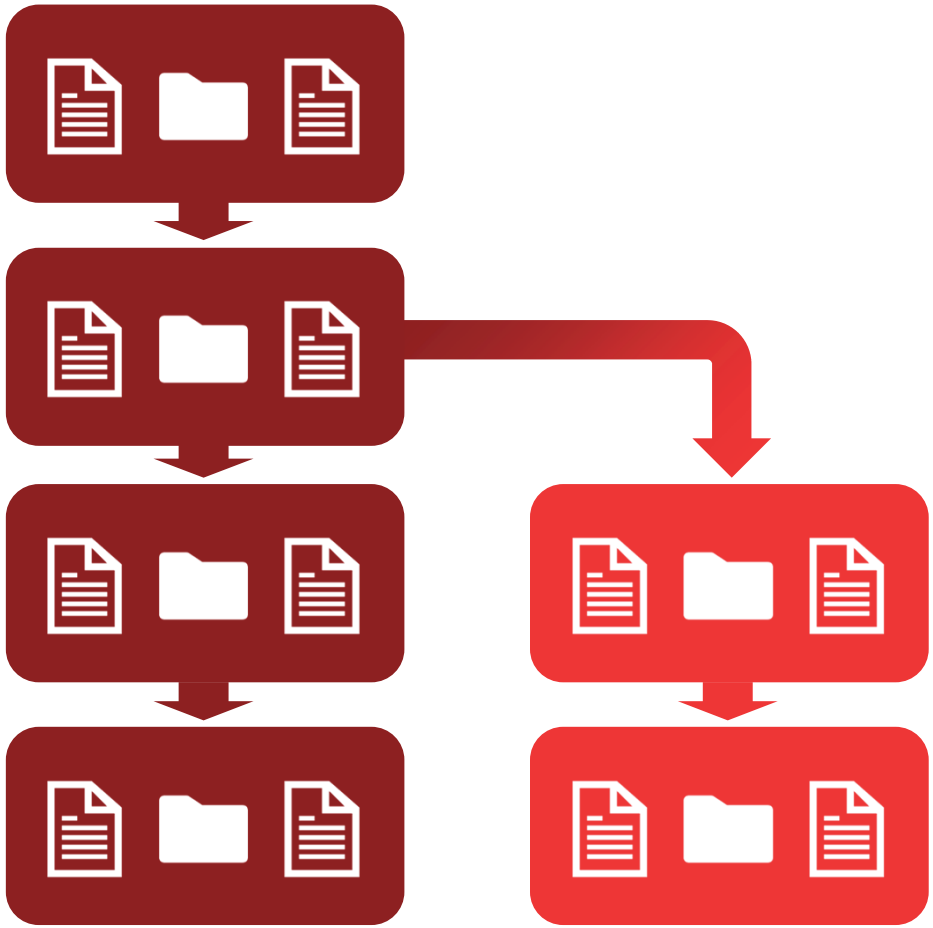
Git powers popular code sharing websites such as GitHub



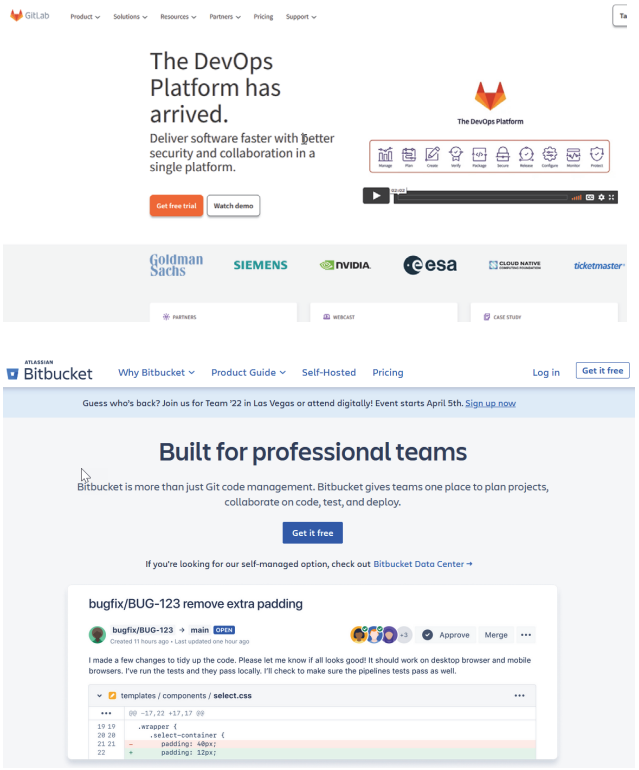
Repos

A Git repository is a collection of tracked files

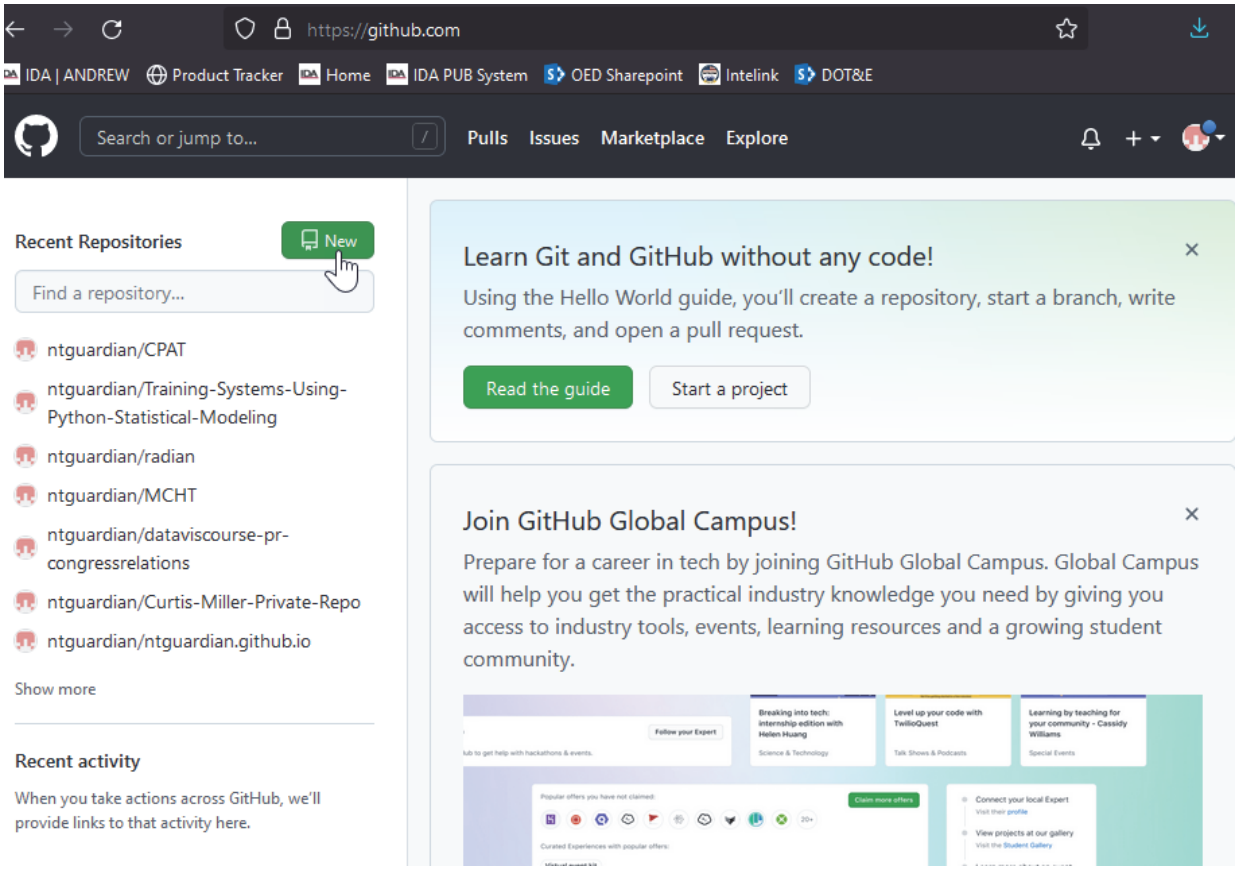
- Git tracks the history of these files and their differences across time and between branches
- Repos generally contain entire projects
- Users access the project by accessing the repo, and the project is shared by sharing the repo
- Repos are as simple as folders containing files



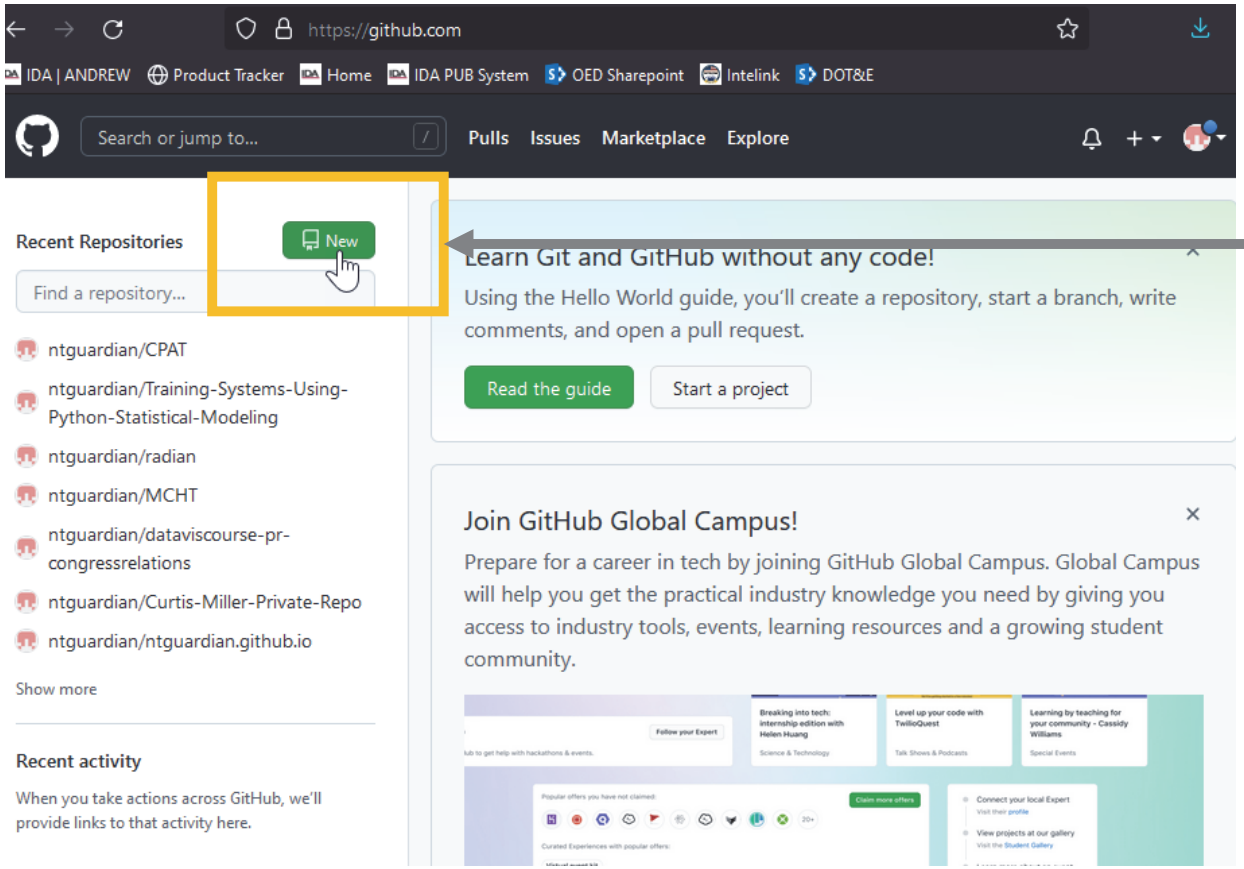
Many web services host repos (but you don't need to use these to use Git on your own computer)



We can create a repo on GitHub and then clone it to a personal computer

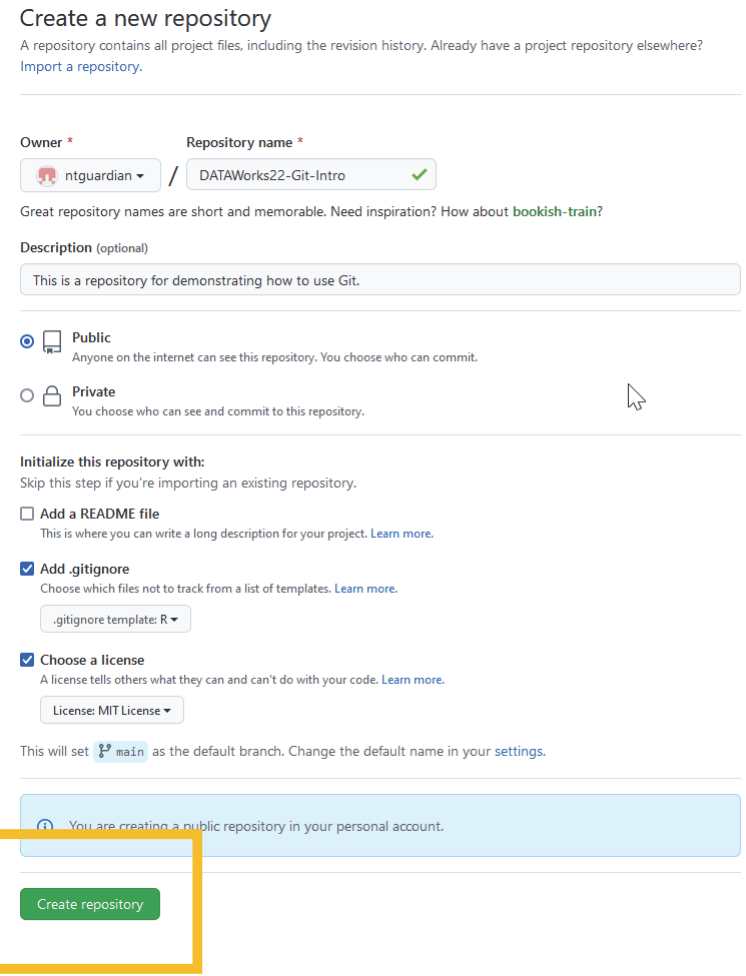


We can create a repo on GitHub and then clone it to a personal computer



Click me to make a new repo on GitHub

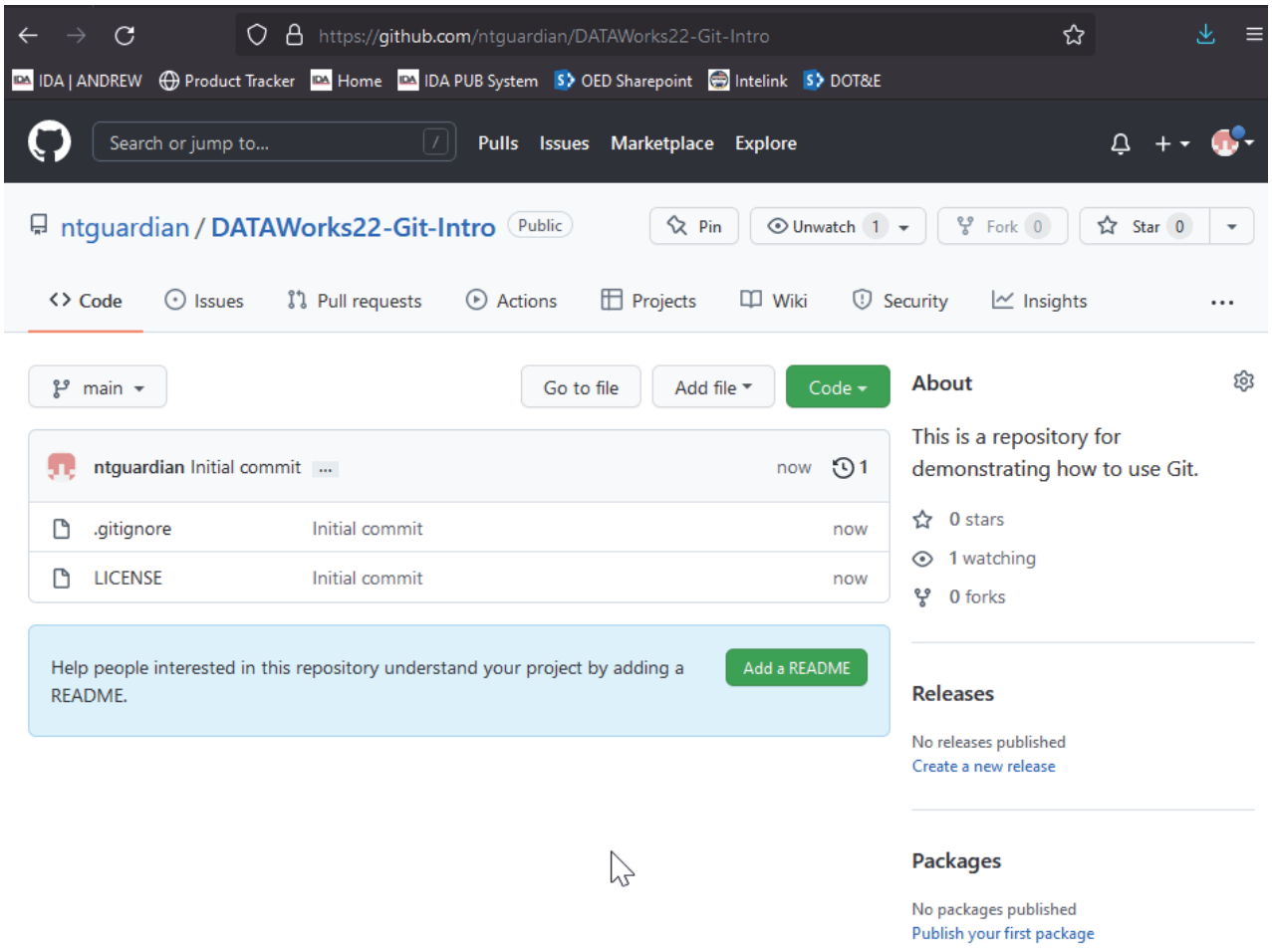
Fill out the form to create a new repository



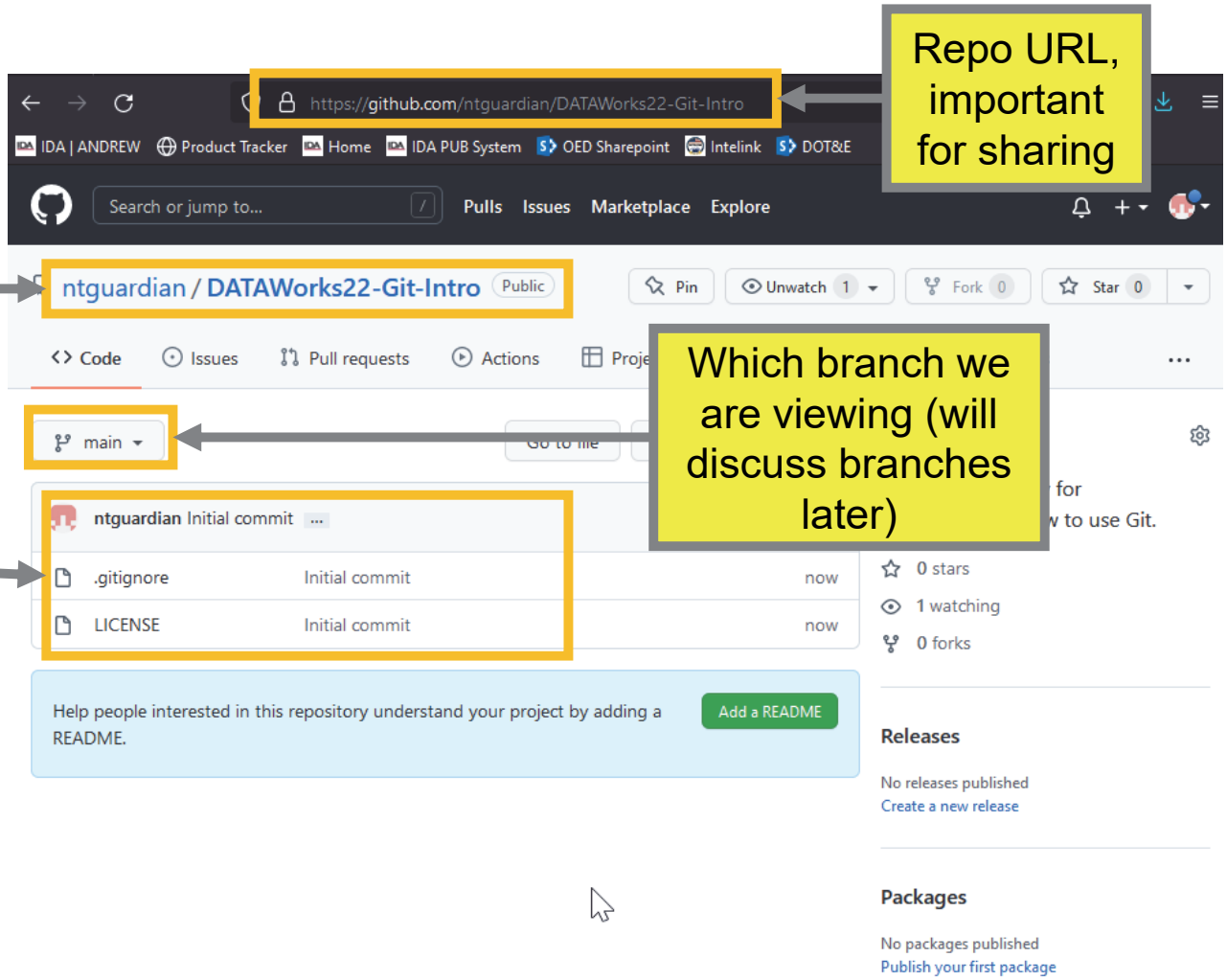
Click me to make a new repo on GitHub

For real work, follow your organization guidelines on where to store data. For example, IDA employees need to use the IDA internal BitBucket for repo hosting, not GitHub (for sponsor work).

Now we can view our repo online



Now we can view our repo online



Repo URL, important for sharing

Repo identifier (note connection to URL)

Which branch we are viewing (will discuss branches later)

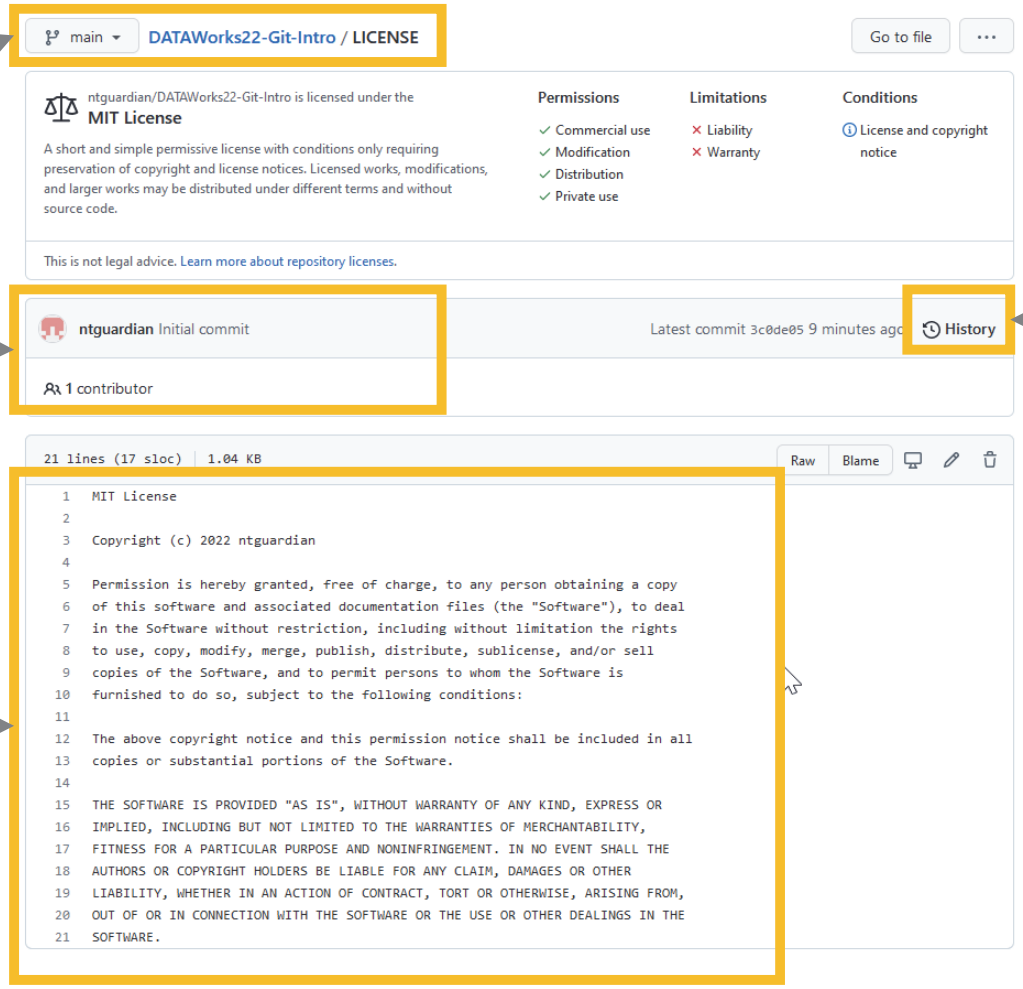
Some files already in the repo, created by GitHub (click to view)

We can view files online

The file we are viewing, including what branch

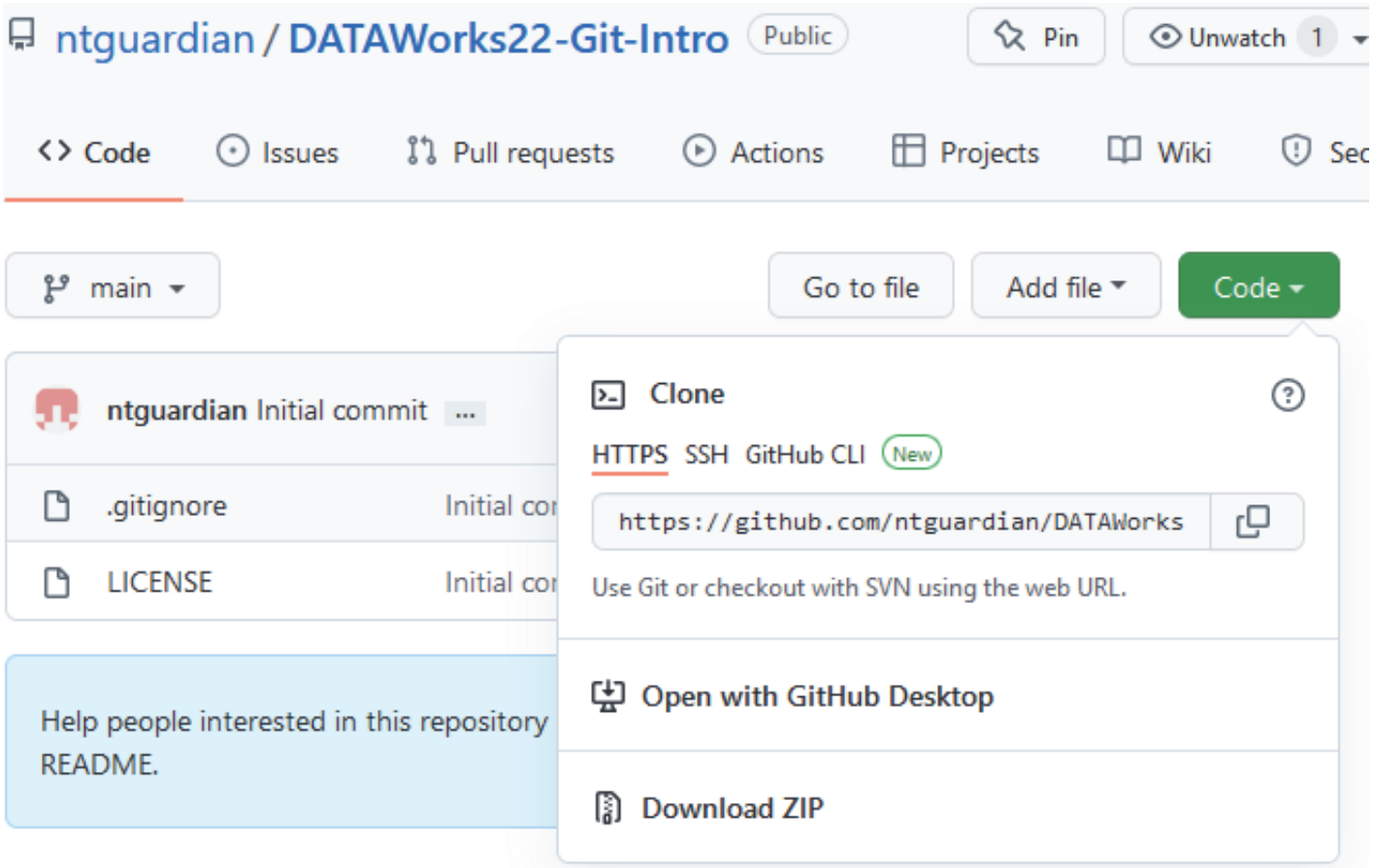
Who works on the file and recent activity

File contents

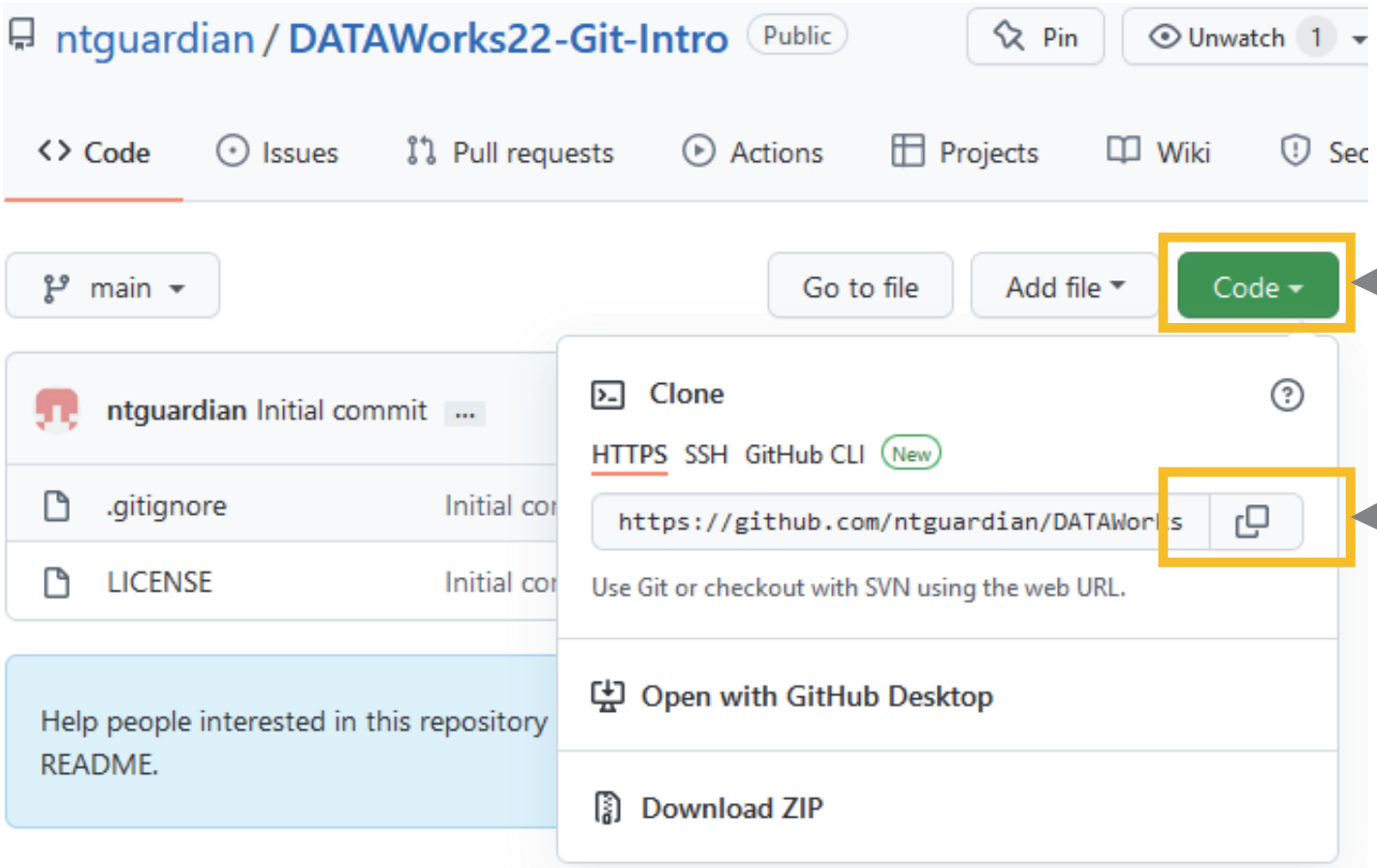


Click this to see the file's commit history (will discuss commits soon)

To get this repo onto our computer, we need to clone it



To get this repo onto our computer, we need to clone it

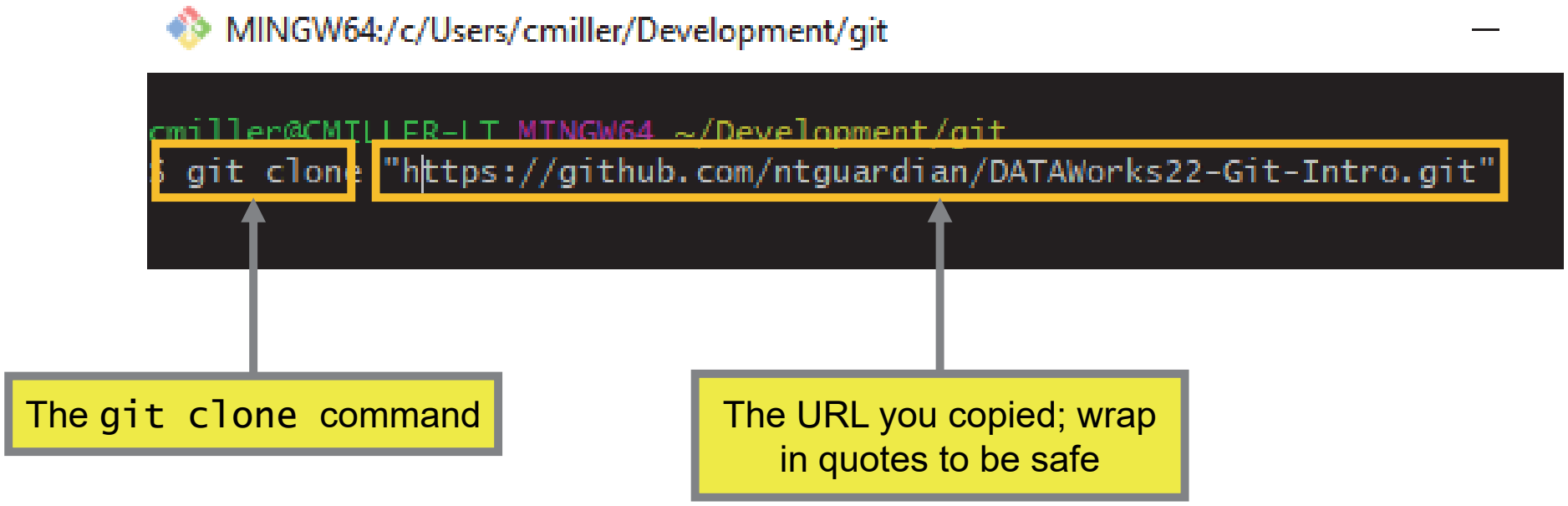


Click this to open a dialog to allow easy cloning

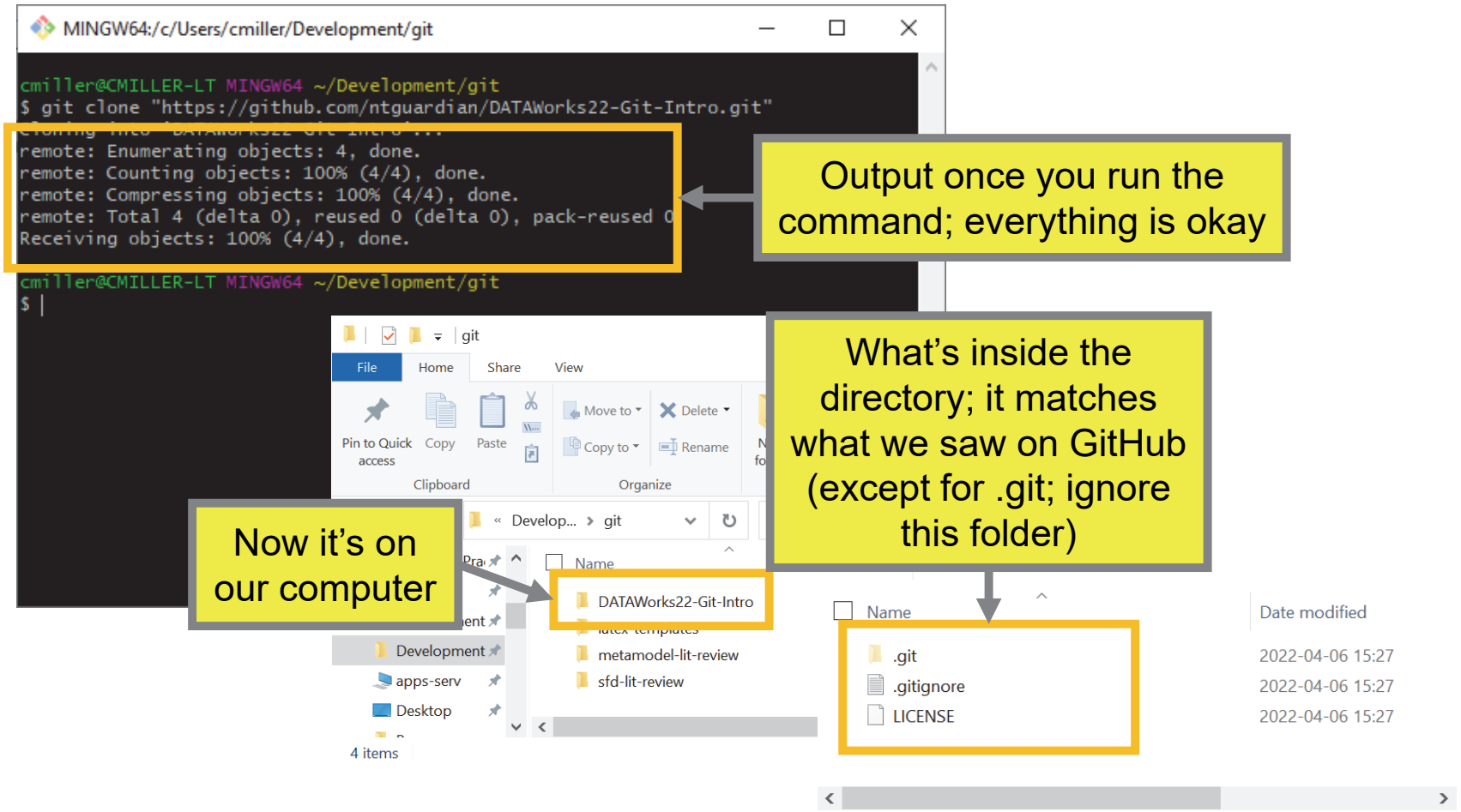
Click to copy the repo URL for cloning

Use the command git clone to get a copy on your computer

Use the command git clone to get a copy on your computer



Once you run the command, you have a working copy of the repo on your computer



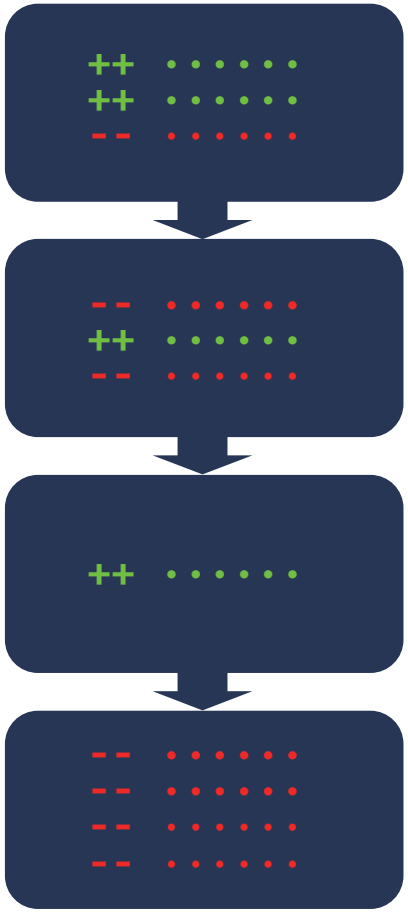
Summary: Making and getting repos

Command	Description	Examples
`git init`	Create a new repo	`git init`
`git clone`	Clone an existing repository	`git clone “https://github.com/someuser/someproject”`

Tracking Data

Git tracks changes via commits

- A Git commit is the basic unit by which Git tracks changes to tracked files
- The Git user adds files for Git to track and tells Git when to track changes made to a file
- For text files, Git tracks only the difference between the current file and the most recent version
- Users can “check out” previous commits to recover the (tracked) state of the project at the time of the commit



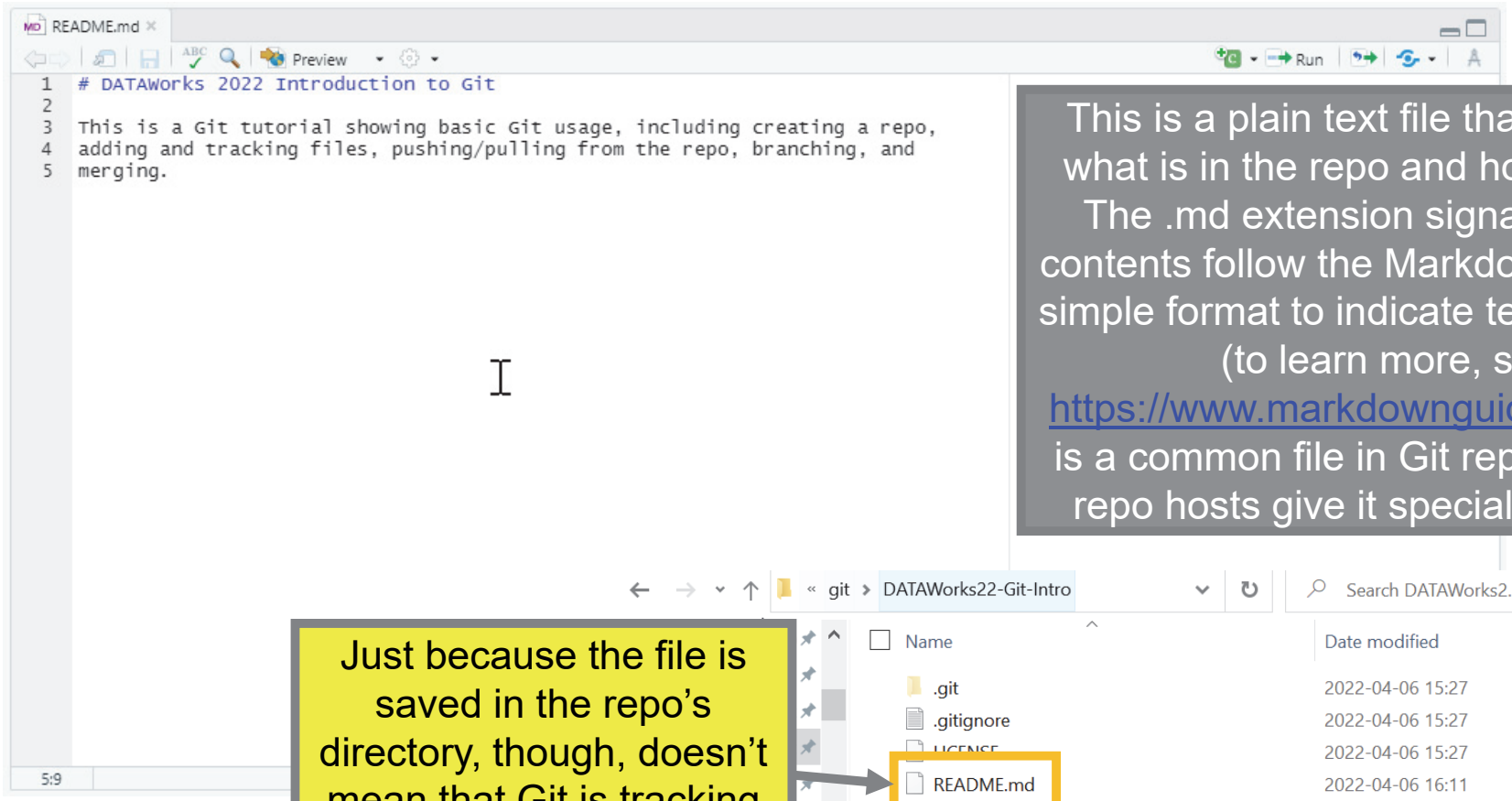
Using Git is like belaying* while mountain climbing; if you fall, you fall three feet, not three hundred feet

- With Git, you can track changes as they are made
- If you make an immediate mistake, it can be undone
- If you made a mistake several weeks ago that is not obvious until now, Git can help find where the error happened and help address it
- If a deleted file turns out to be useful, you can retrieve it
- You can branch to try out an idea without losing the original files known to work
- You can easily manage multiple versions of a product simultaneously

Wikimedia Commons

*Belaying is climbing a rock wall with a rope. The climber is attached to a rope that is either held by someone at the top of the cliff or looped around something sturdy while a partner stands on the ground with the other end of the rope. As the climber progresses, their partner keeps the rope taut; should the climber let go of the wall, they will only fall as far as the slack in the rope allows, which keeps the climber safe and ensures not too much progress is lost.

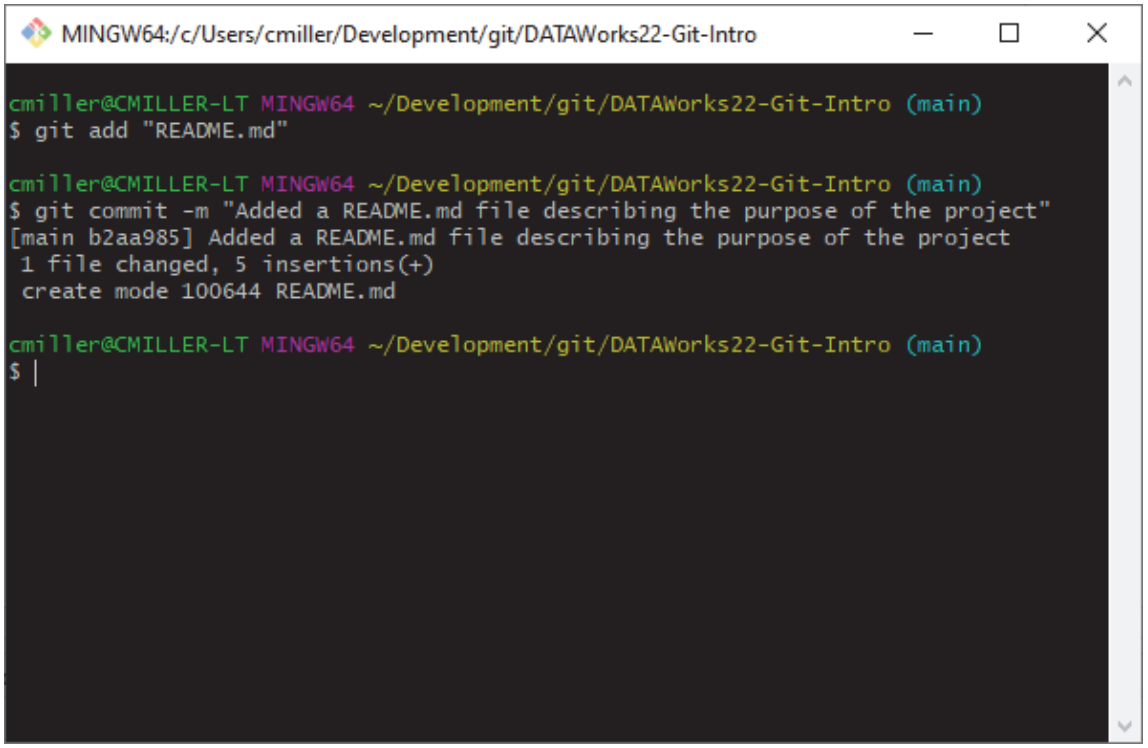
Our Git repo should have a README.md file; let's make one!



This is a plain text file that describes what is in the repo and how to use it. The .md extension signals that the contents follow the Markdown format, a simple format to indicate text formattin (to learn more, see https://www.markdownguide.org). This is a common file in Git repos so some repo hosts give it special treatment.

Just because the file is saved in the repo's directory, though, doesn't mean that Git is tracking it; we need to make a commit that adds it.

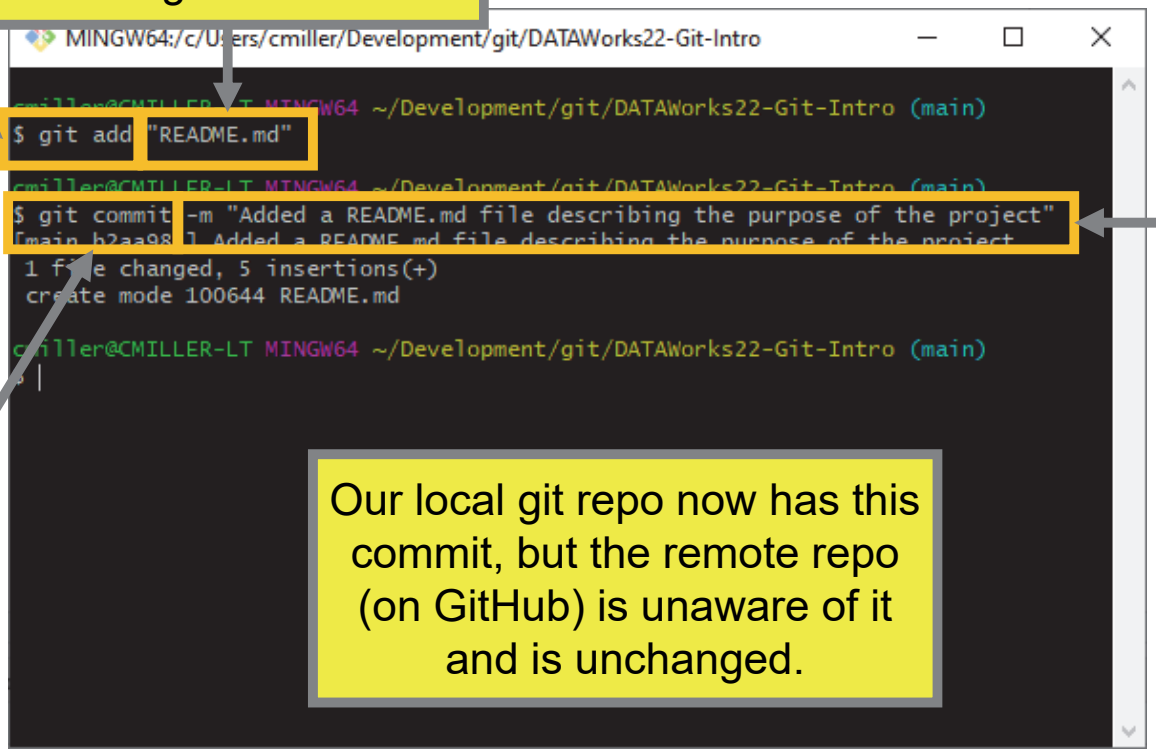
Use `git commit` to create a commit that adds the file and describes its contents



Use `git commit` to create a commit that adds the file and describes its contents

`git add` is the command for adding changes to a commit to track

Name the file with the changes to track



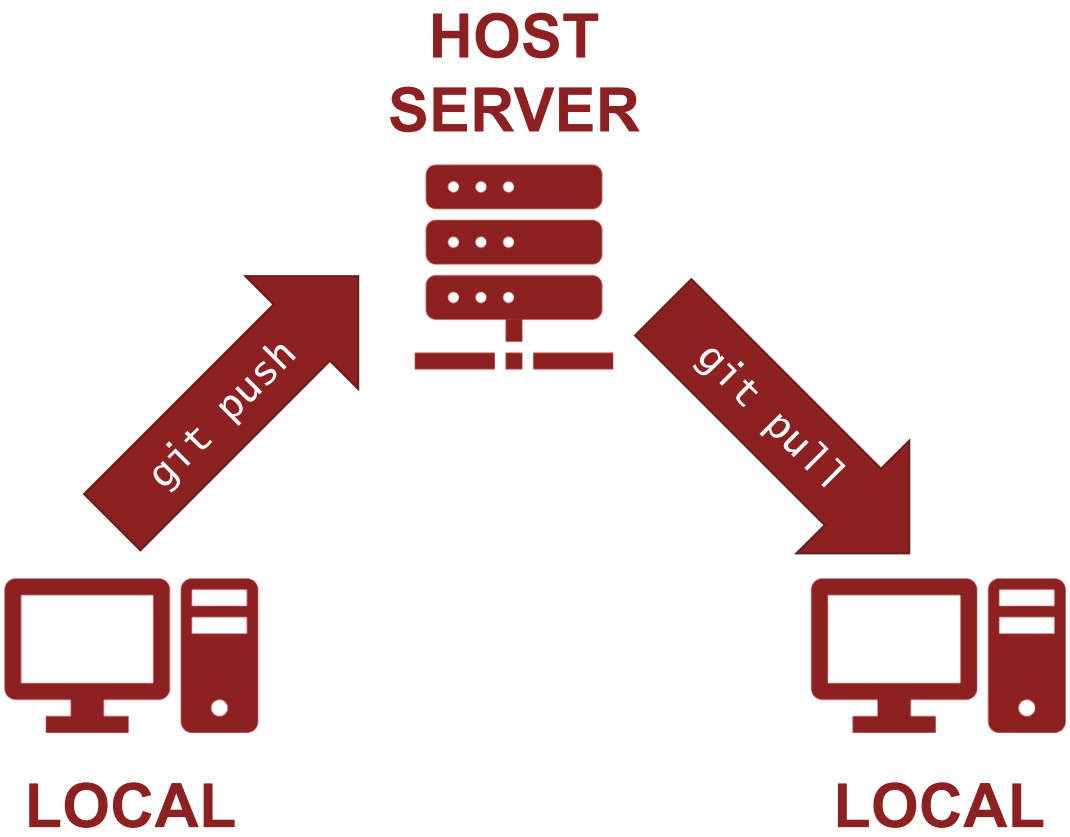
`git commit` is the command for creating a commit, a "save point" containing a set of changes made to the files in a repo

The `-m` option is for adding a message summarizing what changes were made in the commit (keep it short)

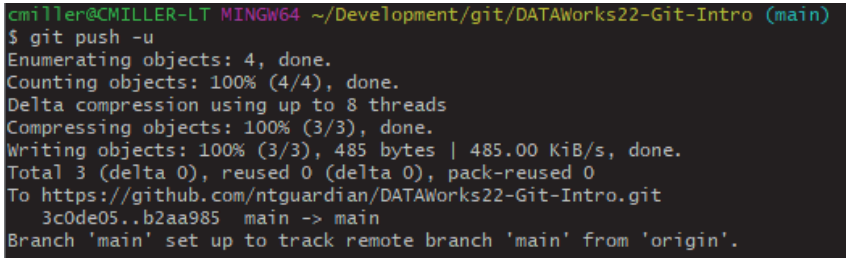
Our local git repo now has this commit, but the remote repo (on GitHub) is unaware of it and is unchanged.

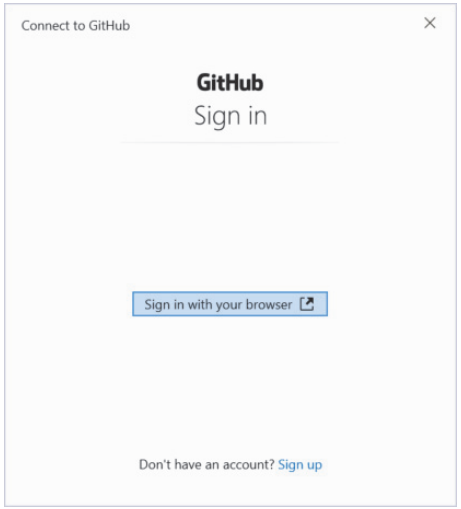
Use git push/git pull to keep your local repo copy up-to-date

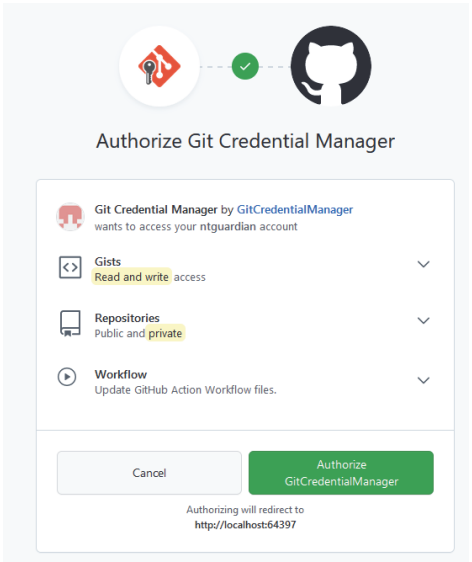
- The commands `git push` and `git pull` keep local copies of git repositories up-to-date with the external host of the repository
- After committing changes, use git push to propagate those changes to the host (and thus to everyone else with an up-to-date repo)
- Push changes at least at the end of the day when work is done
- Use git pull to make the local repo up-to-date with whatever is on the server
- Pull changes at the start of the day before starting work



Pushing to GitHub for the first time may require authentication



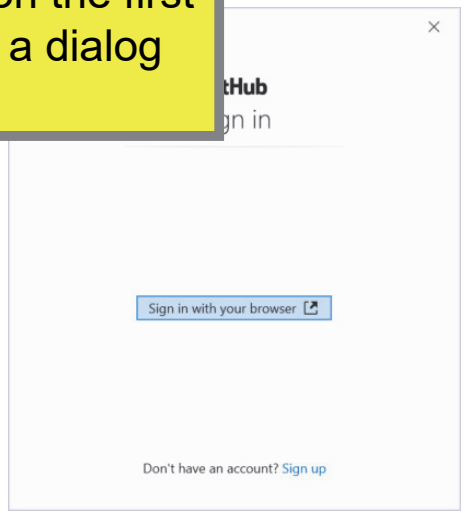




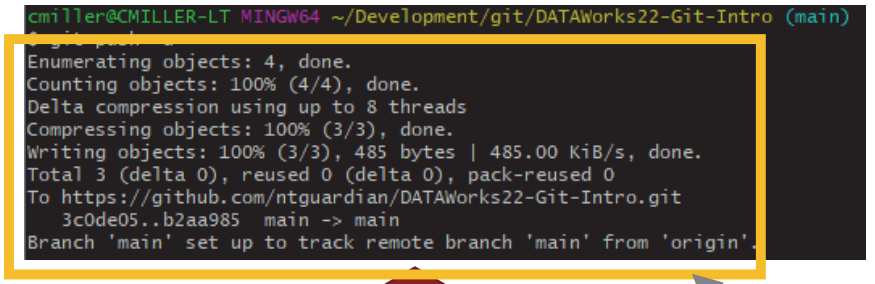
Pushing to GitHub for the first time may require authentication

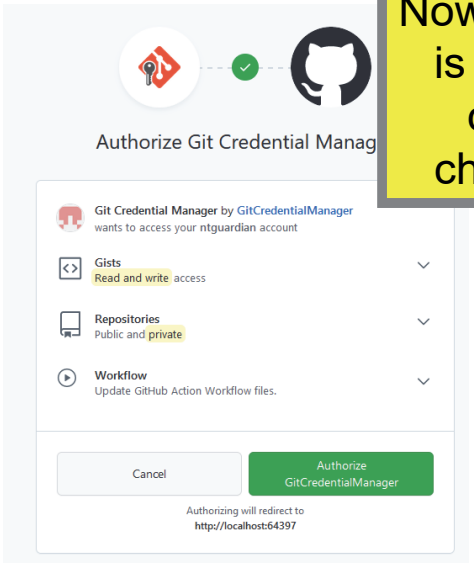
`git push` is the command for propagating changes to the external repo copy

The -u option is needed for authentication the first time, opening a dialog box



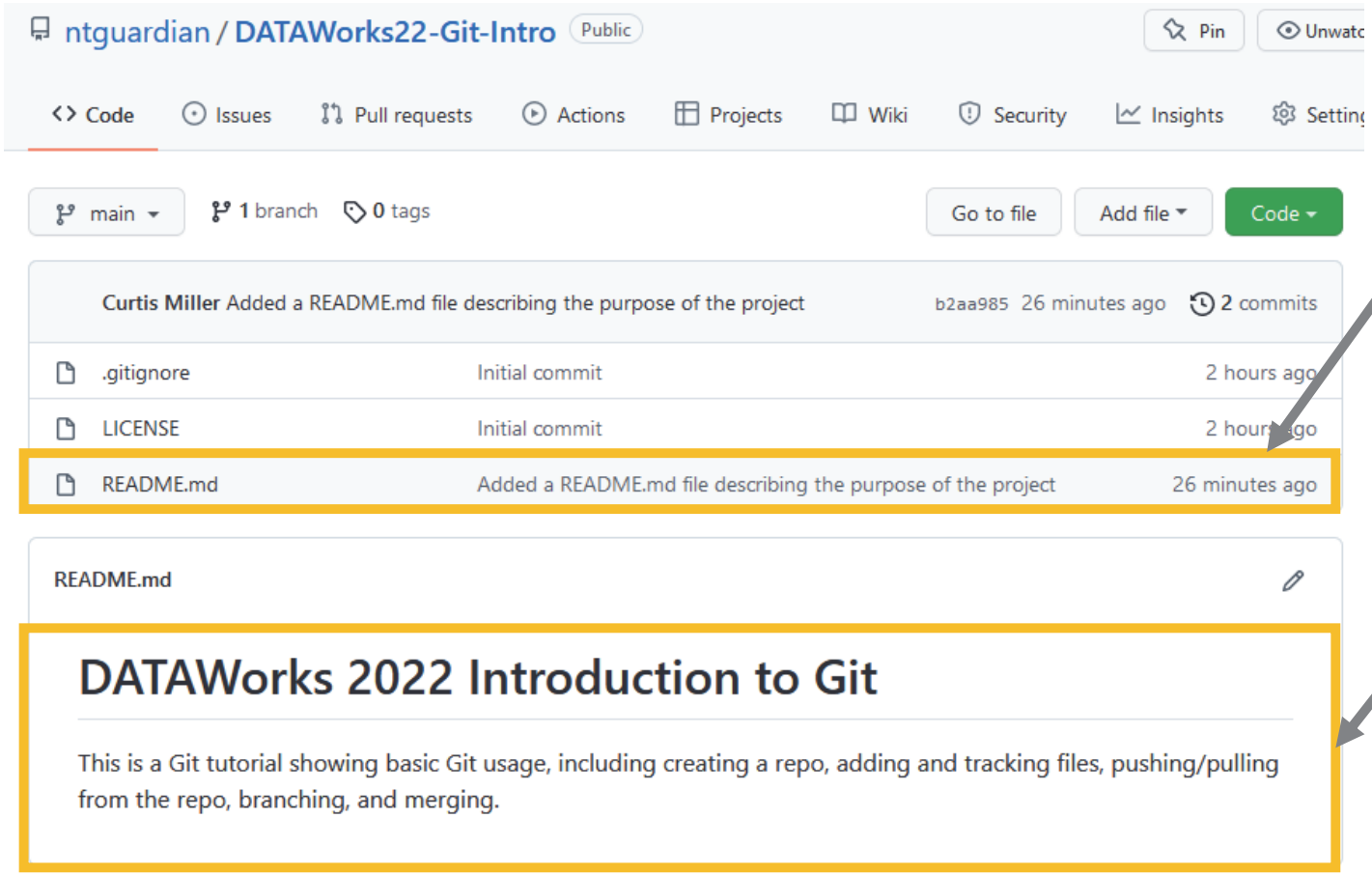
Follow the instructions…





Now the command is done and we can see our changes online

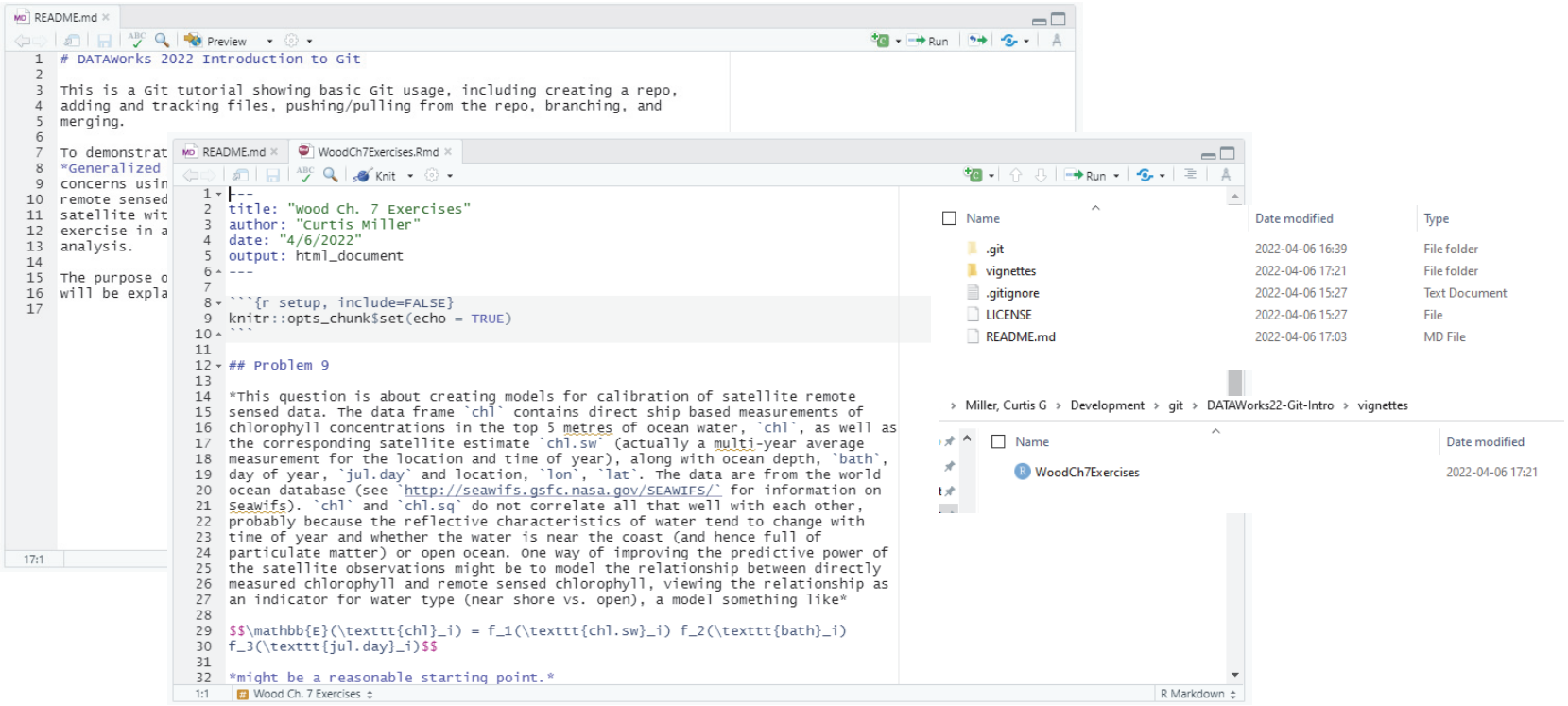
The changes are now online



This file is in the repo now; GitHub shows the time since the last commit, and the message associated with that commit

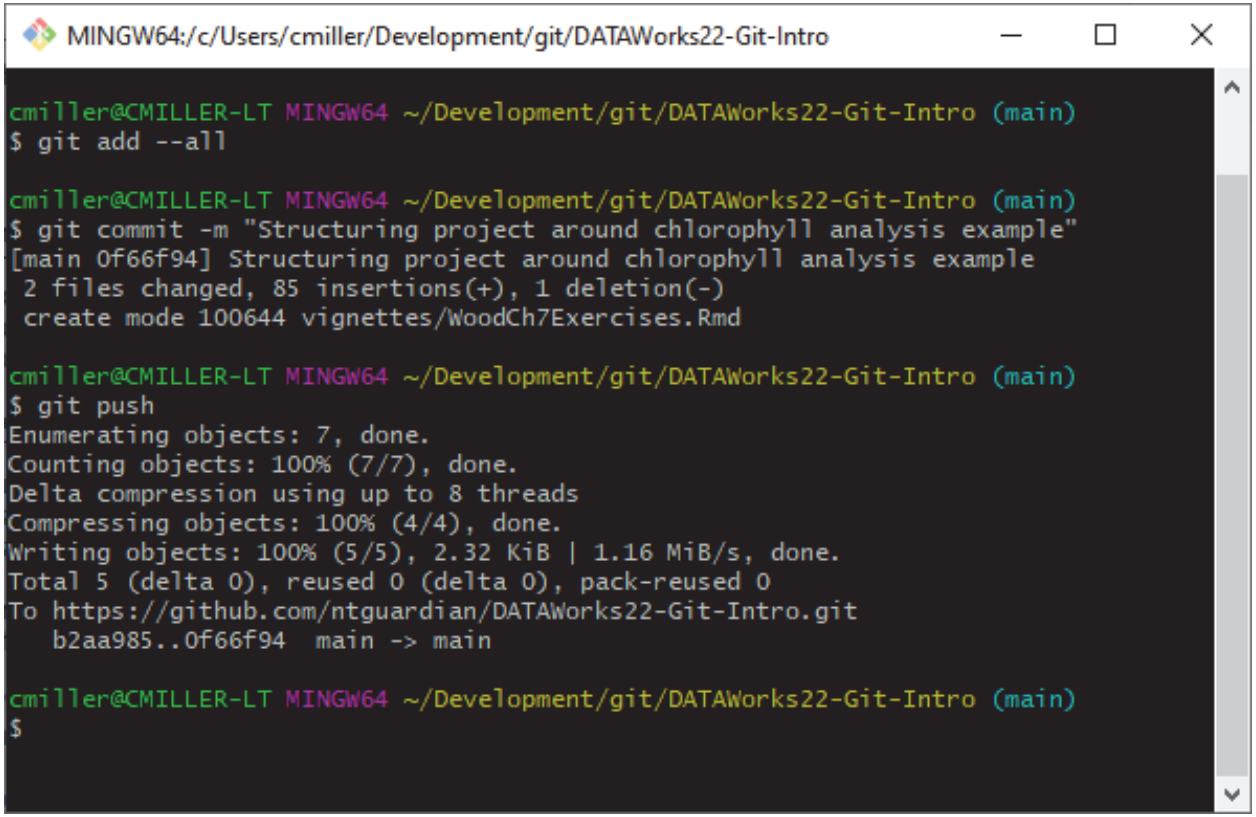
GitHub looks for README.md files and renders them specially; these files describe your project to others

Let's make more changes!



We're structuring the project around a statistical analysis of ocean chlorophyll data using generalized additive models. That will be described in a file stored in the vignettes directory. README.md has been changed to reflect this purpose.

Track them changes!

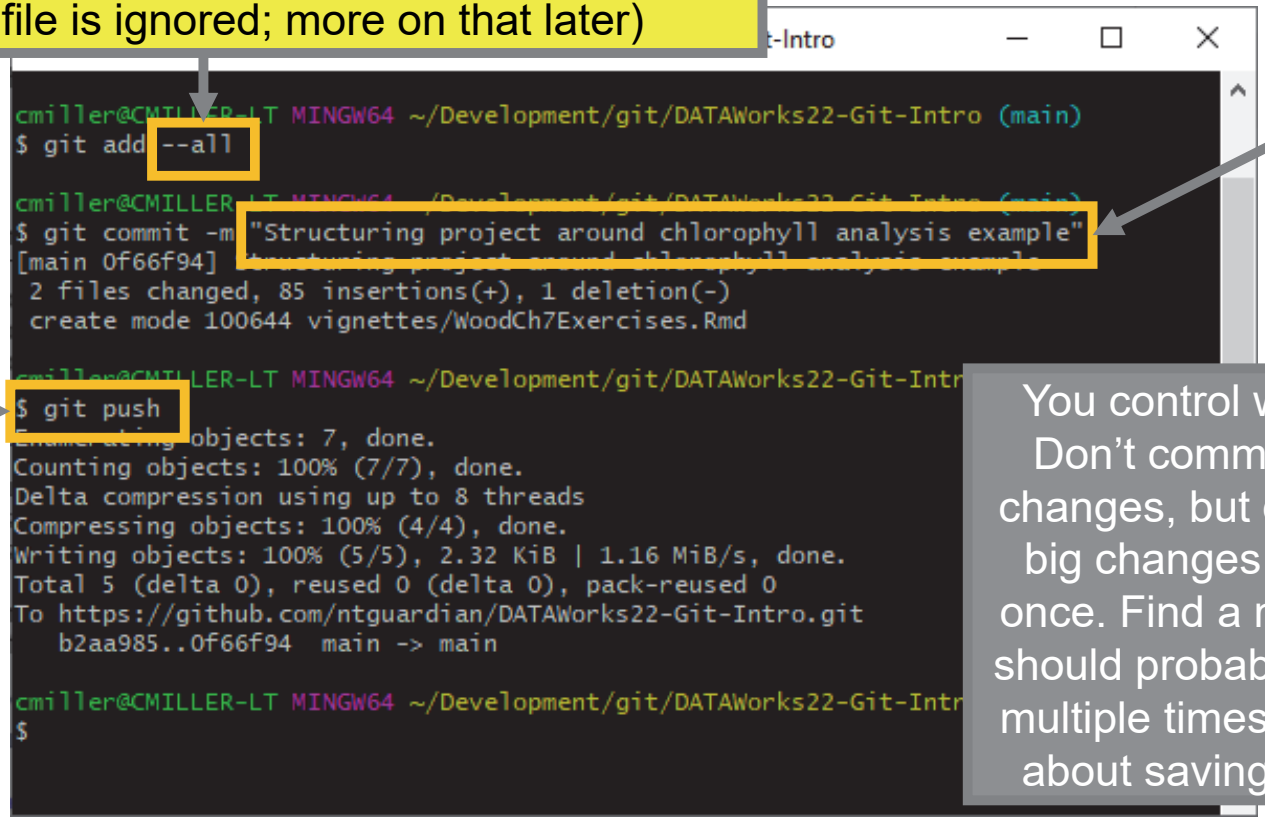


Repeating these three commands will be 90% of your Git commands

Instead of listing files to track individually, use this option to add all changes made to any file in the repo (unless that file is ignored; more on that later)

Commit message describing what change was made; please make your message informative

Push your commands to the remote host; you don't need to do this with every commit, but do it at least at the end of a workday, or when wanting to share immediately

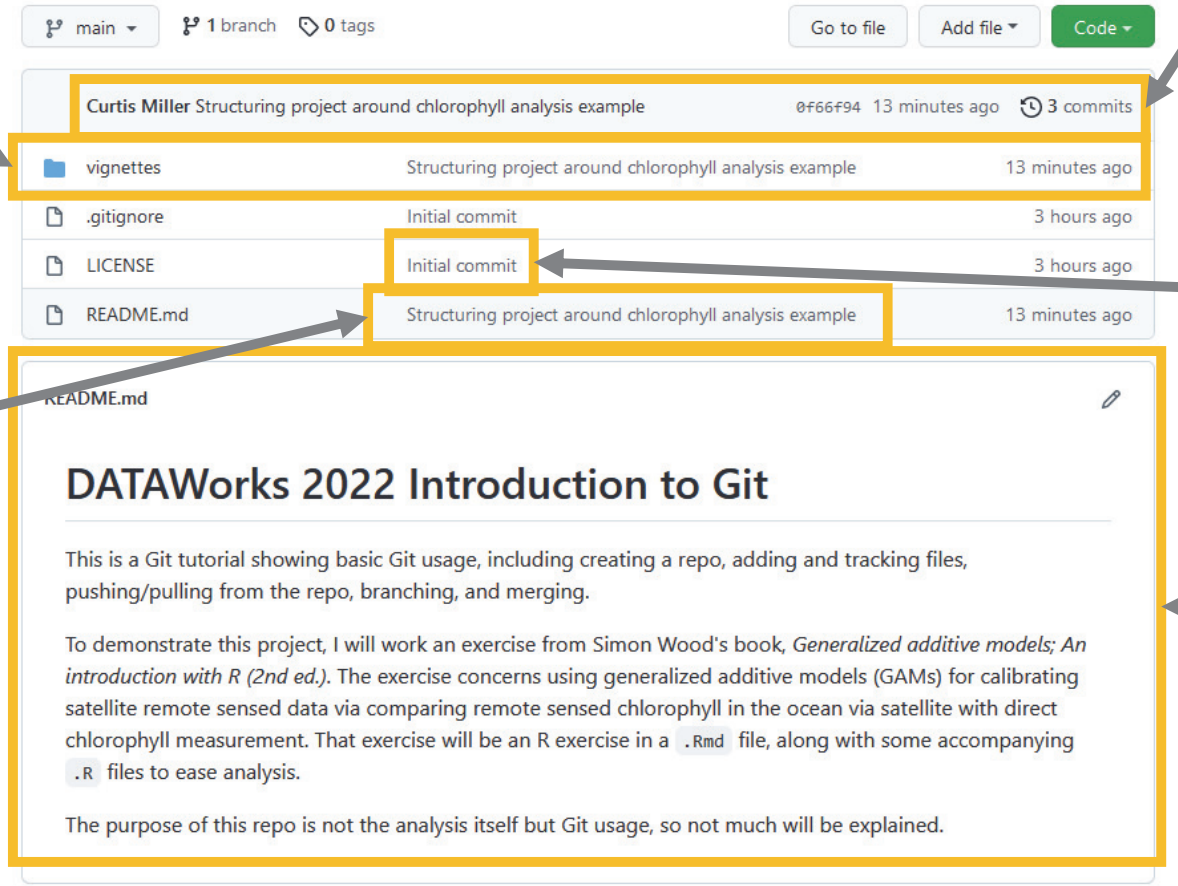


You control when you commit. Don't commit a bunch of little changes, but don't make several big changes and commit all at once. Find a middle ground. You should probably commit changes multiple times a day (don't worry about saving space too much)

Notice what all has changed on GitHub

There are new files in the project

Now using the most recent commit message since this file was altered

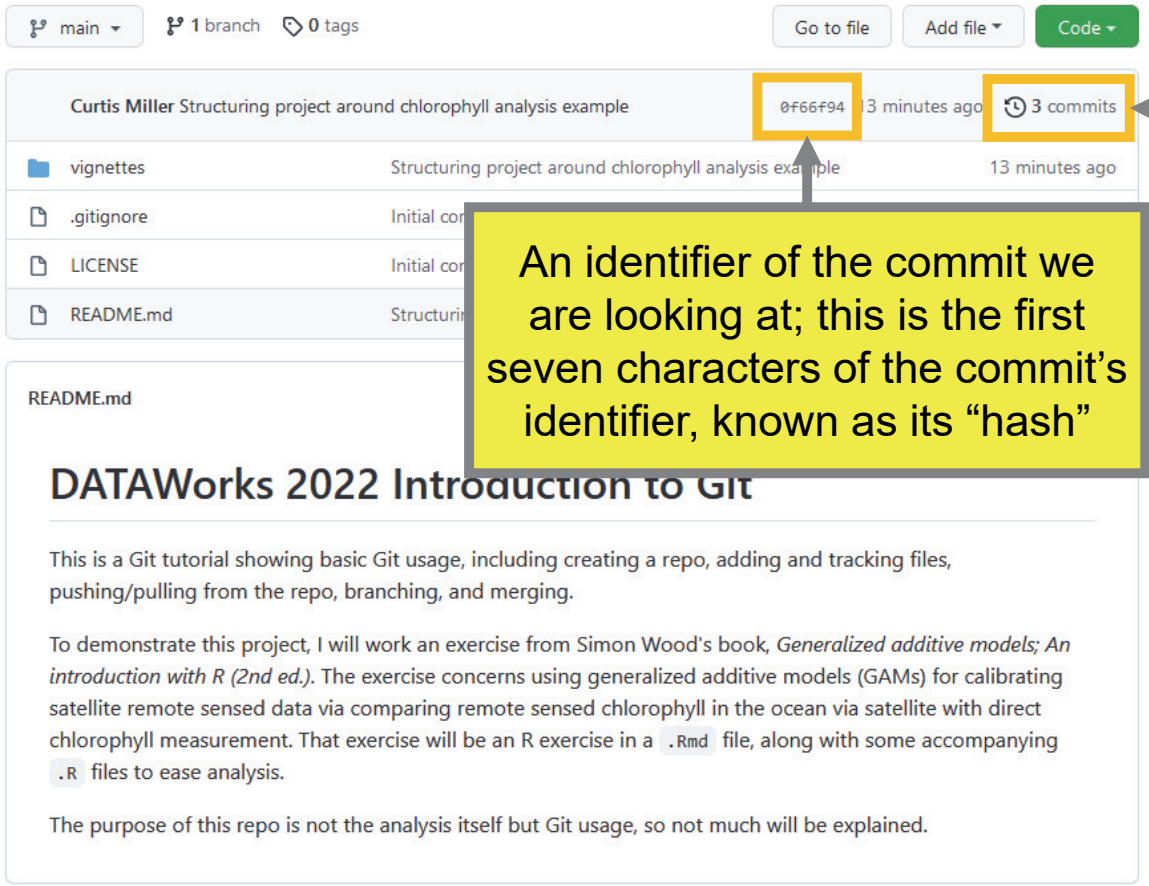


Indicates the most up-to-date state of the project

This file has not been altered so nothing has changed

README.md now has the new text

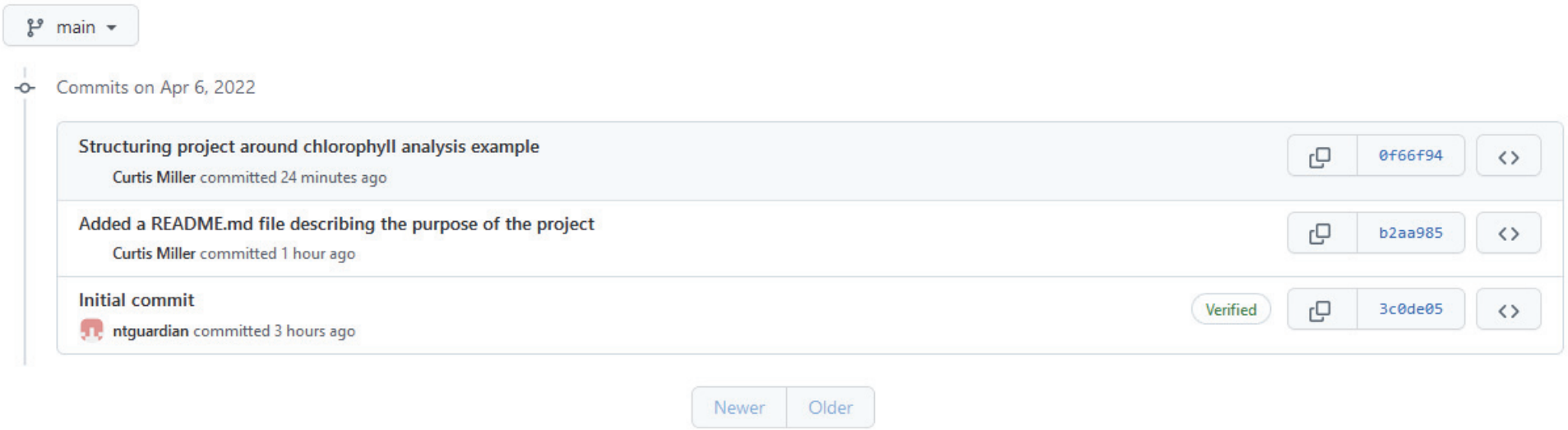
Let's see the history of the project so far



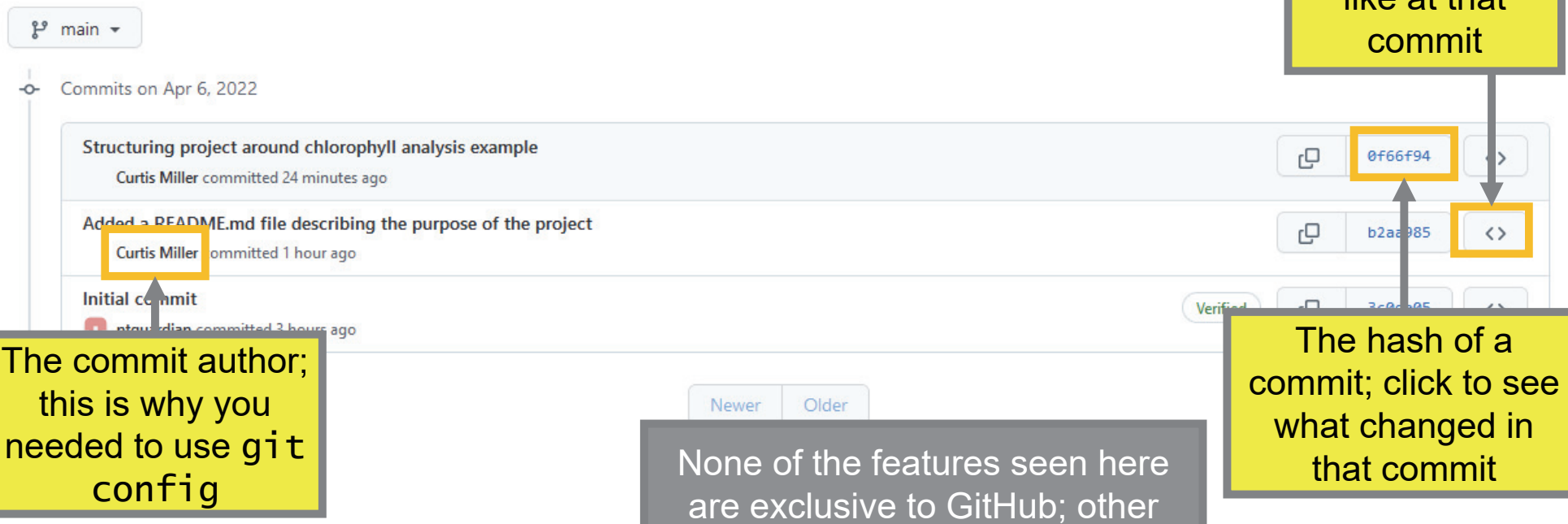
Click this to see the history of the project

An identifier of the commit we are looking at; this is the first seven characters of the commit's identifier, known as its "hash"

Our project has a short history



From here we can explore that history



Click this to see what the project looked like at that commit

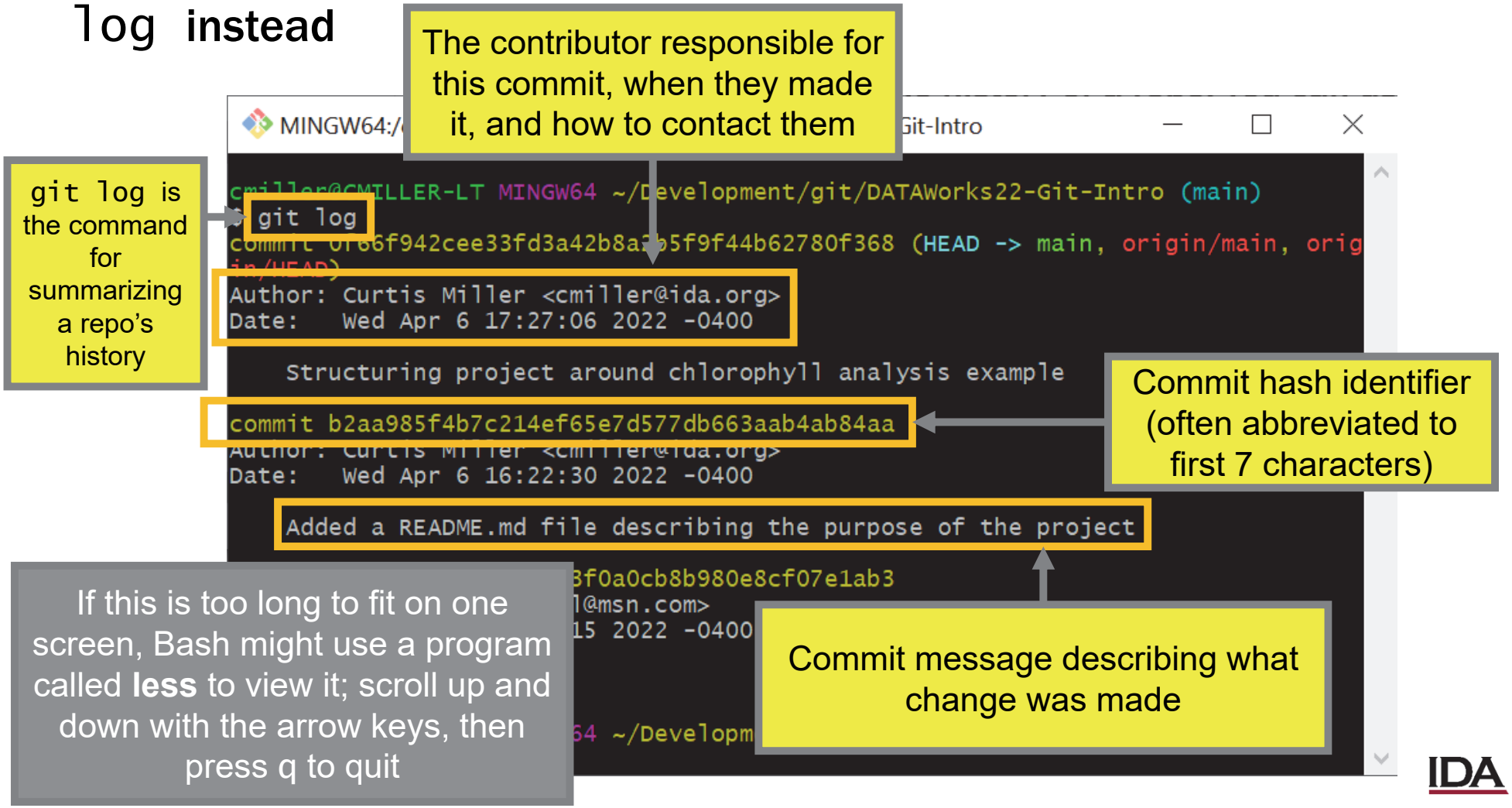
The commit author; this is why you needed to use `git config`

The hash of a commit; click to see what changed in that commit

None of the features seen here are exclusive to GitHub; other repo hosts have their own versions of this, and Git itself can allow looking at this information.

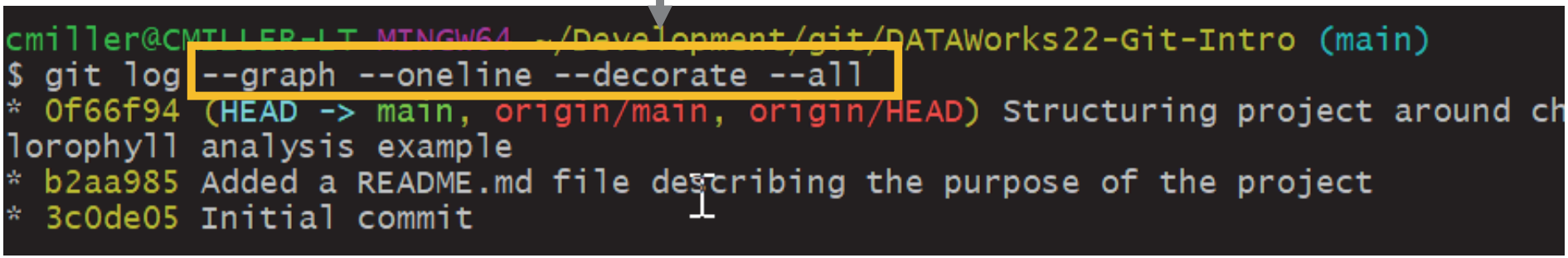
You don't need GitHub to see the history of a repo; you can use `git log` instead

You don't need GitHub to see the history of a repo; you can use `git log` instead

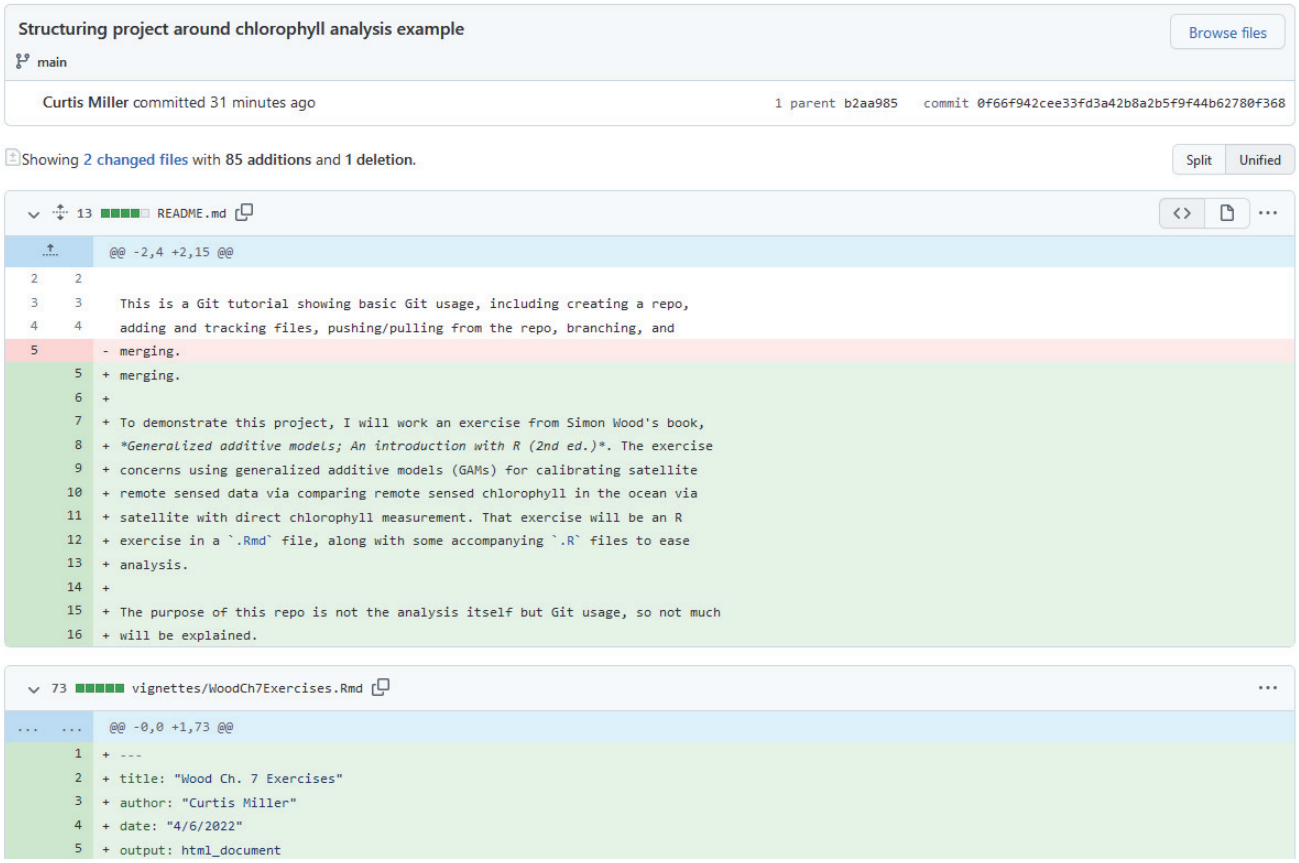


Options you can pass to `git log` can alter the output and make it easier to see at a glance

Using these options can make a more digestible history of a project; see `git log --help` for more



A commit lists precisely what the difference between it and the previous commit is; for text files, Git tracks *differences only*



Structuring project around chlorophyll analysis example

Browse files

main

Curtis Miller committed 31 minutes ago 1 parent b2aa985 commit 0f66f942cee33fd3a42b8a2b5f9f44b62780f368

Showing 2 changed files with 85 additions and 1 deletion. Split Unified

13 README.md

```
@@ -2,4 +2,15 @@
 
  This is a Git tutorial showing basic Git usage, including creating a repo,
  adding and tracking files, pushing/pulling from the repo, branching, and
- merging.
+ merging.
+
+ To demonstrate this project, I will work an exercise from Simon Wood's book,
+ *Generalized additive models; An introduction with R (2nd ed.)*. The exercise
+ concerns using generalized additive models (GAMs) for calibrating satellite
+ remote sensed data via comparing remote sensed chlorophyll in the ocean via
+ satellite with direct chlorophyll measurement. That exercise will be an R
+ exercise in a `.Rmd` file, along with some accompanying `.R` files to ease
+ analysis.
+
+ The purpose of this repo is not the analysis itself but Git usage, so not much
+ will be explained.
```

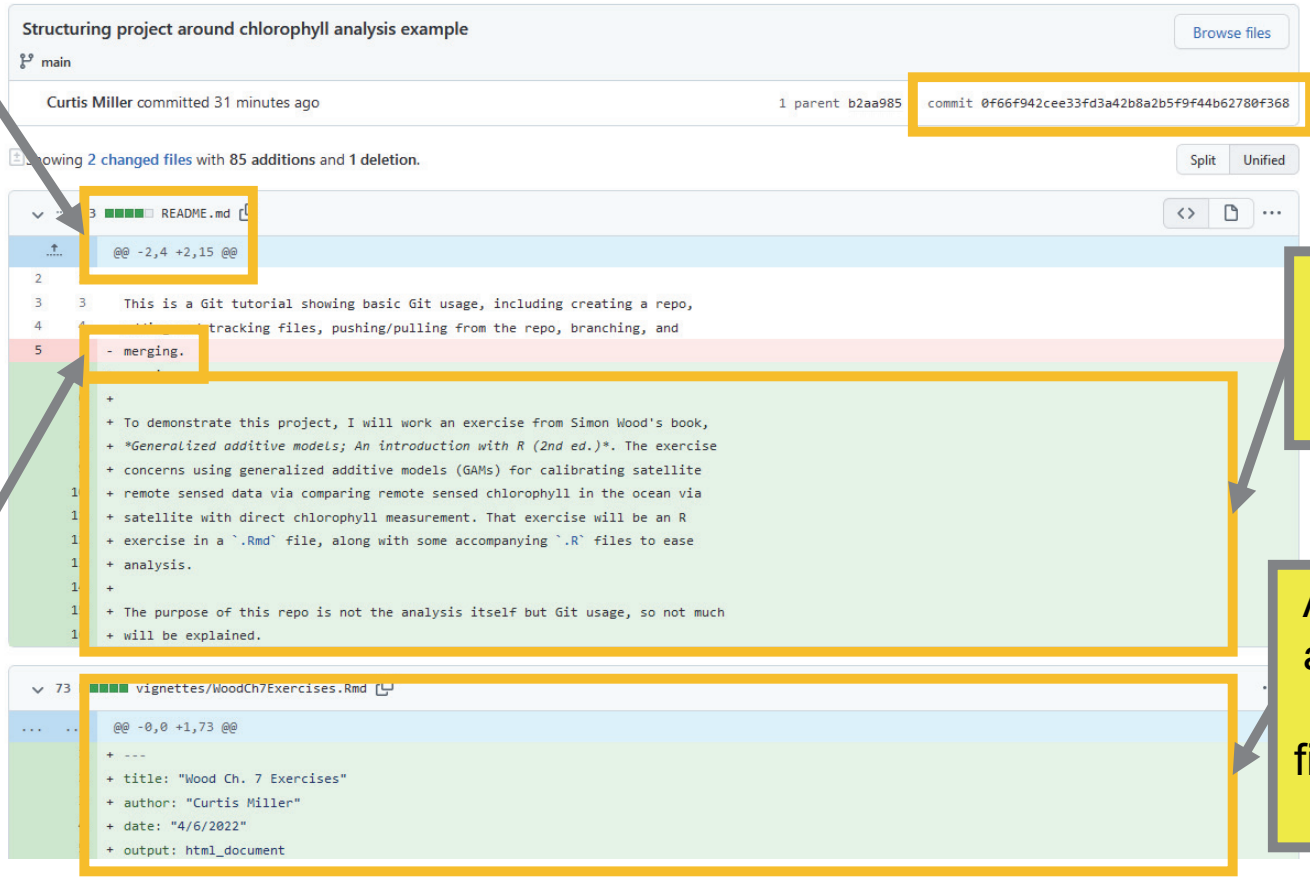
73 vignettes/WoodCh7Exercises.Rmd

```
@@ -0,0 +1,73 @@
+ ---
+ title: "Wood Ch. 7 Exercises"
+ author: "Curtis Miller"
+ date: "4/6/2022"
+ output: html_document
```

We are looking at what's known as a Git diff (short for "difference")

The file that was changed, with descriptor of where the change was made

Deleted lines are shown in red; if a line was modified, the old version is deleted (red) and the new version added (green)



The full hash identifier of the commit

Additions to a file are shown in green

A whole new file is all additions, so all green (a deleted file would be all red for all deletions)

We can get the same information using `git diff`

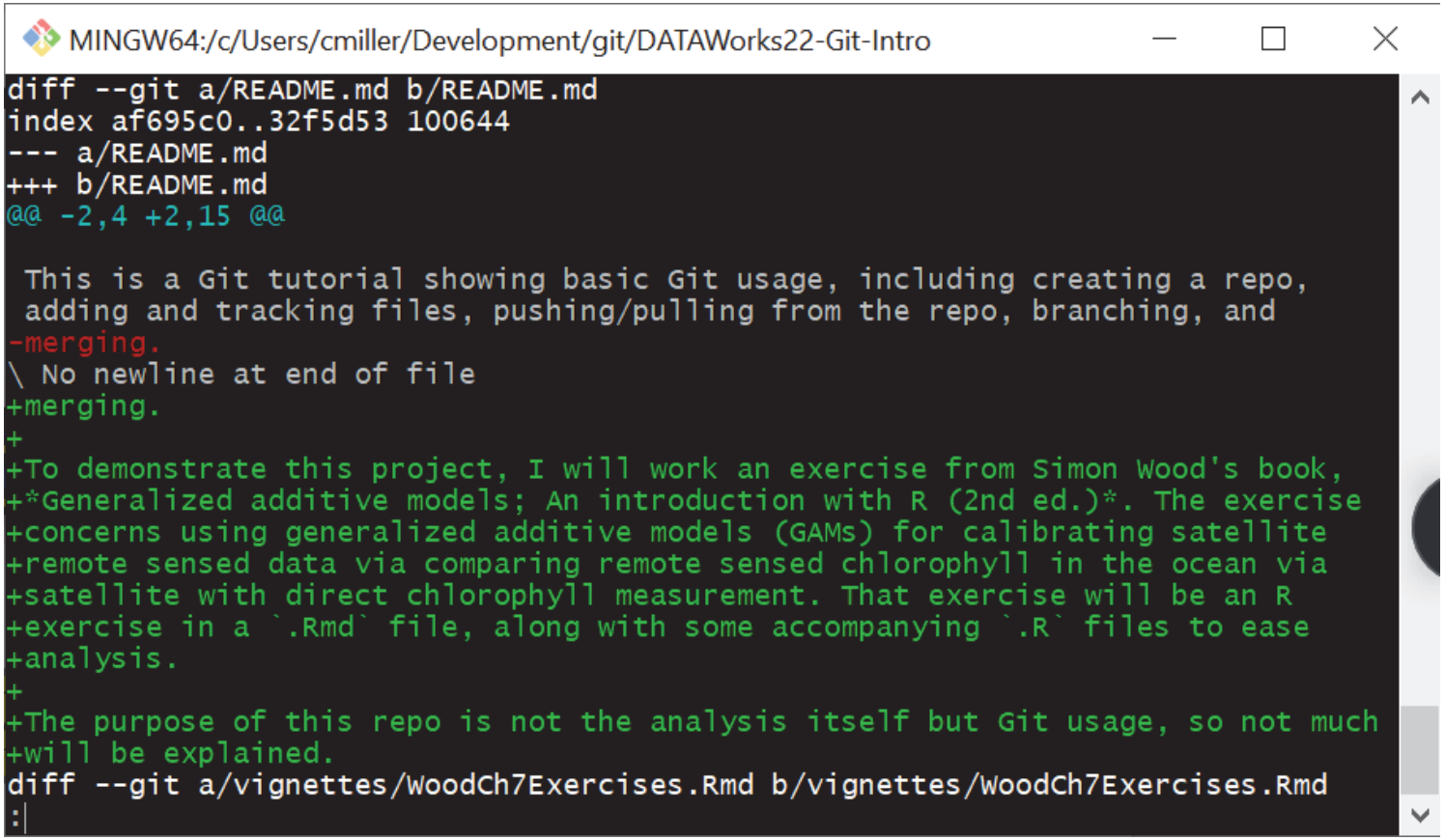
```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git diff HEAD~ HEAD
```

`git diff` can compare many entities Git tracks

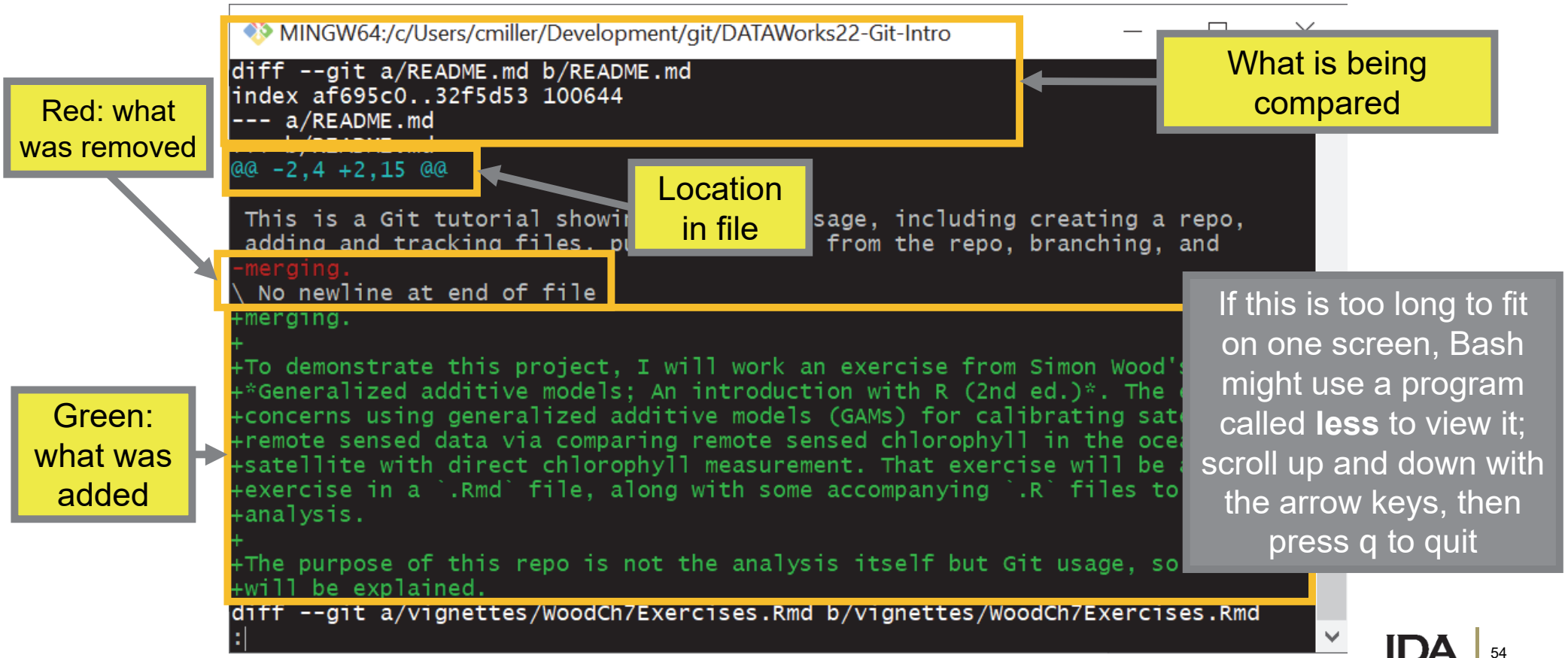
`git diff` is the command for comparing files tracked by Git

Determines what to compare; in this case, the current commit with the one prior

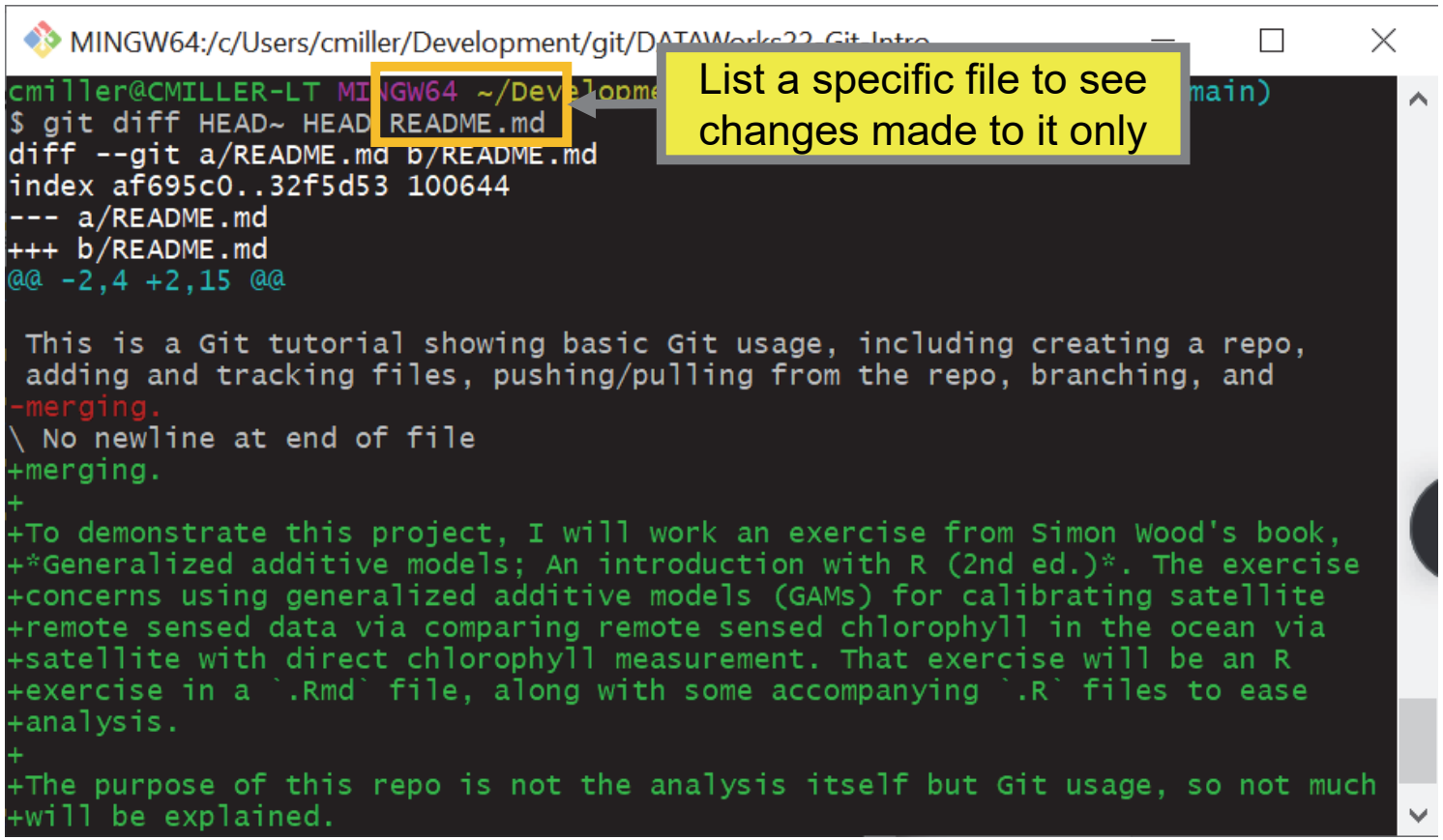
Using `git diff` we can see changes made in commits to files



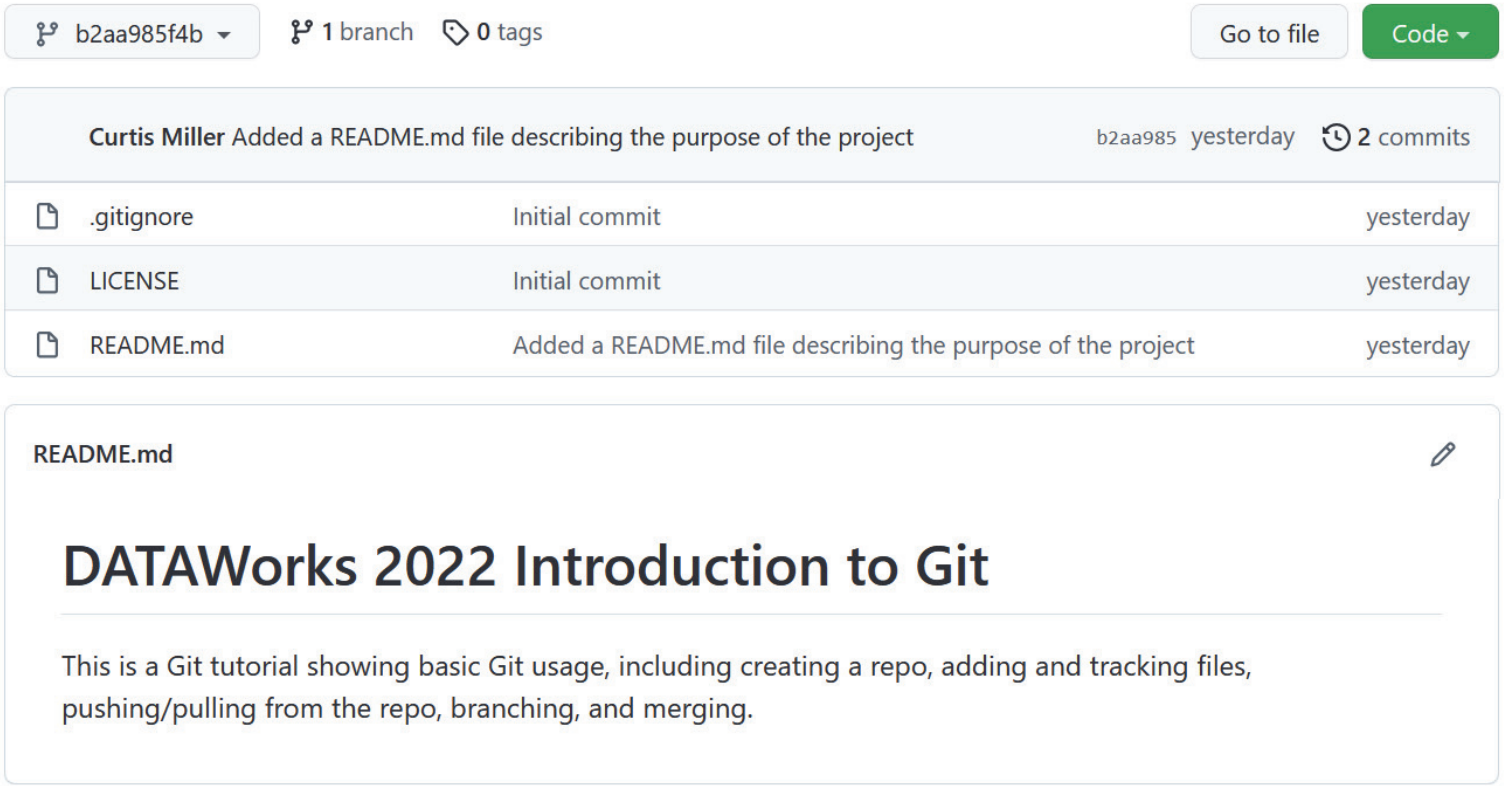
For plain text files, Git tracks only the changes, and doesn't copy whole files if it doesn't need to



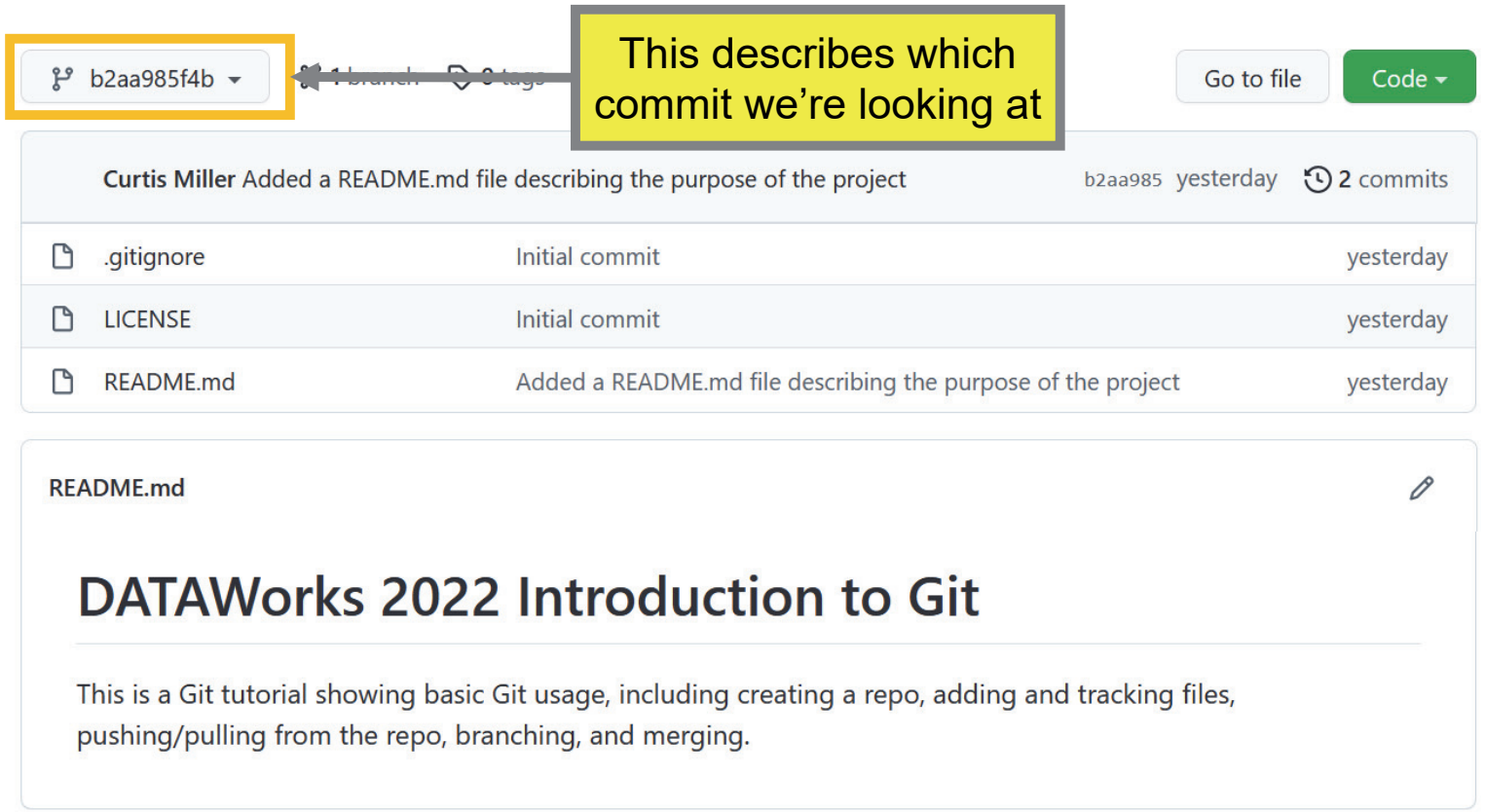
We can also see what changed in a specific file



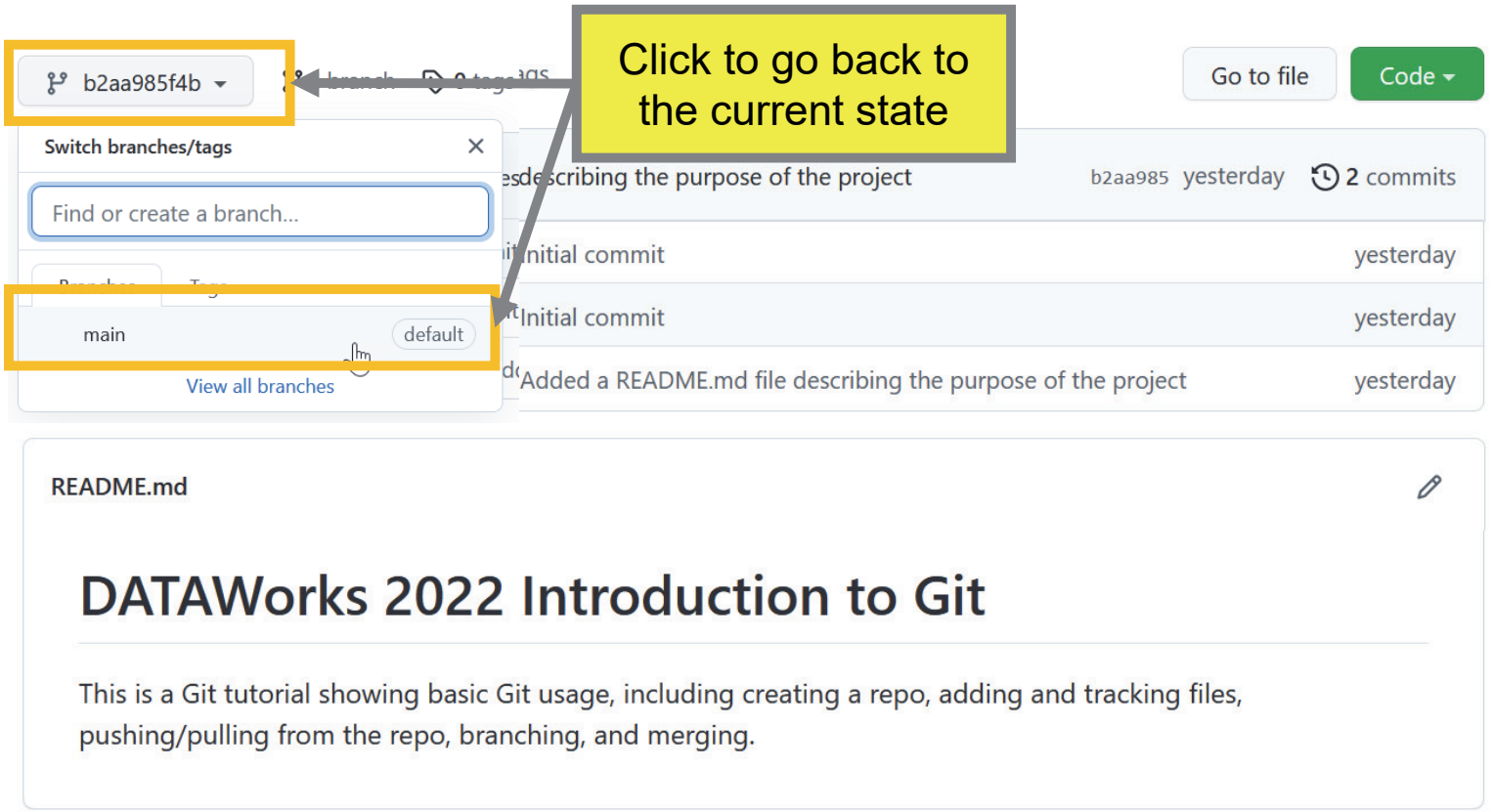
We can also see what a repo looked like at the moment a commit was made



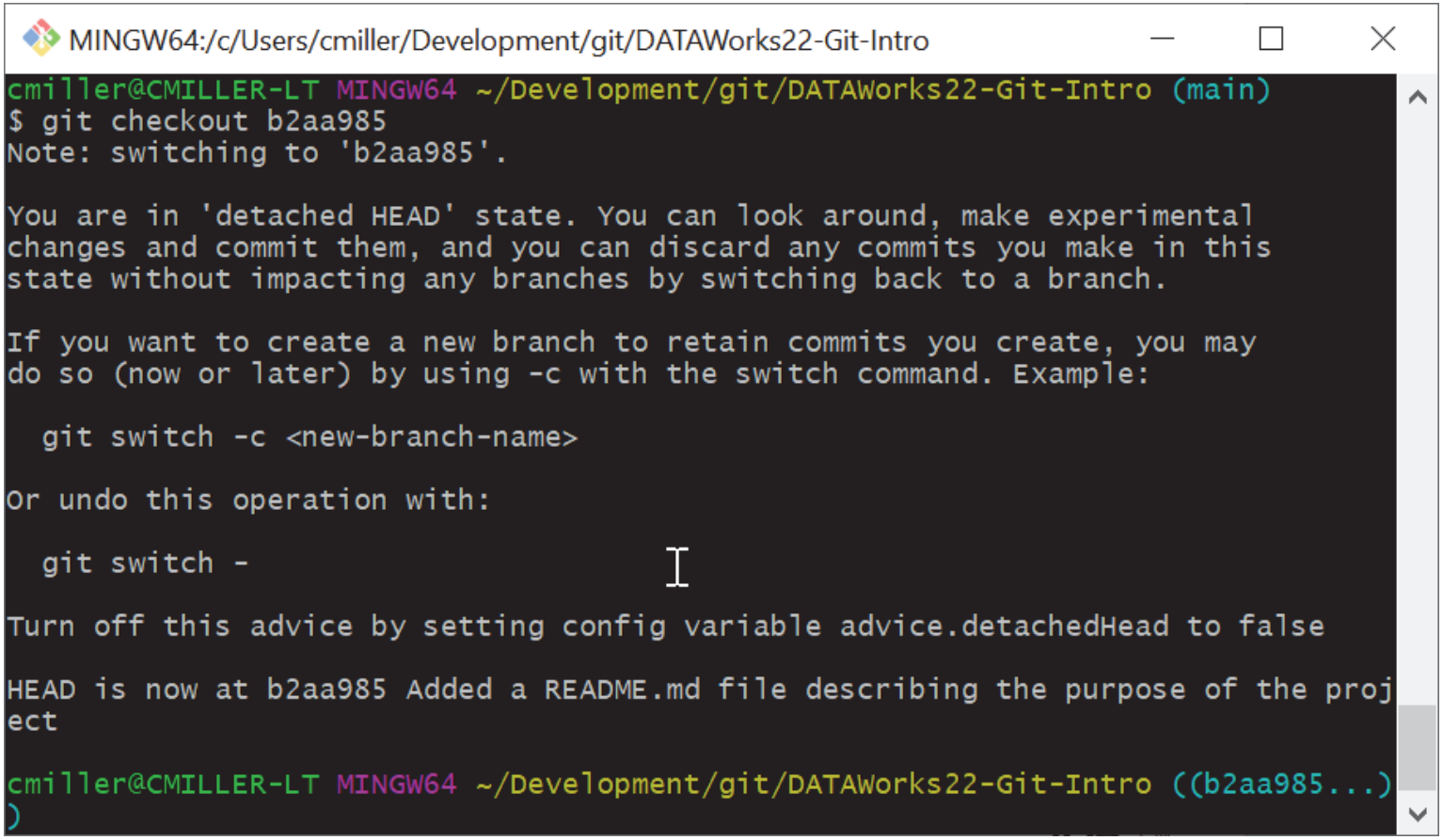
This includes not only changes made but the entire state of the project



We can also see what a repo looked like at the moment a commit was made



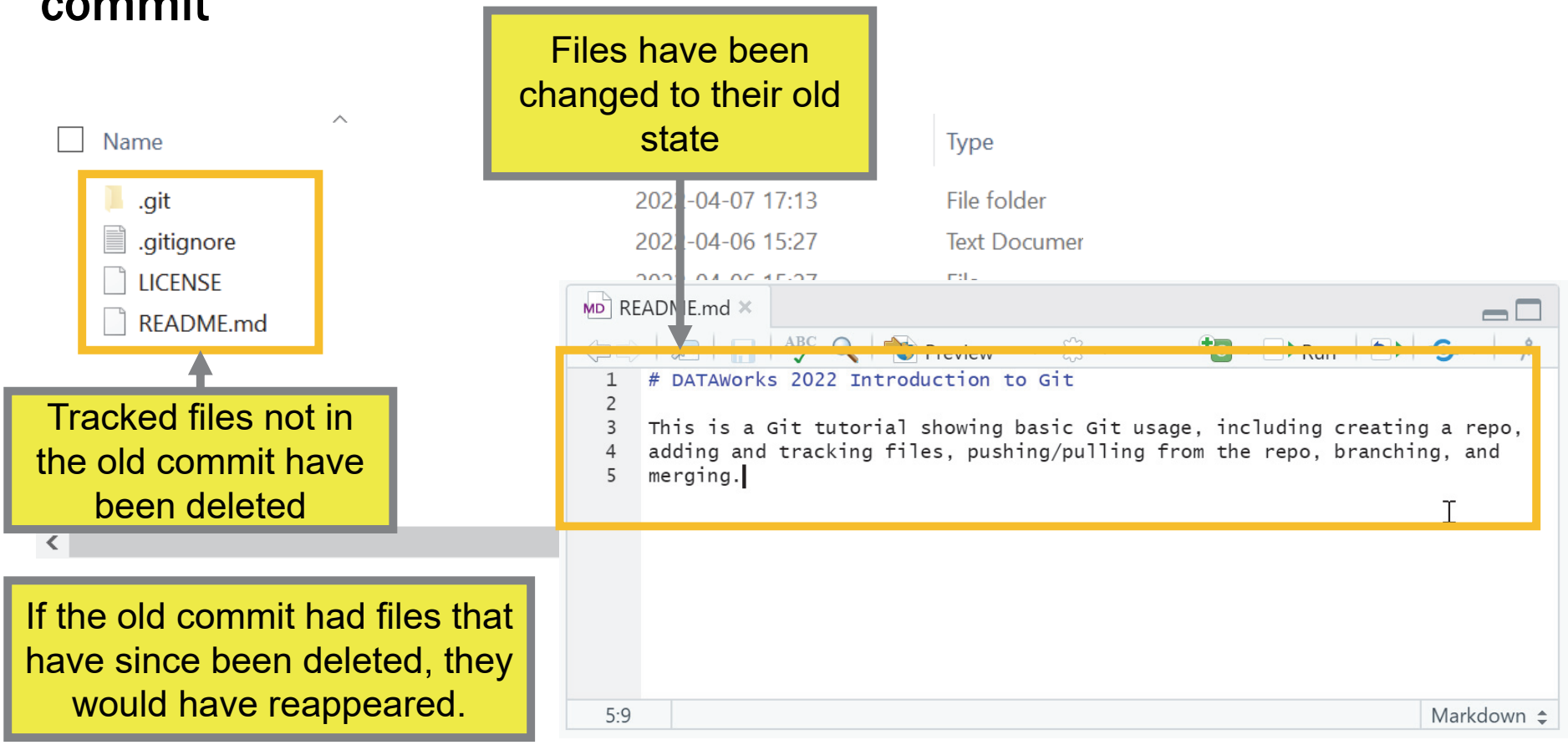
We can do this without GitHub using `git checkout`



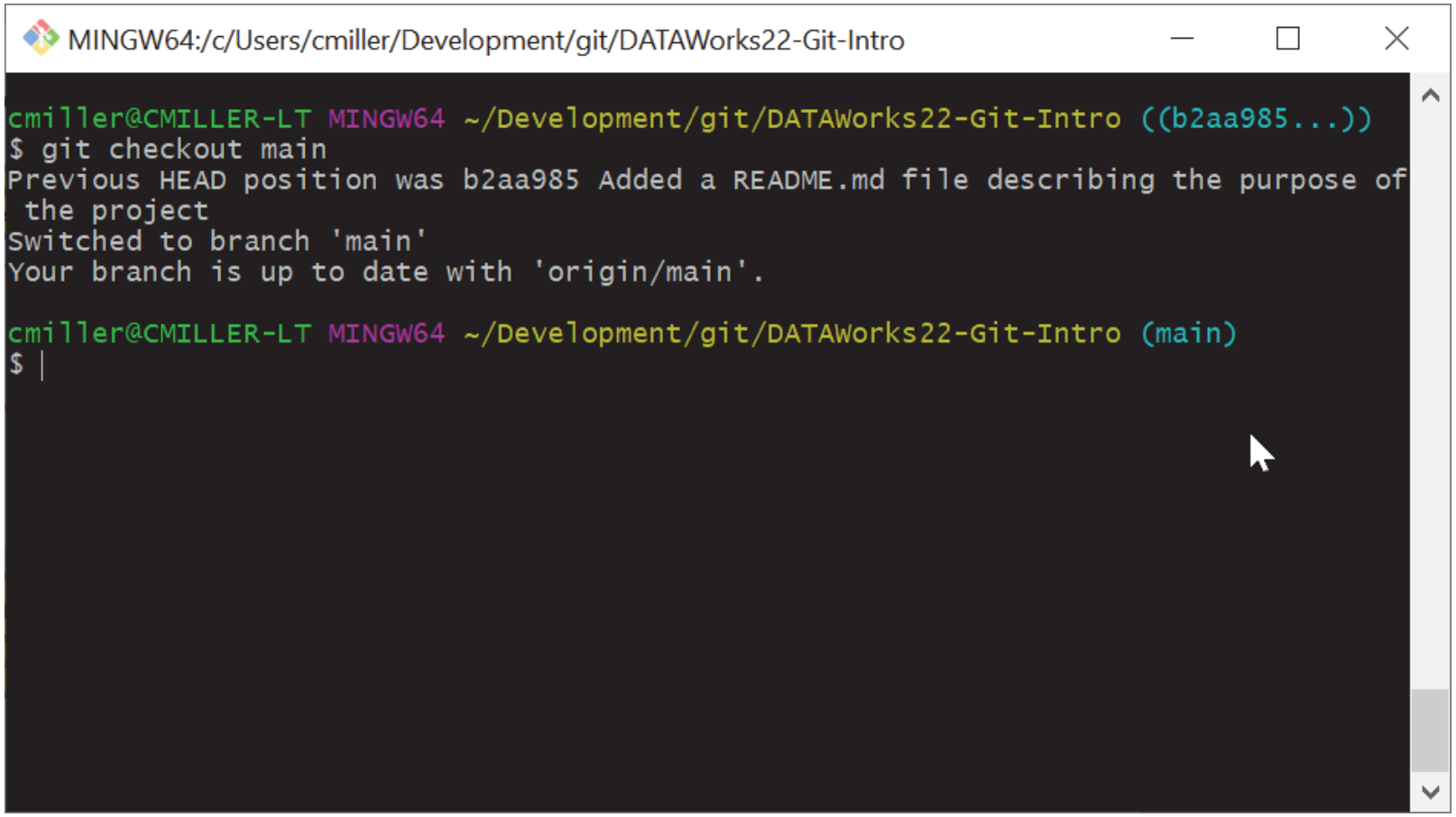
This will bring a repo's files to a previously tracked state

`git checkout` gets different versions of the repo, including old commits and branches

The repo's contents on our hard drive will change when checking out a commit

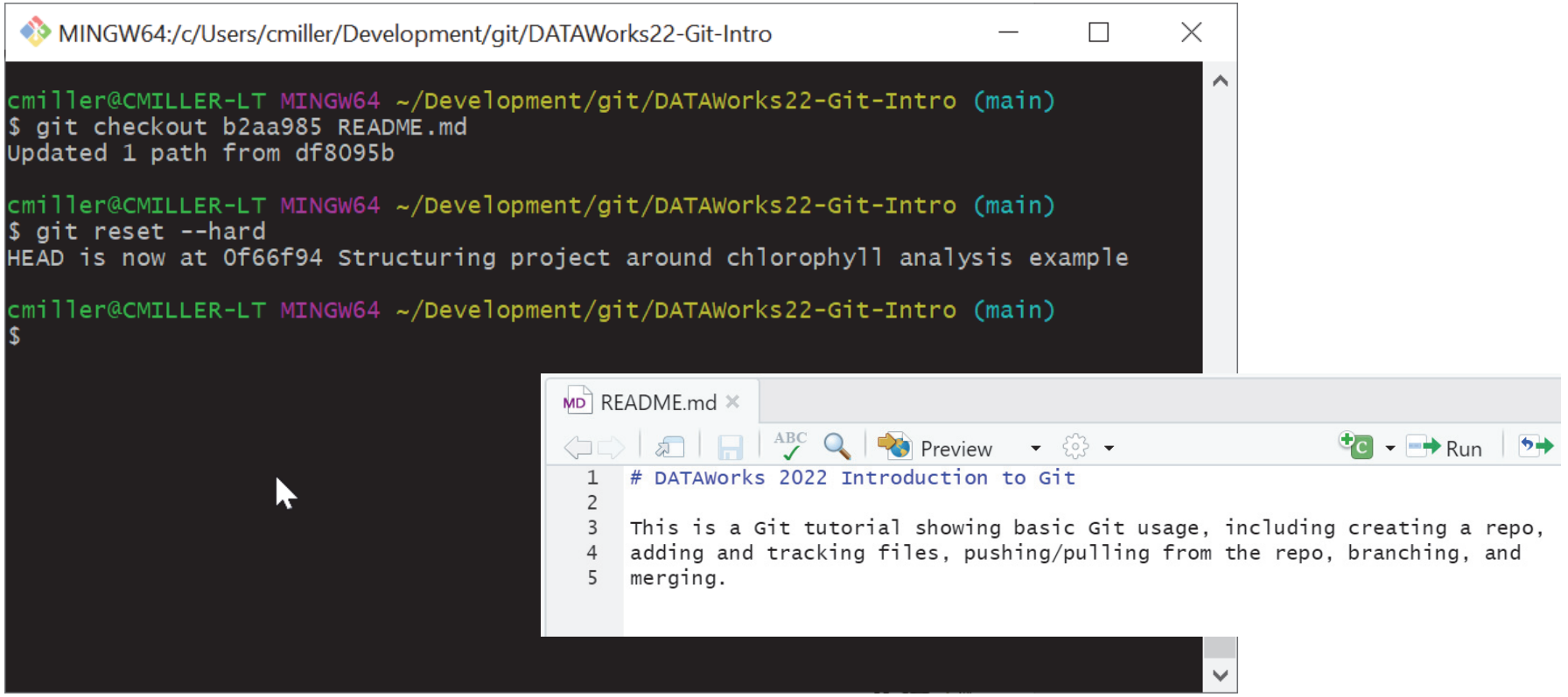


“So how do I go back to my most recent commit?”

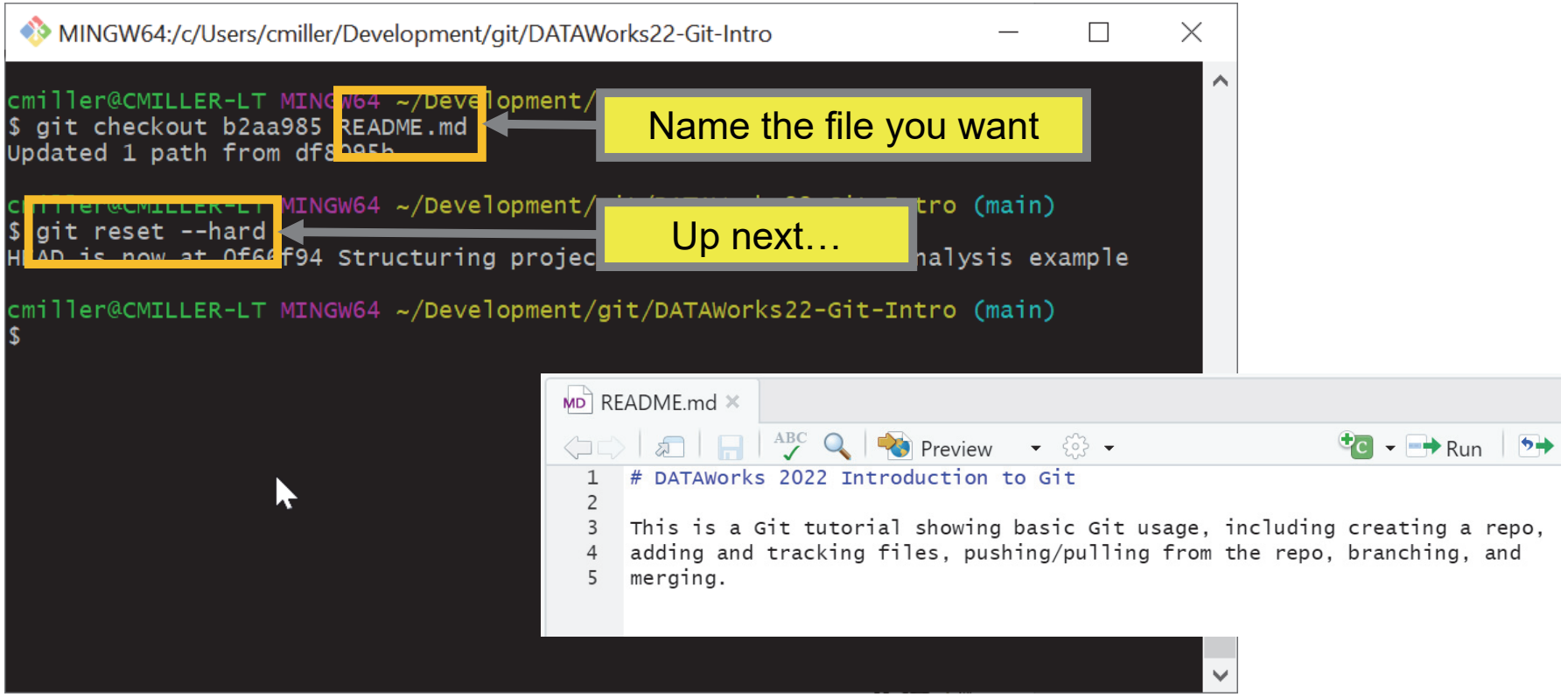


You check out the main branch

“Well, I only want one file from that commit.”

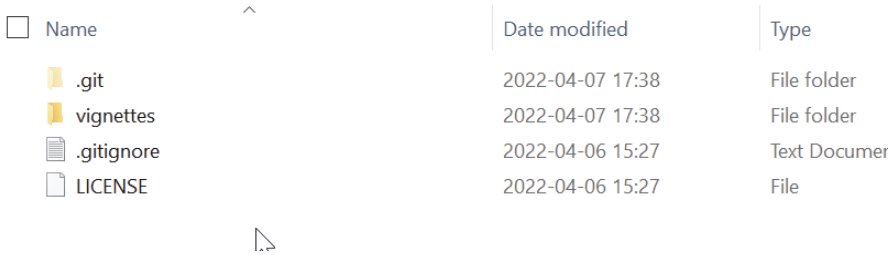


Just say what file you want



“Oh no! I deleted an important file! I want it back!”

1

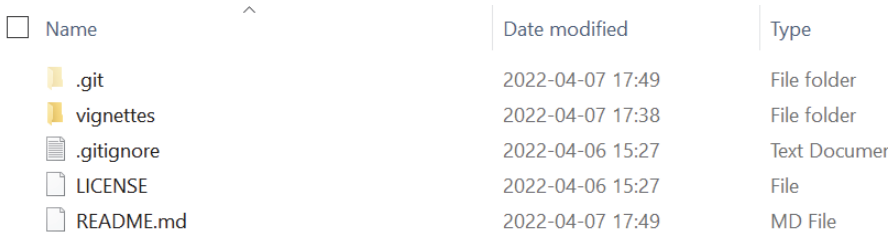


Name	Date modified	Type
.git	2022-04-07 17:38	File folder
vignettes	2022-04-07 17:38	File folder
.gitignore	2022-04-06 15:27	Text Documer
LICENSE	2022-04-06 15:27	File

2

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git reset --hard
HEAD is now at 0f66f94 Structuring project around chlorophyll analysis example
```

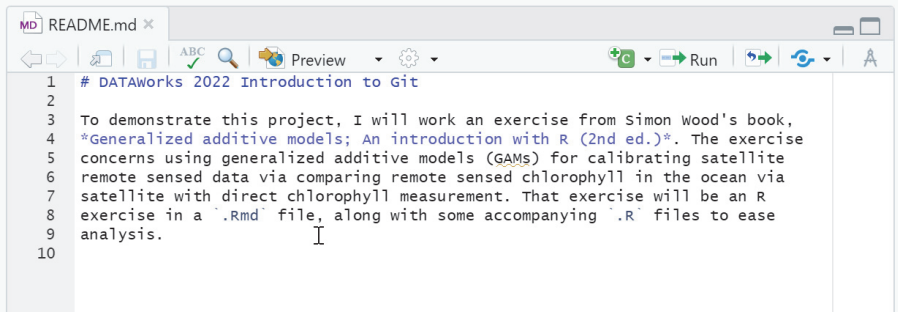
3



Name	Date modified	Type
.git	2022-04-07 17:49	File folder
vignettes	2022-04-07 17:38	File folder
.gitignore	2022-04-06 15:27	Text Documer
LICENSE	2022-04-06 15:27	File
README.md	2022-04-07 17:49	MD File

“Oh no! I changed and saved a file but all my changes were stupid! I want the old one back!”

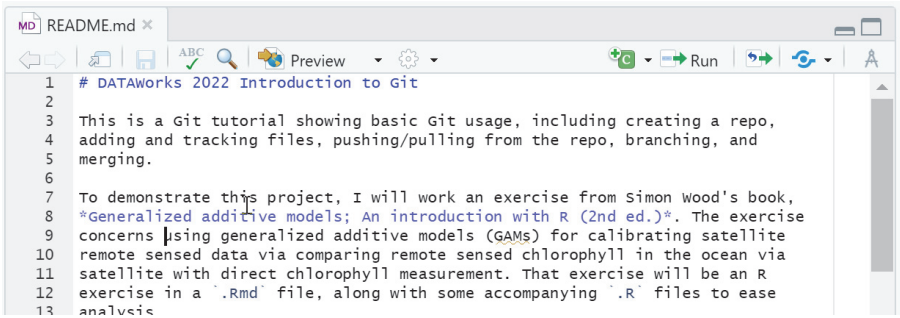
1



2

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git reset --hard
HEAD is now at 0f66f94 Structuring project around chlorophyll analysis example
```

3



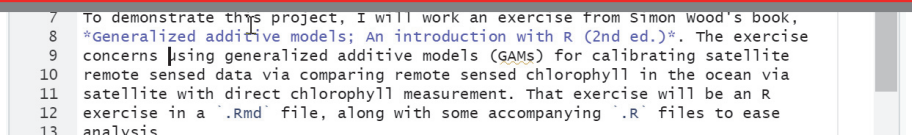
“Oh no! I changed and saved a file but all my changes were stupid! I want the old one back!”

2

WARNING!!!

`git reset --hard` **will undo EVERYTHING!**
Be sure there are no changes in tracked files you actually want to keep!

THIS IS THE ONLY WARNING YOU WILL EVER GET BEFORE LOSING WORK THIS WAY!

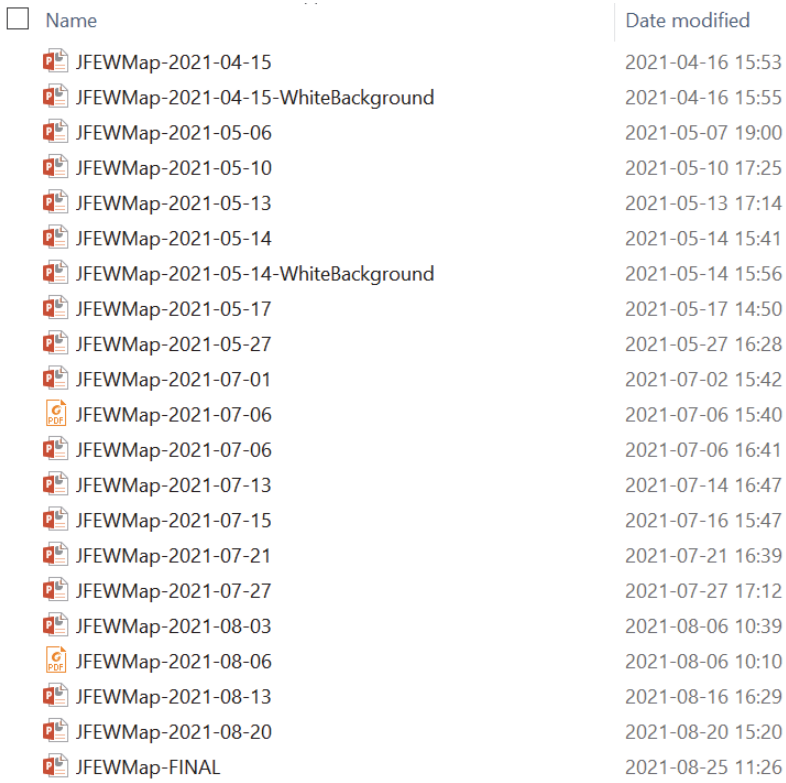


When using Git, you can now drop some terrible habits, like…

Name	Date modified
JFEWMap-2021-04-15	2021-04-16 15:53
JFEWMap-2021-04-15-WhiteBackground	2021-04-16 15:55
JFEWMap-2021-05-06	2021-05-07 19:00
JFEWMap-2021-05-10	2021-05-10 17:25
JFEWMap-2021-05-13	2021-05-13 17:14
JFEWMap-2021-05-14	2021-05-14 15:41
JFEWMap-2021-05-14-WhiteBackground	2021-05-14 15:56
JFEWMap-2021-05-17	2021-05-17 14:50
JFEWMap-2021-05-27	2021-05-27 16:28
JFEWMap-2021-07-01	2021-07-02 15:42
JFEWMap-2021-07-06	2021-07-06 15:40
JFEWMap-2021-07-06	2021-07-06 16:41
JFEWMap-2021-07-13	2021-07-14 16:47
JFEWMap-2021-07-15	2021-07-16 15:47
JFEWMap-2021-07-21	2021-07-21 16:39
JFEWMap-2021-07-27	2021-07-27 17:12
JFEWMap-2021-08-03	2021-08-06 10:39
JFEWMap-2021-08-06	2021-08-06 10:10
JFEWMap-2021-08-13	2021-08-16 16:29
JFEWMap-2021-08-20	2021-08-20 15:20
JFEWMap-FINAL	2021-08-25 11:26

Using filenames to keep a history…

When using Git, you can now drop some terrible habits, like…



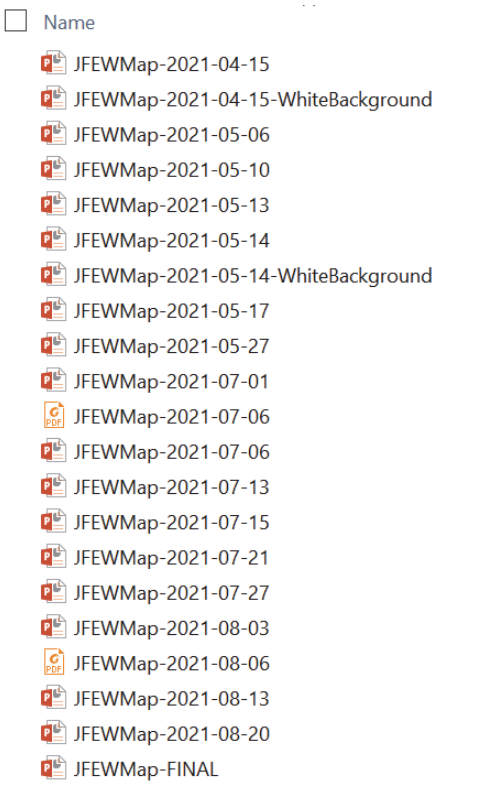
Name	Date modified
JFEWMap-2021-04-15	2021-04-16 15:53
JFEWMap-2021-04-15-WhiteBackground	2021-04-16 15:55
JFEWMap-2021-05-06	2021-05-07 19:00
JFEWMap-2021-05-10	2021-05-10 17:25
JFEWMap-2021-05-13	2021-05-13 17:14
JFEWMap-2021-05-14	2021-05-14 15:41
JFEWMap-2021-05-14-WhiteBackground	2021-05-14 15:56
JFEWMap-2021-05-17	2021-05-17 14:50
JFEWMap-2021-05-27	2021-05-27 16:28
JFEWMap-2021-07-01	2021-07-02 15:42
JFEWMap-2021-07-06	2021-07-06 15:40
JFEWMap-2021-07-06	2021-07-06 16:41
JFEWMap-2021-07-13	2021-07-14 16:47
JFEWMap-2021-07-15	2021-07-16 15:47
JFEWMap-2021-07-21	2021-07-21 16:39
JFEWMap-2021-07-27	2021-07-27 17:12
JFEWMap-2021-08-03	2021-08-06 10:39
JFEWMap-2021-08-06	2021-08-06 10:10
JFEWMap-2021-08-13	2021-08-16 16:29
JFEWMap-2021-08-20	2021-08-20 15:20
JFEWMap-FINAL	2021-08-25 11:26

Using filenames to keep a history…

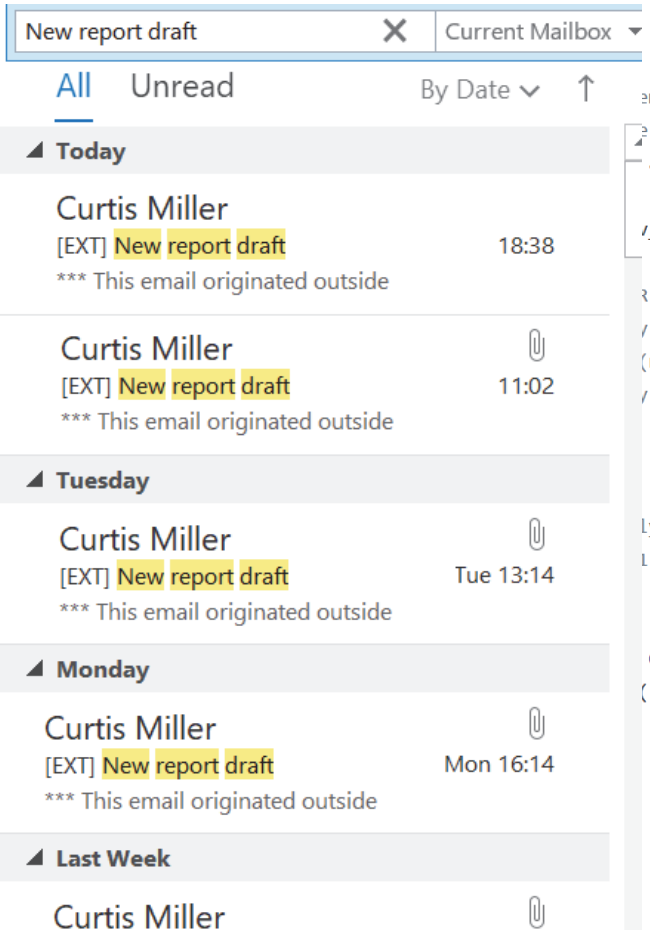
```
215   # Function implemented in Rcpp for speed
216   # It accepts the matrix y and the evaluation of the kernel function stored
217   # in kern_vals, and returns a vector containing estimated long-run variances
218   # at points t
219   sigma <- get_lrv_vec_cpp(y, kern_vals, max_l)
220
221   # "Equivalent" R code (slower)
222   # covs <- sapply(1:n, function(t) {
223   #     sapply(0:(n - 1), function(l) {
224   #         mean(y[1:(n - l),t]*y[(1 + l):n,t])
225   #     })
226   # })
227   #
228   # sigma <- sapply(2:(n - 2), function(t) {
229   #     covs[0 + 1, t] + 2 * sum(vkernel(1:(n - 1)/h) * covs[1:(n - 1), t])
230   # })
231
232   if (any(sigma < 0)) {
233     warning(paste("A negative variance was computed! This may be due to a bad",
234                   "kernel being chosen."))
235   }
236
237   sigma
```

Commenting out/keeping old useless code…

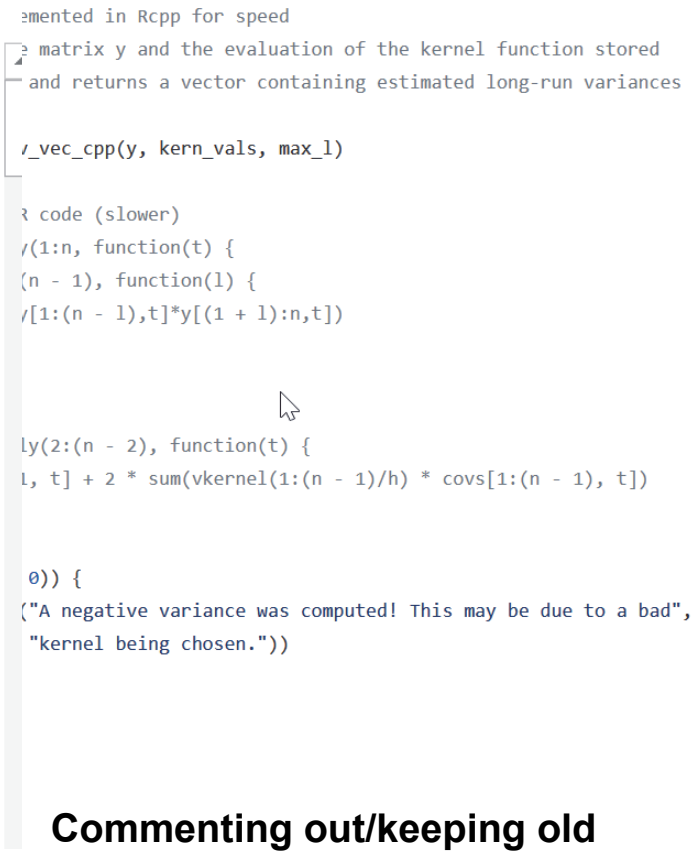
When using Git, you can now drop some terrible habits, like…



Using filenames to keep a history…

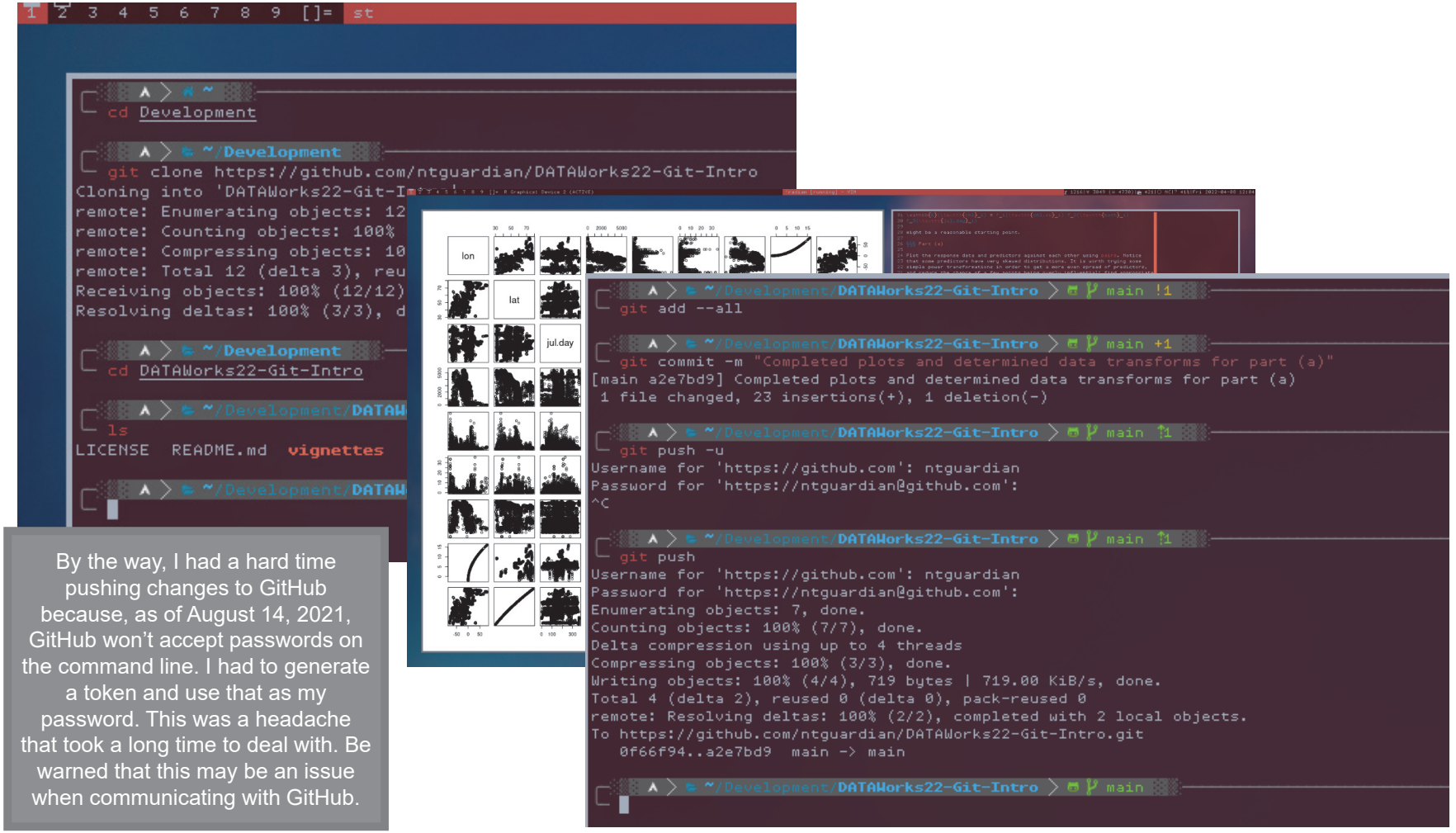


Using e-mail to manage collaboration/records…



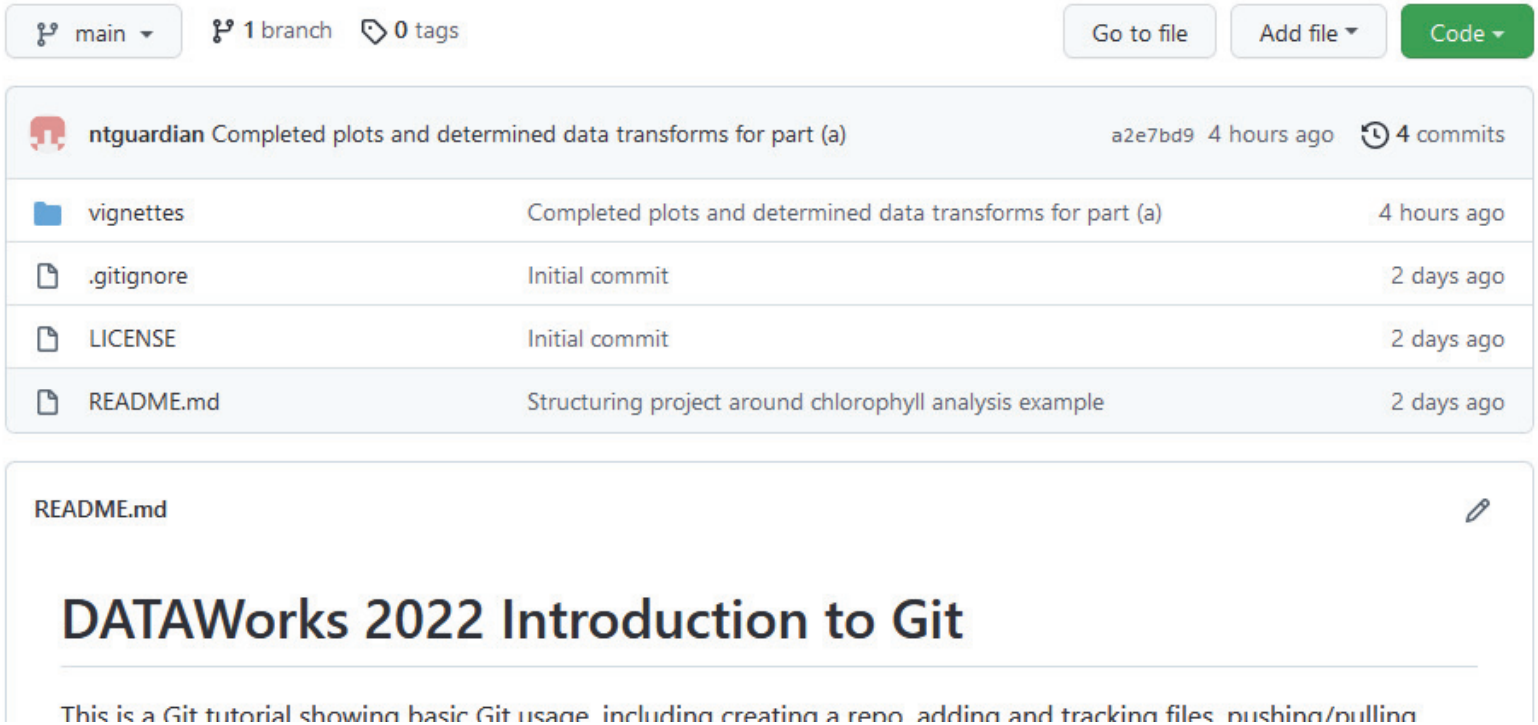
Commenting out/keeping old useless code…

A collaborator just did some stuff on their own computer...

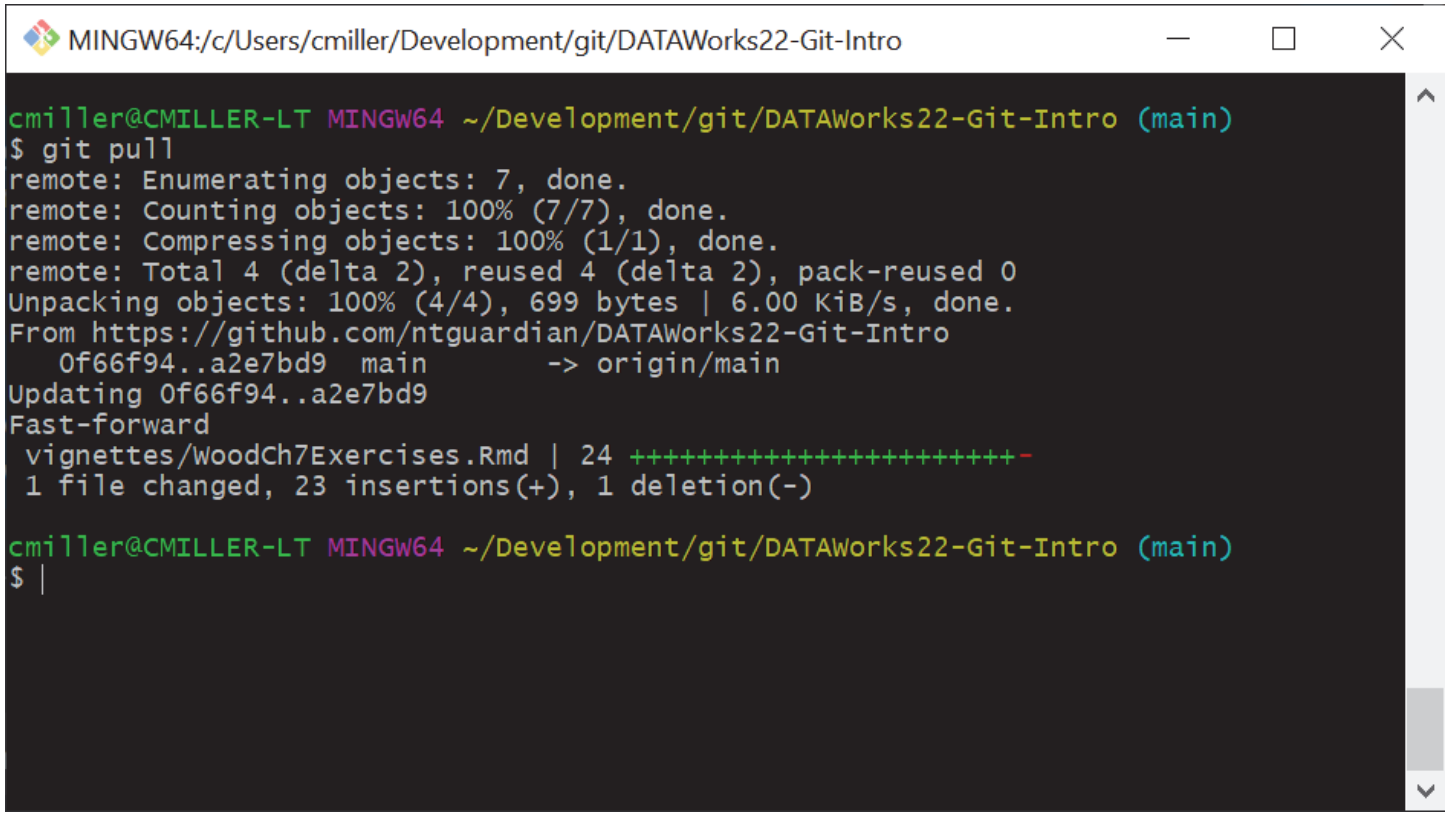


By the way, I had a hard time pushing changes to GitHub because, as of August 14, 2021, GitHub won't accept passwords on the command line. I had to generate a token and use that as my password. This was a headache that took a long time to deal with. Be warned that this may be an issue when communicating with GitHub.

We can see these changes on GitHub and now need to get them locally



Use `git pull` to bring the local repo on your hard drive up-to-date



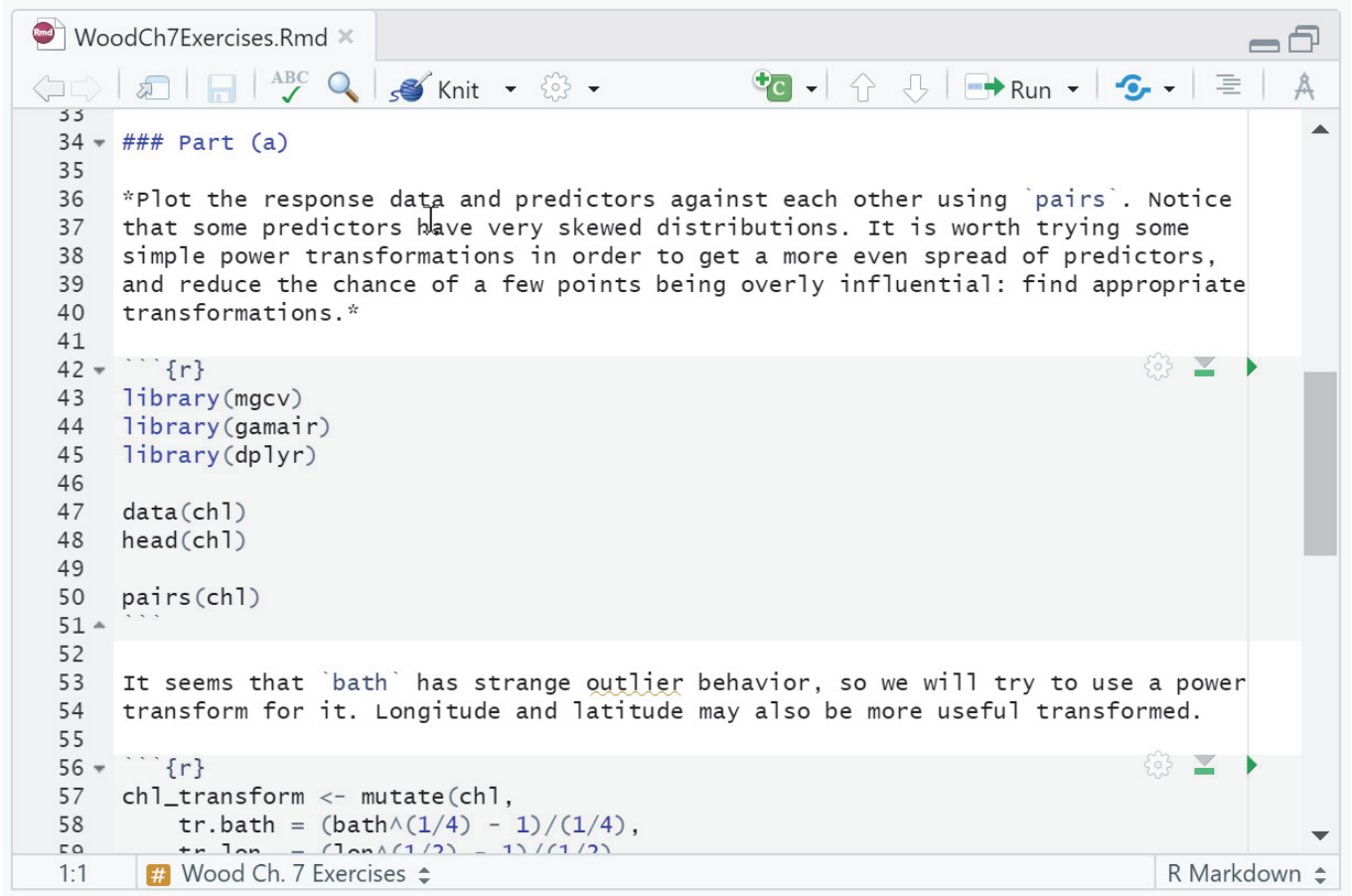
Use `git pull` to bring the local repo on your hard drive up-to-date

This command downloads all changes and brings the files up-to-date (it will fail if you have local changes)

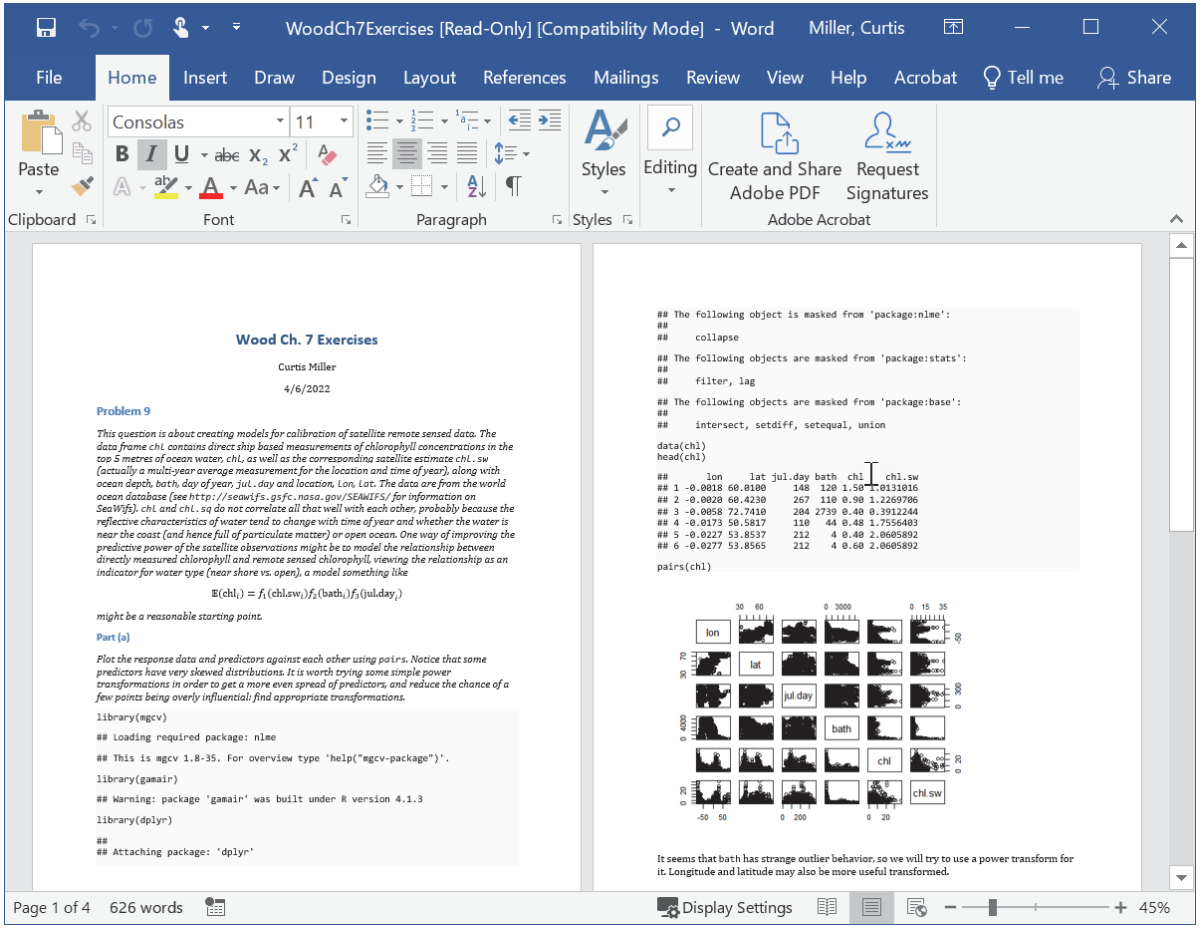
This is a summary of what's changed and how much

You should run this at least in the morning before starting work to make sure you're up-to-date. Run again if someone just made changes needed immediately.

Now those changes are available to work with

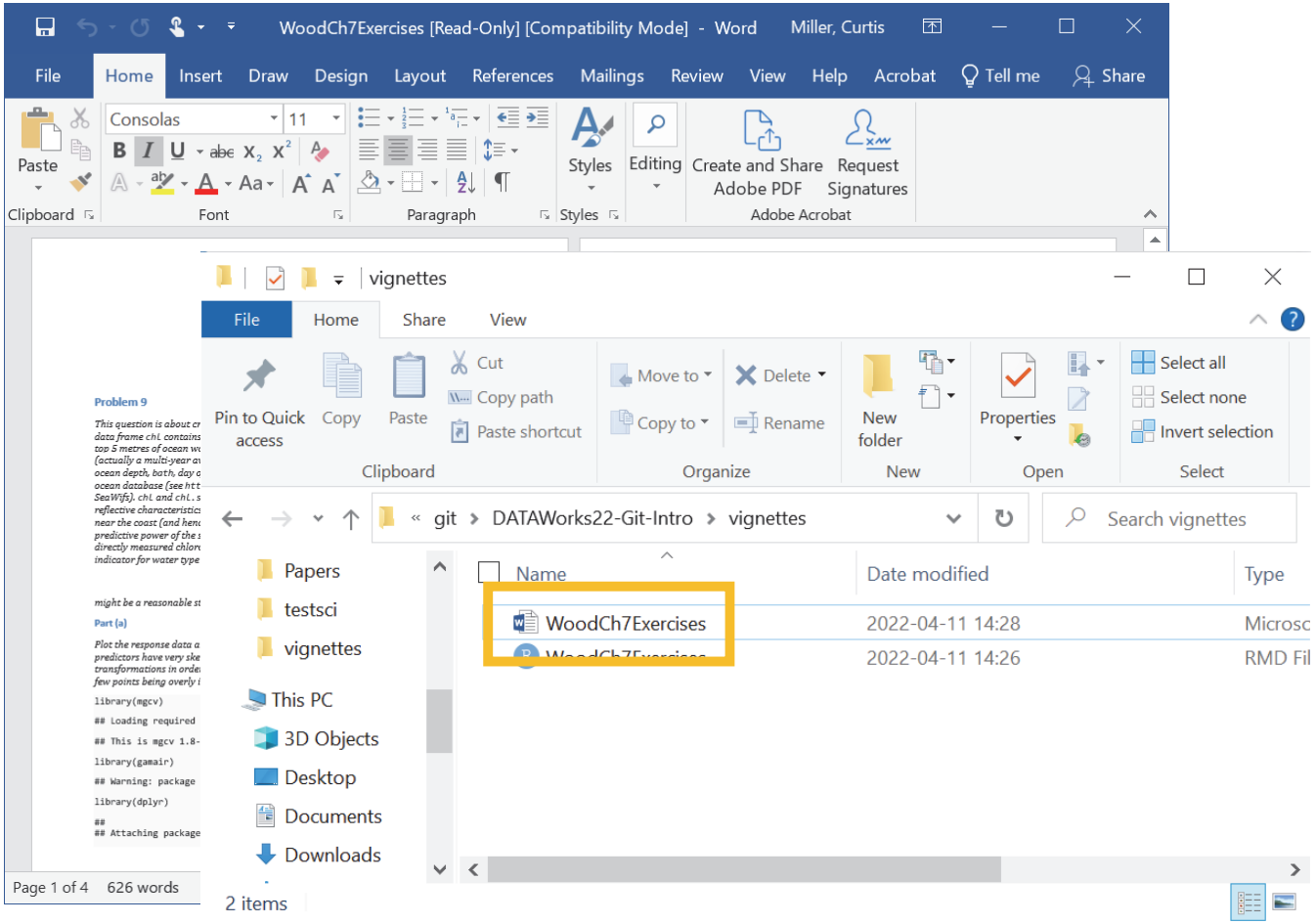


We can convert that Rmd file into a Word file...

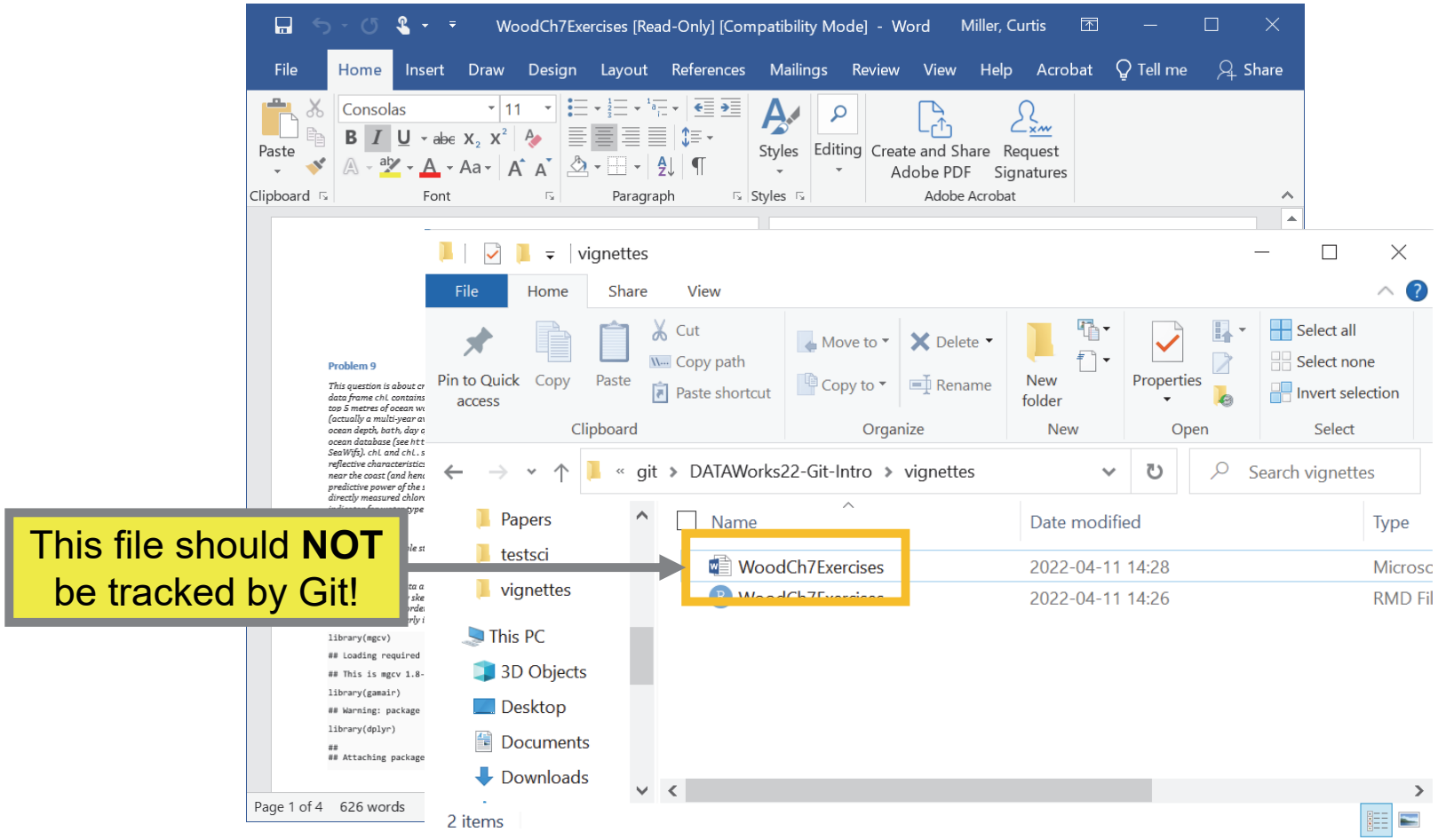


R Markdown is a format that can be translated into many other formats. Word is one of them, in addition to HTML and PDF.

… that then appears in the directory with the original file…



... but we *don't* want to track this file



This file should **NOT** be tracked by Git!

Not every file in a repository should be tracked

TRACK

Plain text files (txt, R, Rmd, py, …)

Files directly edited

Essential files that are hard to generate

AVOID

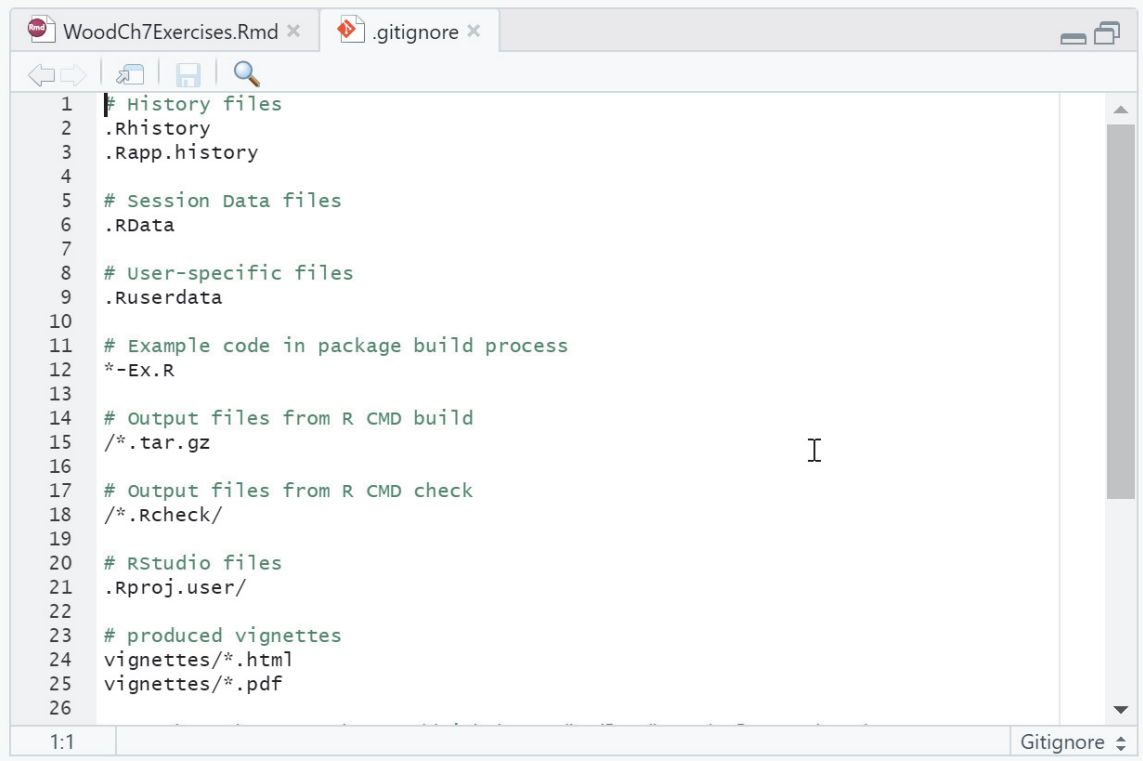
Binary files (exe, docx, xlsx)

Files generated by other files

Irrelevant files

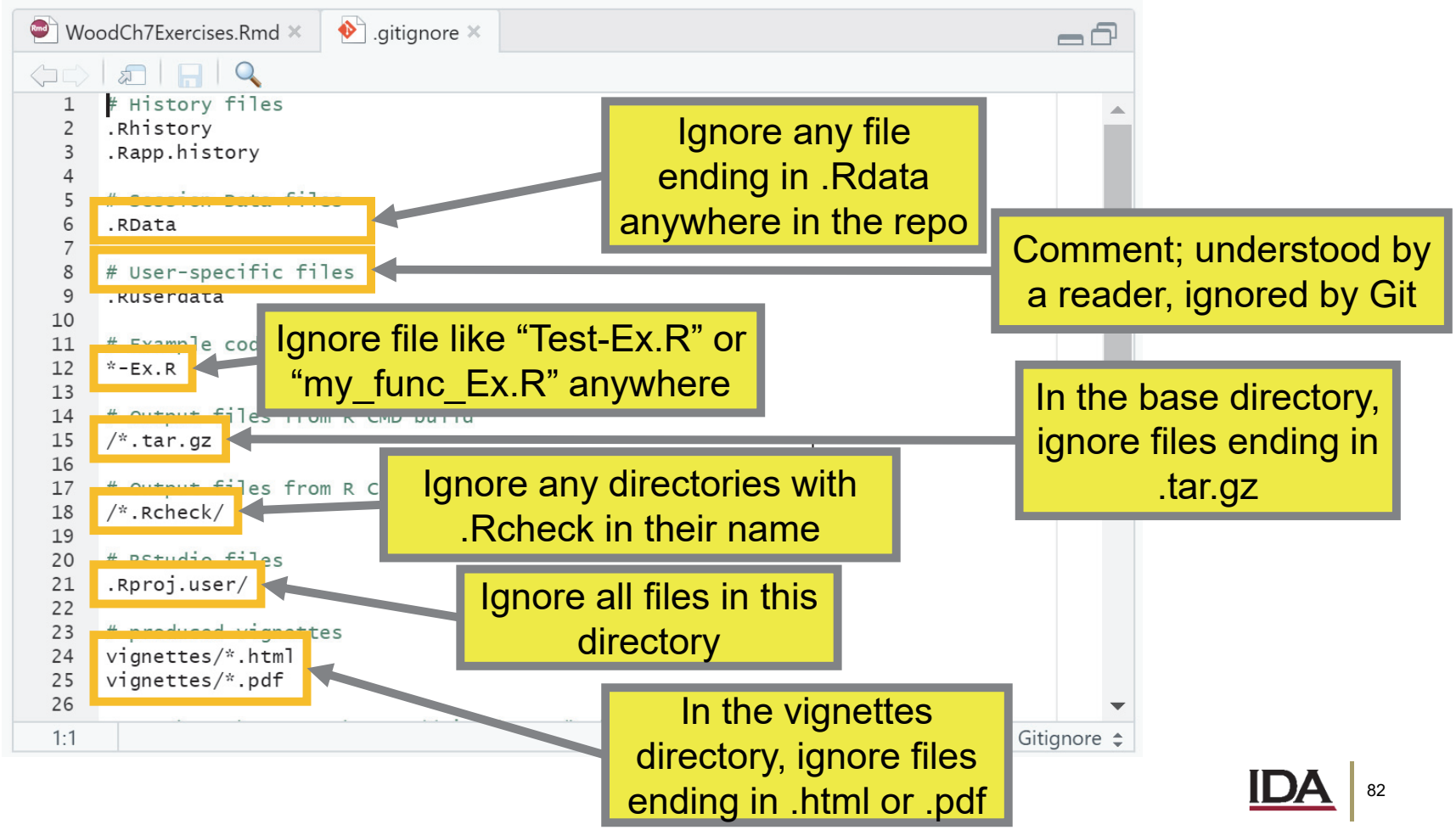
** These are guidelines and violating them may be reasonable*

Use the .gitignore file to describe files Git should ignore when adding files to track changes



- .gitignore describes files that, by default, should not be tracked in commits
- .gitignore is a plain text file containing descriptions of files that should be ignored
- File names and locations can be entered manually
- Pattern can also be used to describe classes of files to be ignored; if a file name matches the pattern the file will be ignored
- See https://git-scm.com/docs/gitignore for more information on writing a .gitignore file

GitHub already made a .gitignore file with defaults



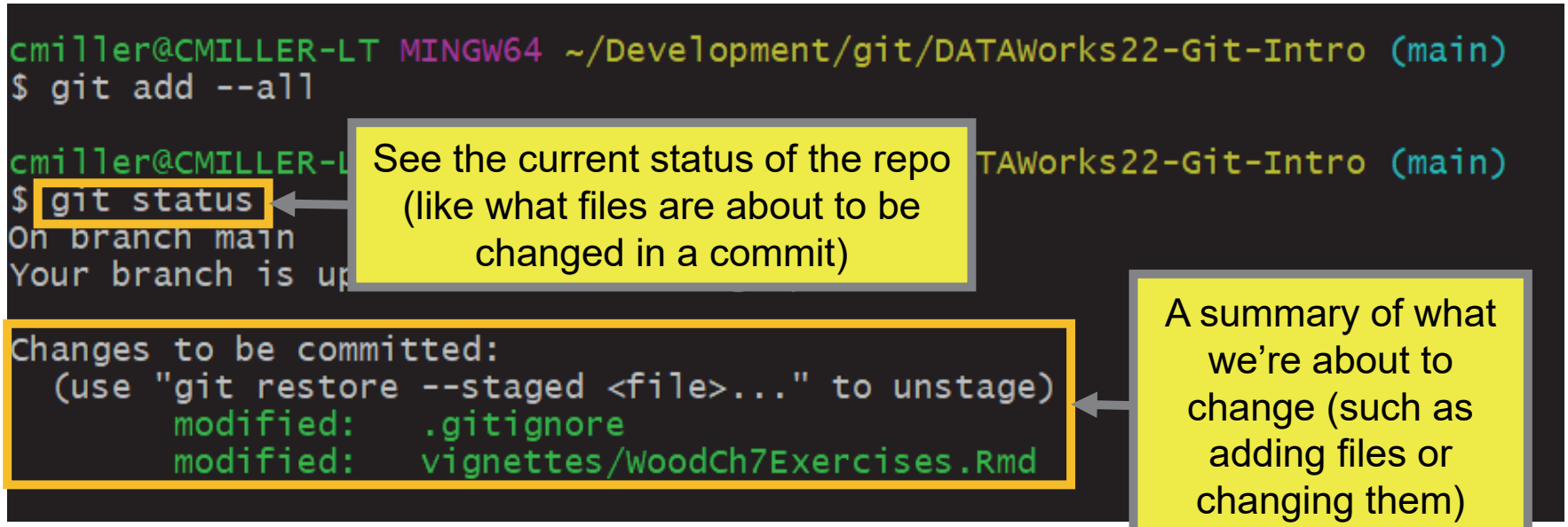
Now just add a line to ignore Word (.docx) files

```
23  # produced vignettes
24  vignettes/*.html
25  vignettes/*.pdf
26  vignettes/*.docx
27
```

`git status` shows us what changes we are about to commit

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        modified:   vignettes/WoodCh7Exercises.Rmd
```

Notice that our Word file was not added with `git add --all`

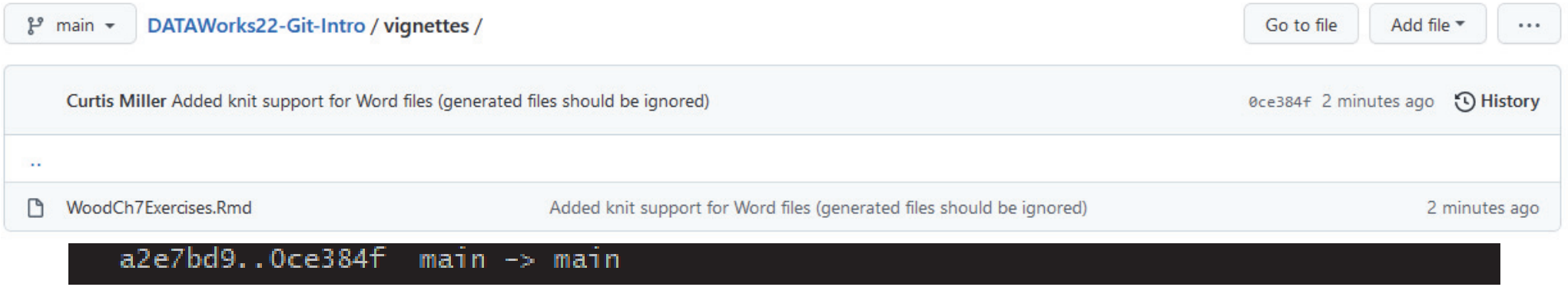


It did not add the Word file when we committed the changes...

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git commit -m "Added knit support for Word files (generated files should be ignored
)"
[main 0ce384f] Added knit support for Word files (generated files should be ignored)
 2 files changed, 4 insertions(+), 1 deletion(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 566 bytes | 283.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ntguardian/DATAWorks22-Git-Intro.git
   a2e7bd9..0ce384f  main -> main
```

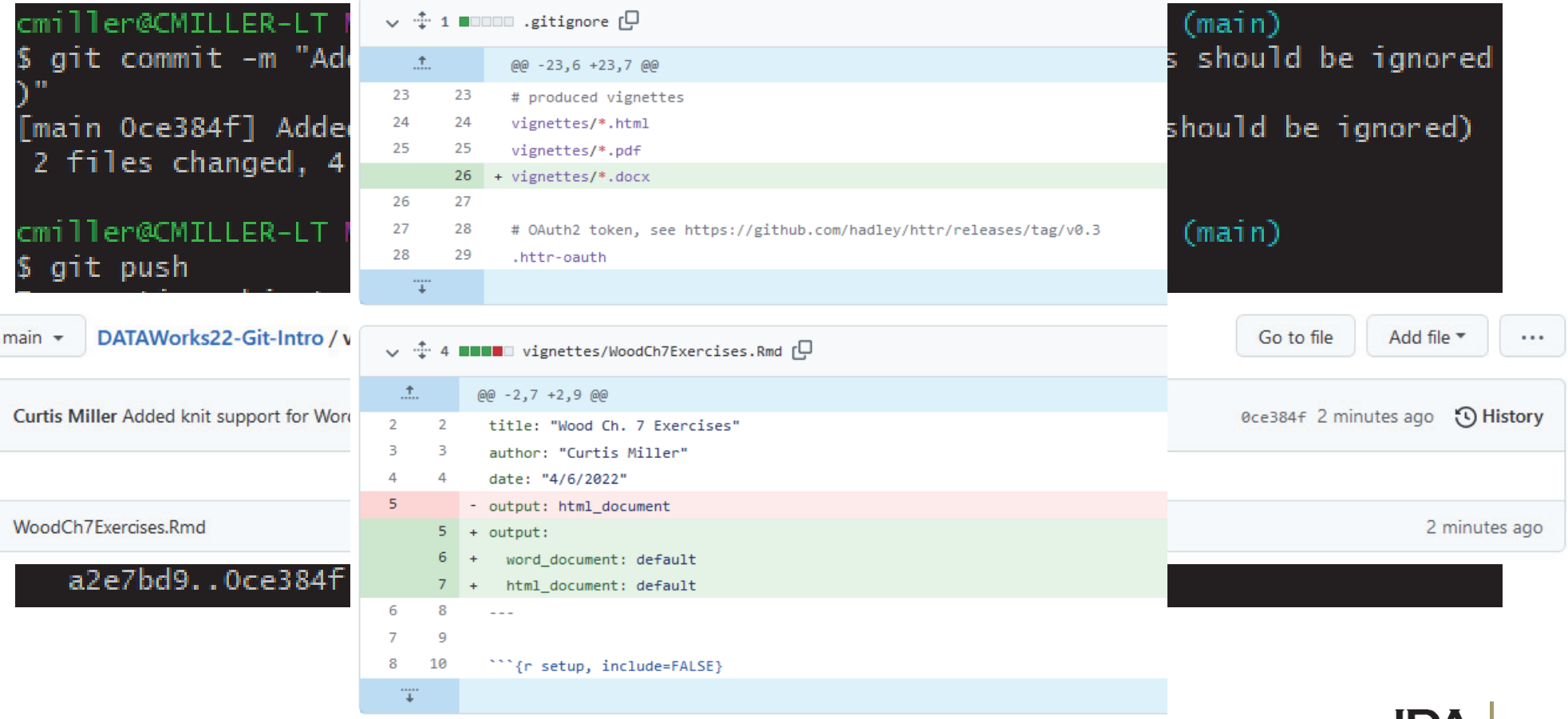
... Or pushed them to GitHub...

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git commit -m "Added knit support for Word files (generated files should be ignored
)"
[main 0ce384f] Added knit support for Word files (generated files should be ignored)
 2 files changed, 4 insertions(+), 1 deletion(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git push
```




```
   a2e7bd9..0ce384f  main -> main
```

… But it did track the other changes



Basic change tracking commands

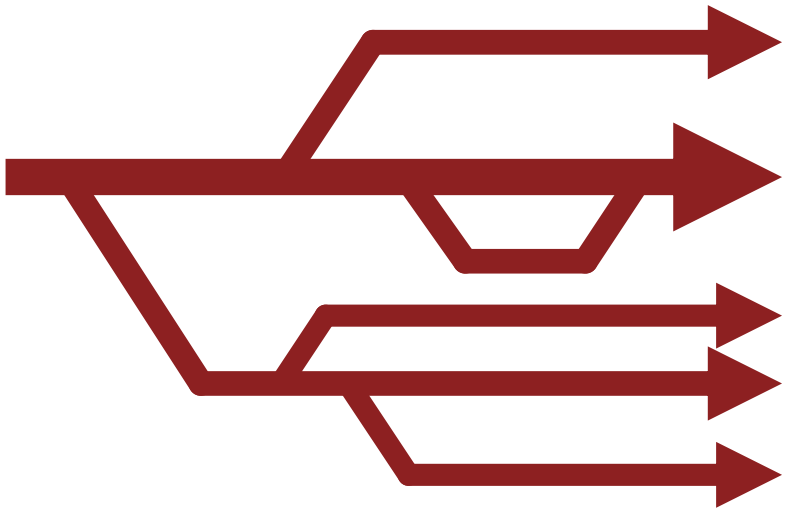
Command	Description	Examples
`git add`	Add changes to (or brand new) files to a staged commit	`git add myfile.txt` `git add --all`
`git commit`	Commit changes	`git commit –m “Message”`
`git pull`	For repos hosted remotely (such as GitHub), update repo with changes from host	`git pull`
`git push`	For repos hosted remotely (such as GitHub), send changes to host	`git push`
`git rm`	Remove a file from tracking	`git rm myfile.txt`
`git reset`	Git’s “undo” command	`git reset --hard  # CAUTION: Deletes changes`

`git diff`	Compare commits and branches	`git diff HEAD~ HEAD`
		`git diff HEAD~ HEAD myfile.txt`
		`git diff onebranch otherbranch`

`git status`	Show the status of a repo (such as what will be changed in a commit)	`git status`
`git log`	See commit history	`git log`
`git checkout`	Retrieve data from commits	`git checkout identifier`

Branches

Branches allow multiple versions of files to exist and be tracked simultaneously

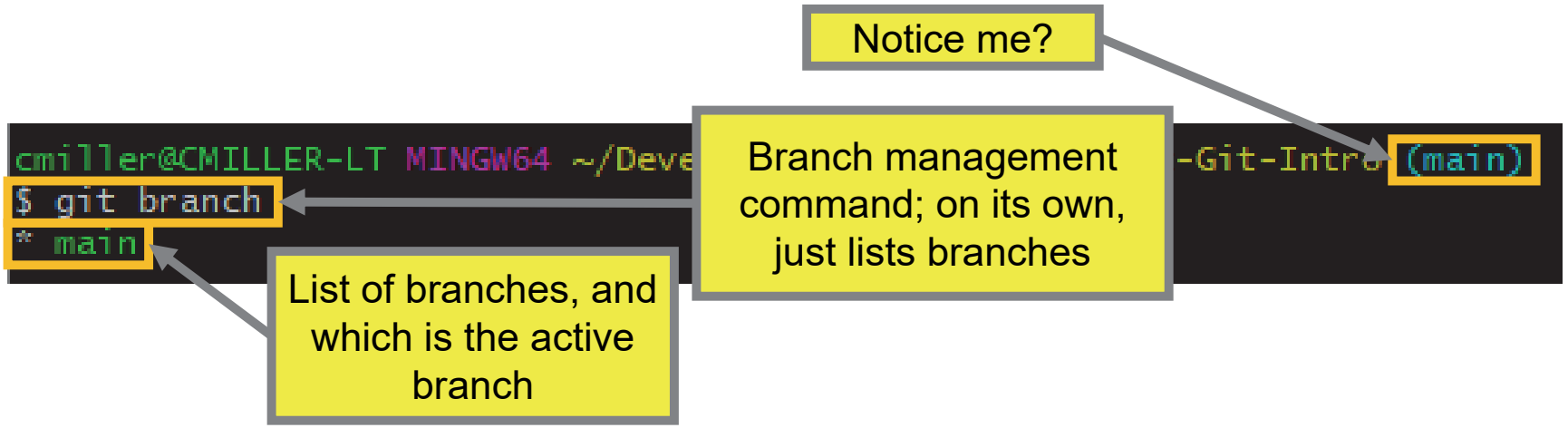


- Commits represent moving forward and backward in time while branches allow for differences between files to exist simultaneously
- Git can track the difference between branches and propagate a change that affects both branches while maintaining what makes them distinct
- Changes made in one branch can be merged into another branch
- Branching provides a means for collaboration; each collaborator creates their own private branch to do their work
- Branching allows collaborators to simultaneously maintain different versions of a product

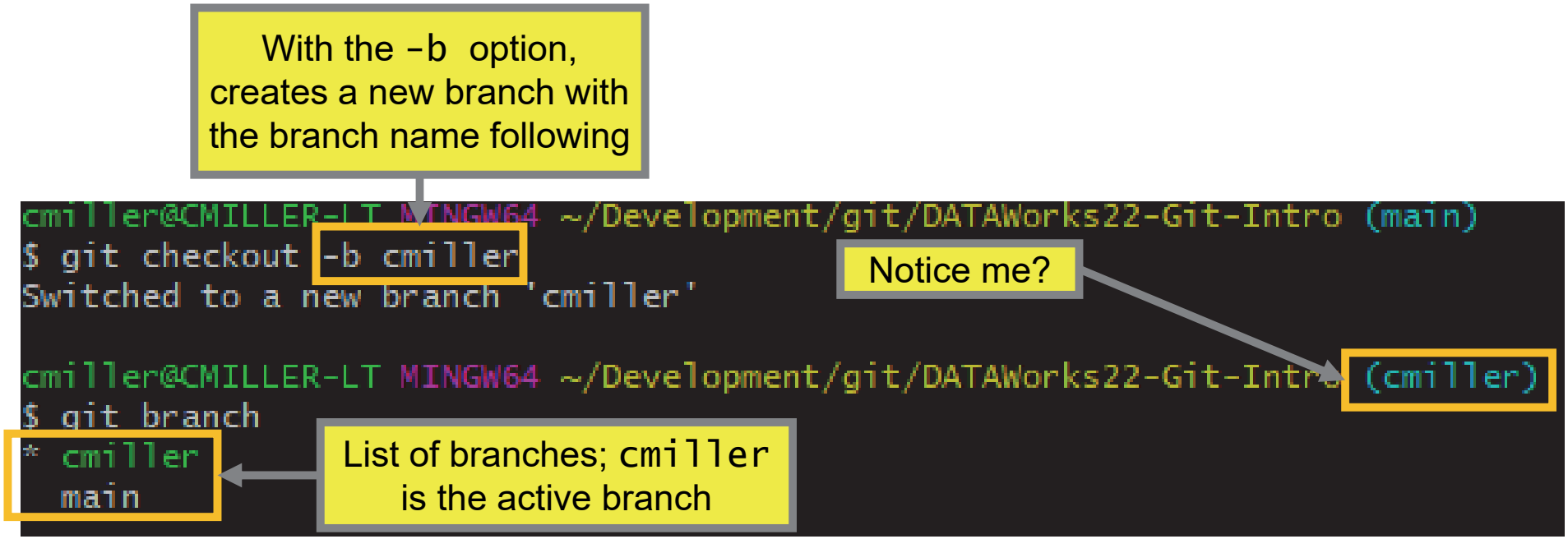
What branches do we start out with?

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git branch
* main
```

The branch that best represents the project's most stable state is the main branch



Let's create a personal branch



When using Git with a team, avoid editing the main branch directly. Make edits on a personal branch freely. Changes to the main branch should be done via merging changes from other branches most of the time (we will discuss branch merging later). Only when the team is ready to "accept" edits should the main branch be changed.

Now work primarily with the personal branch

WoodCh7Exercises.Rmd

Knit

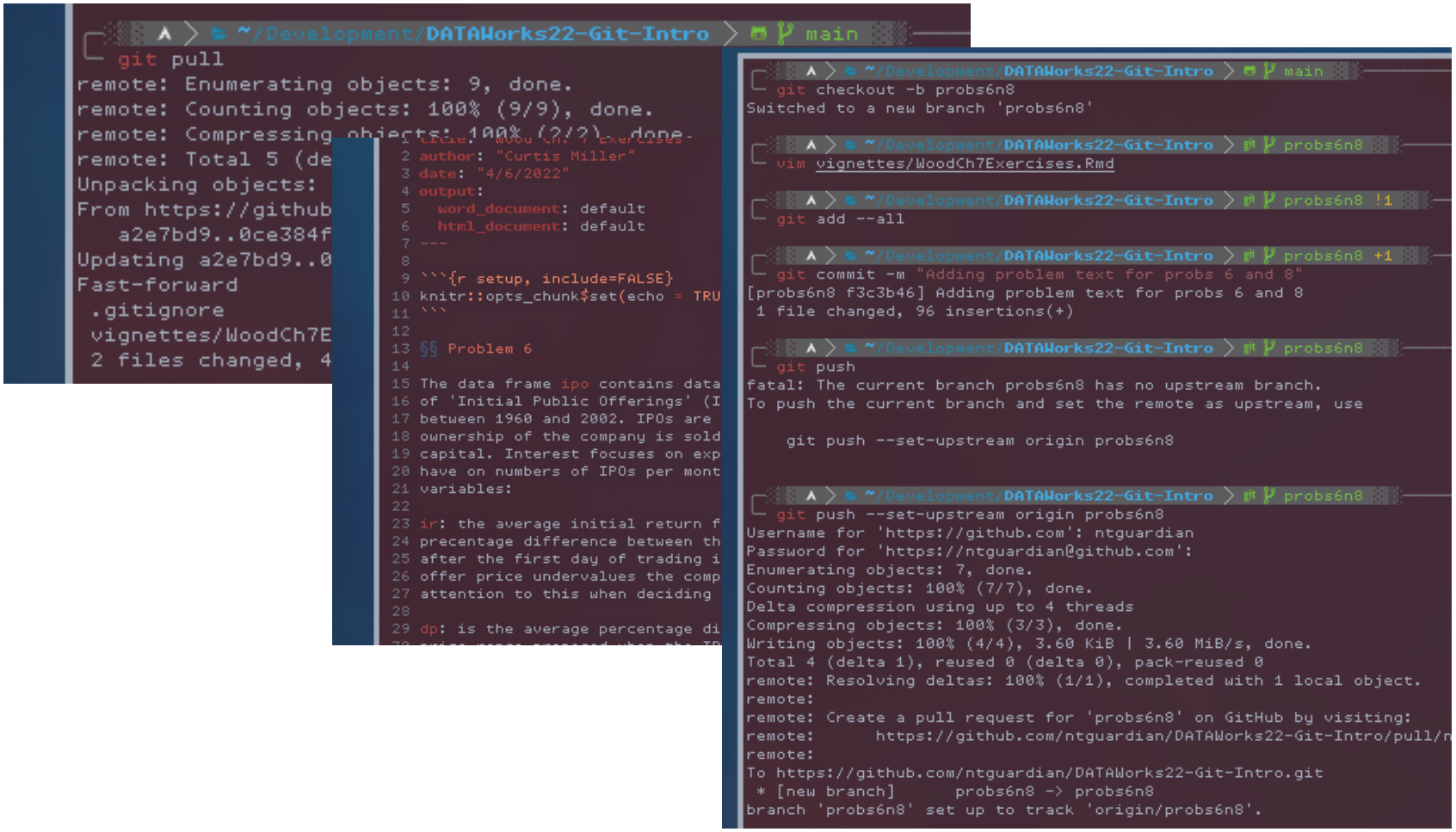
```
77
78  Let's consider the range of the response variable:
79
80  ```{r}
81  range(chl_transform$chl)
82  ```
83
84  The gamma distribution makes for an attractive potential distribution family,
85  due to being non-negative, but unfortunately there are concentrations of zero. A
86  quick fix would be to add a small value to the zeroes, reflecting the fact that
87  such a concentration likely is an issue with rounding.
88
89  ```{r}
90  chl_transform <- mutate(chl_transform, chl = ifelse(chl == 0, 0.001, chl))
91  ```
92
93  Now fit a GAM.
94
95  ```{r}
96  chl_fit <- gam(chl ~
97                   s(tr.lon, k = 60, bs = 'cr') +
98                   s(tr.lat, k = 60, bs = 'cr') +
99                   s(tr.bath, k = 60, bs = 'cr') +
100                  s(jul.day, k = 120, bs = 'cr') +
101                  s(chl.sw, k = 60, bs = 'cr'),
102                data = chl_transform,
103                family = Gamma(link = "log"),
104                method = "REML"
105              )
106 ```
107
108 Sadly the latitude and longitude are not interacted due to time to compute
109 solution being too long.
110
111 ### Part (c)
112
113 *In this sort of modeling the aim is to improve on simply using the satellite
114 measurements as predictions of the direct measurements. Given the large number
115 of data, it is easy to end up using rather complex models after quite a lot of
116 examination of the data. It is therefore important to check that the model is
```

77:1 Part (b)

MINGW64:/c/Users/cmiller/Development/git/DATAWorks22-Git-Intro

```
[cmiller ca00e1b] Attempting part (b) using log link/gamma family
 1 file changed, 28 insertions(+), 3 deletions(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git commit -m "Attempted changes to GAM only to return to Gamma, but different form"
[cmiller 01361ea] Attempted changes to GAM only to return to Gamma, but different form
 1 file changed, 5 insertions(+), 19 deletions(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git checkout HEAD~ vignettes/WoodCh7Exercises.Rmd
Updated 1 path from fc1a383

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git commit -m "Restore old GAM model and start working on train/test split"
[cmiller ed2d1fe] Restore old GAM model and start working on train/test split
 1 file changed, 32 insertions(+), 2 deletions(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$
```

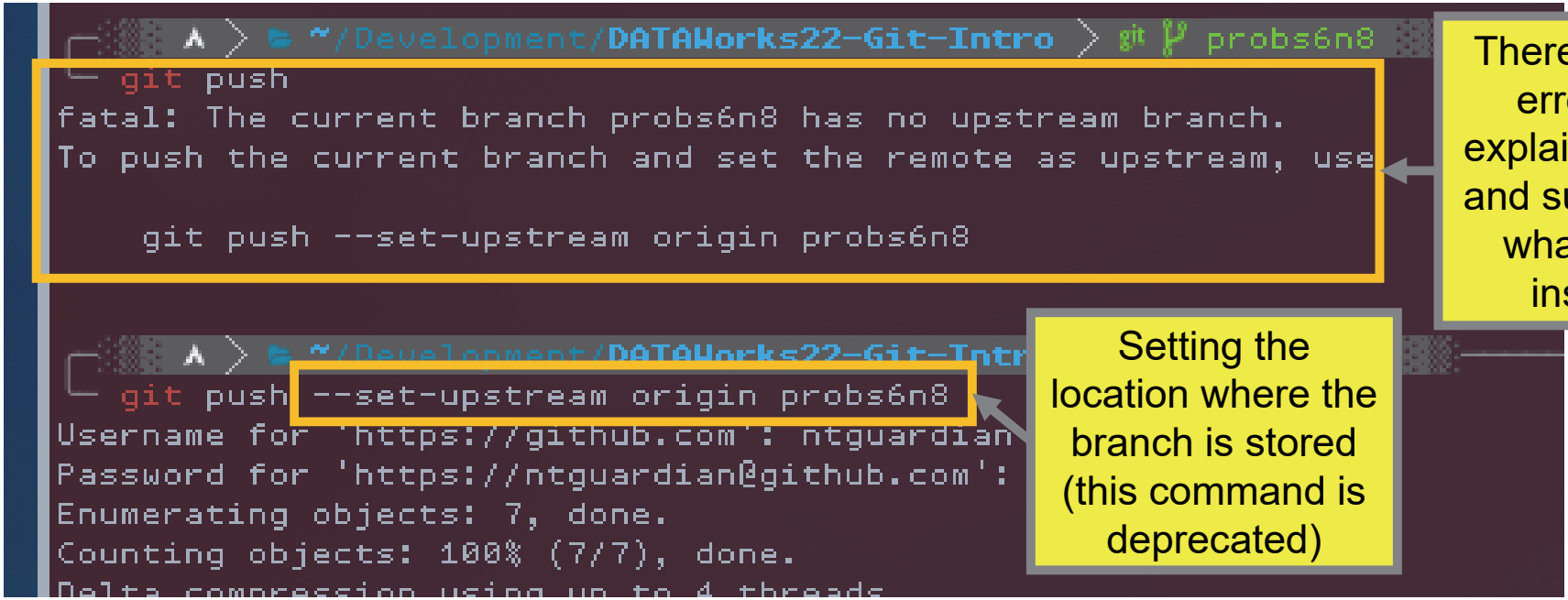
As we were doing work, a “collaborator” did some work too



Note how to add a branch to the external repo

```
~/Development/DATAWorks22-Git-Intro  git probs6n8
git push
fatal: The current branch probs6n8 has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin probs6n8


~/Development/DATAWorks22-Git-Intro  git probs6n8
git push --set-upstream origin probs6n8
Username for 'https://github.com': ntguardian
Password for 'https://ntguardian@github.com':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
```

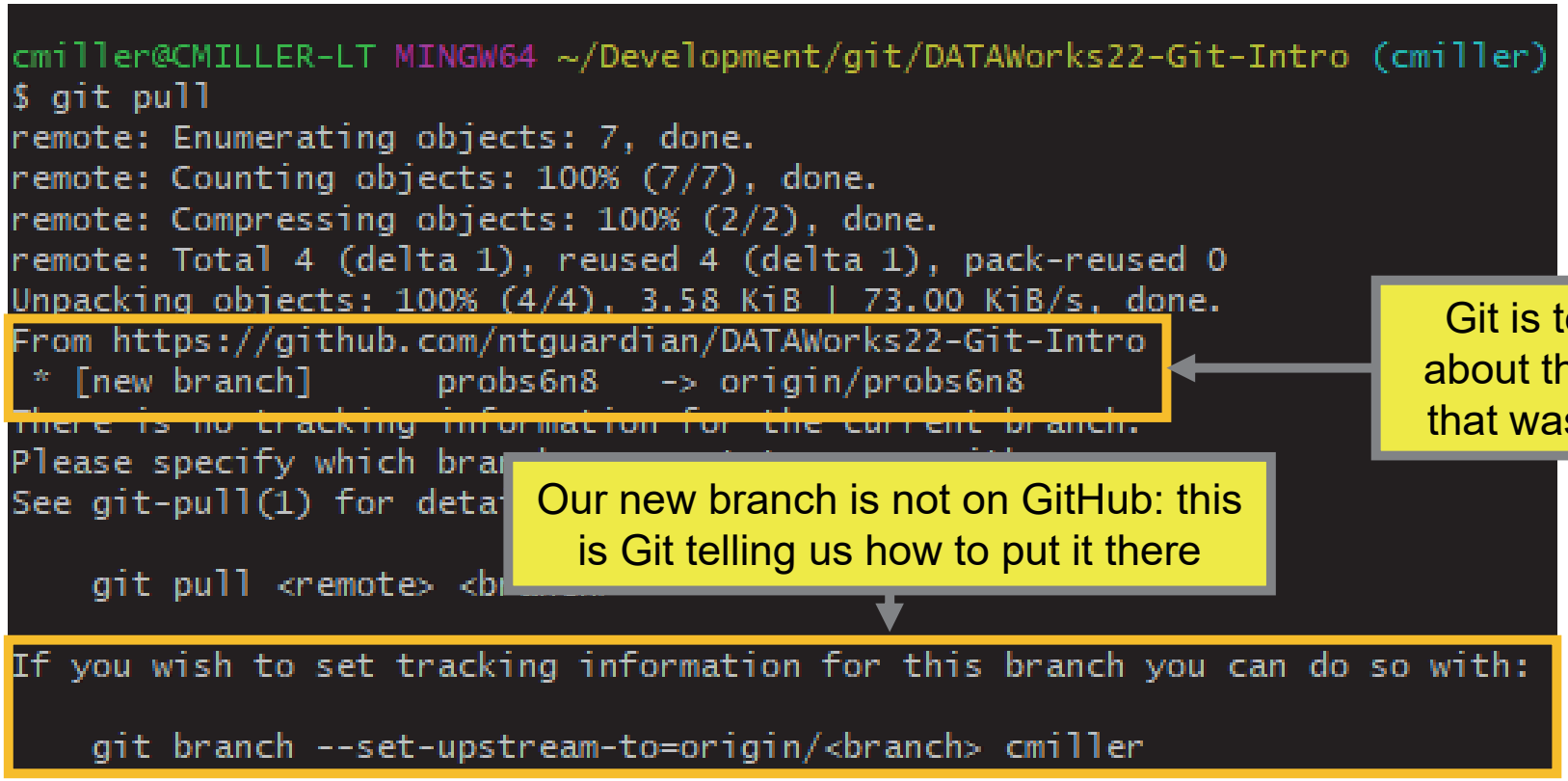
Note how to add a branch to the external repo



We should sync our local copy of the repo now

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), 3.58 KiB | 73.00 KiB/s, done.
From https://github.com/ntguardian/DATAWorks22-Git-Intro
 * [new branch]      probs6n8   -> origin/probs6n8
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=origin/<branch> cmiller
```

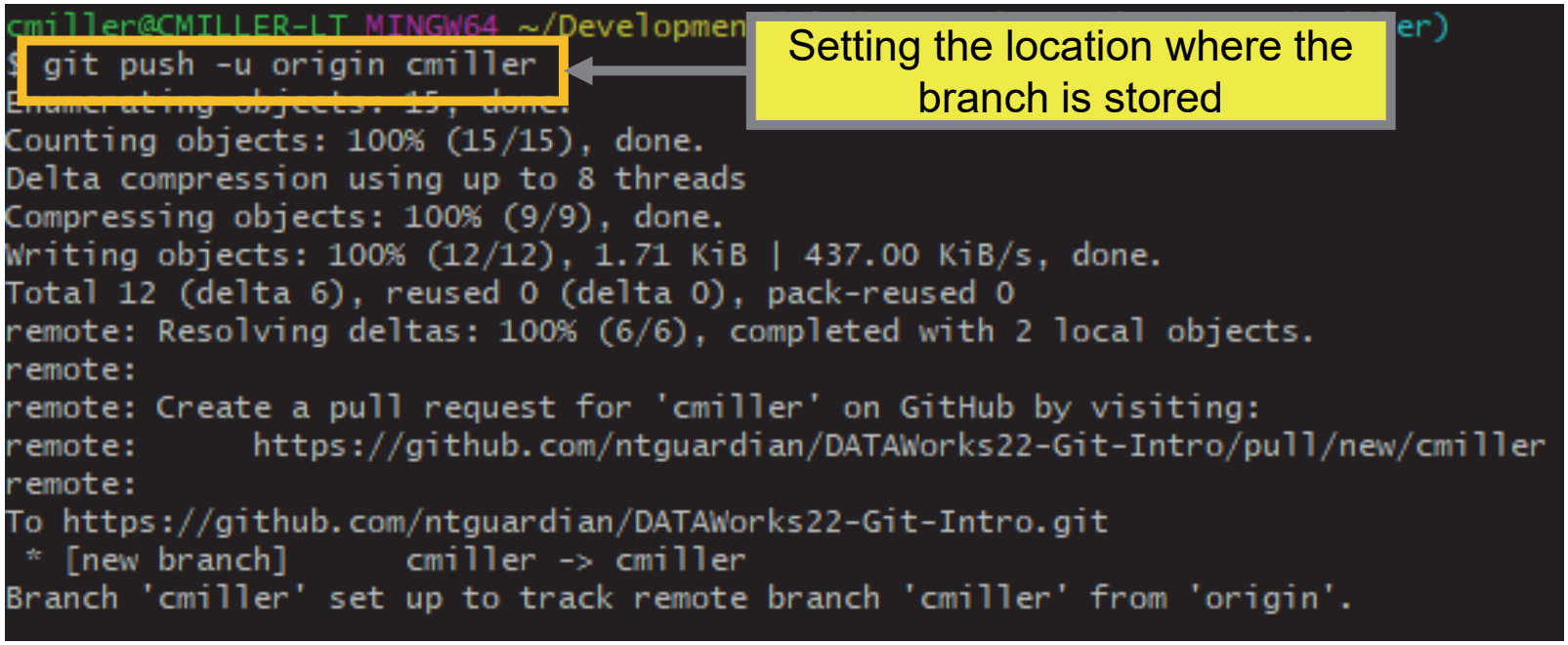
Read the resulting messages



We can use the following command to track our branch remotely

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git push -u origin cmiller
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.71 KiB | 437.00 KiB/s, done.
Total 12 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 2 local objects.
remote:
remote: Create a pull request for 'cmiller' on GitHub by visiting:
remote:      https://github.com/ntguardian/DATAWorks22-Git-Intro/pull/new/cmiller
remote:
To https://github.com/ntguardian/DATAWorks22-Git-Intro.git
 * [new branch]      cmiller -> cmiller
Branch 'cmiller' set up to track remote branch 'cmiller' from 'origin'.
```

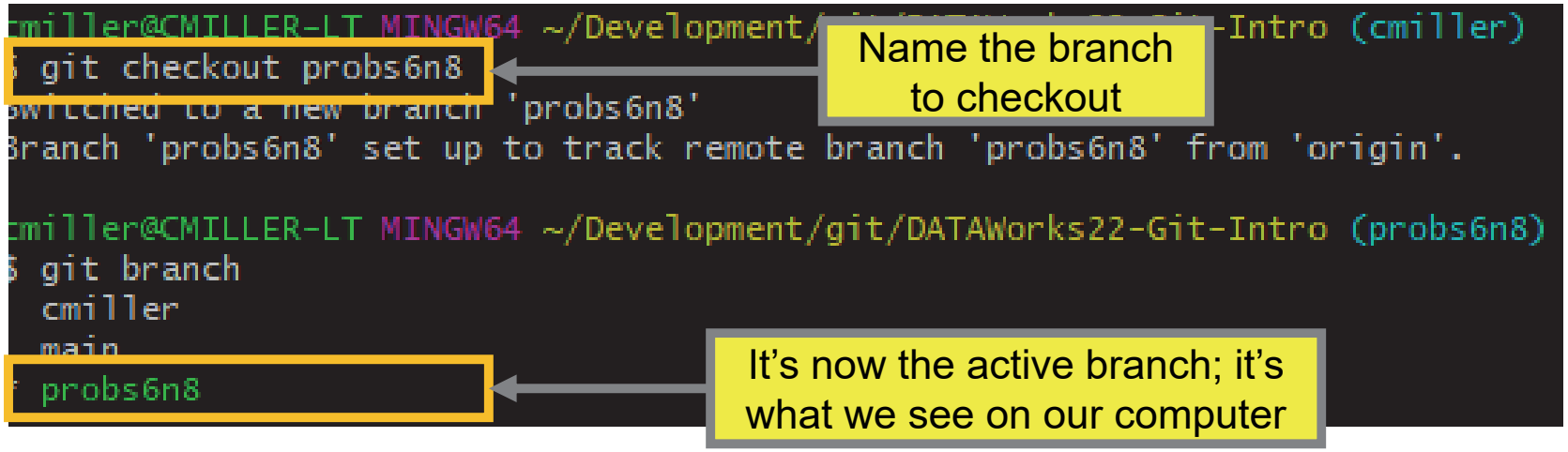
We can use the following command to track our branch remotely



To see the new branch, use `git checkout`

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git checkout probs6n8
Switched to a new branch 'probs6n8'
Branch 'probs6n8' set up to track remote branch 'probs6n8' from 'origin'.

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git branch
  cmiller
  main
* probs6n8
```

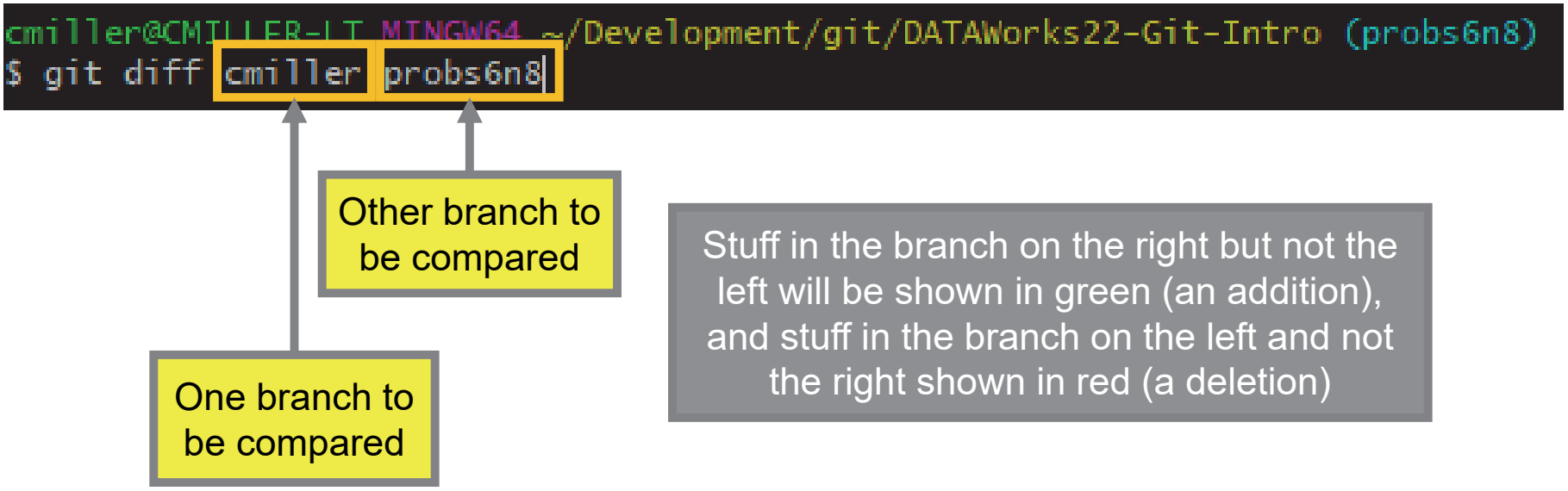
To see the new branch, use `git checkout`



To compare branches, use `git diff`

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git diff cmiller probs6n8
```

To compare branches, use `git diff`



We can now compare what's in the branches

```
diff --git a/vignettes/WoodCh7Exercises.Rmd b/vignettes/WoodCh7Exercises.Rmd
index 9e07578..94cb527 100644
--- a/vignettes/WoodCh7Exercises.Rmd
+++ b/vignettes/WoodCh7Exercises.Rmd
@@ -11,6 +11,101 @@ output:
 knitr::opts_chunk$se
 ```


+## Problem 6
+
+*The data frame `ipo
+of 'Initial Public Of
+between 1960 and 200
```

```
Part (b)

*Using `mgcv::gam`, try modeling the data using a model of the sort suggested
*Using `mgcv::gam`, try modelling the data using a model of the sort suggested
(but with predictors transformed as in part (a)). Make sure that you use an
appropriate family. It will probably help to increase the default basis
dimensions used for smoothing, somewhat (especially for `jul.day`). Use the
`"cr"` basis to avoid excessive comput ional cost.*

Let's consider the range of the response variable:

```{r}
range(chl_transform$chl)
# Your code here
```

The gamma distribution makes for an attractive potential distribution family,
due to being non-negative, but unfortunately there are concentrations of zero. A
```

If this is too long to fit on one screen, Bash might use a program called **less** to view it; scroll up and down with the arrow keys, then press q to quit
```

“I only wanted to see which files are different, not the differences themselves.”

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git diff --name-only cmiller probs6n8
vignettes/WoodCh7Exercises.Rmd
```

“I only wanted to see which files are different, not the differences themselves.”

Now go back to our personal branch

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git checkout cmiller
Switched to branch 'cmiller'
Your branch is up to date with 'origin/cmiller'.

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$
```

A few more edits...

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git commit -m "Attempting part (c) but did not complete deviance computation o
n test set"
[cmiller 0569dd3] Attempting part (c) but did not complete deviance computation
on test set
 1 file changed, 11 insertions(+), 8 deletions(-)
```

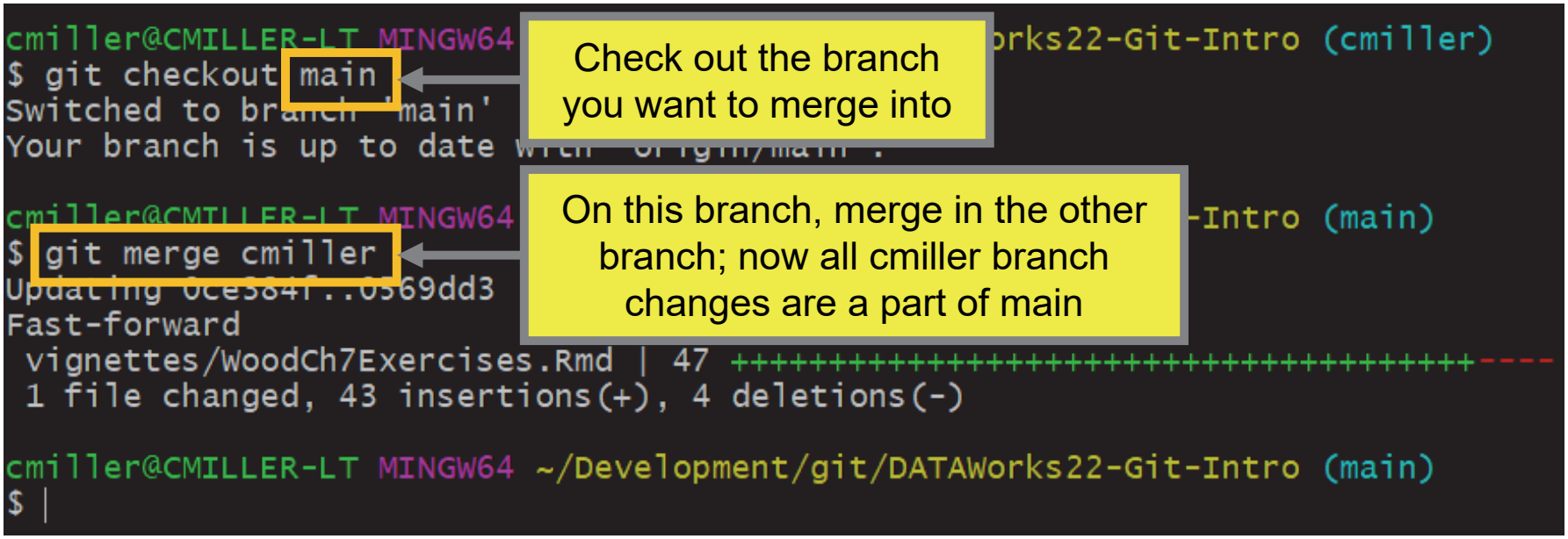
Use git merge to propagate changes made in one branch to another branch

- Eventually one may want to merge the changes made in one branch to another branch
- `git merge` will determine how the two branches should be merged together
- When using Git merge, *check out the destination branch for the merge first*, then call `git merge sourcebranch`
- The changes from the source branch will then be propagated to the destination branch
- Git may ask for a message to be written to explain the merge

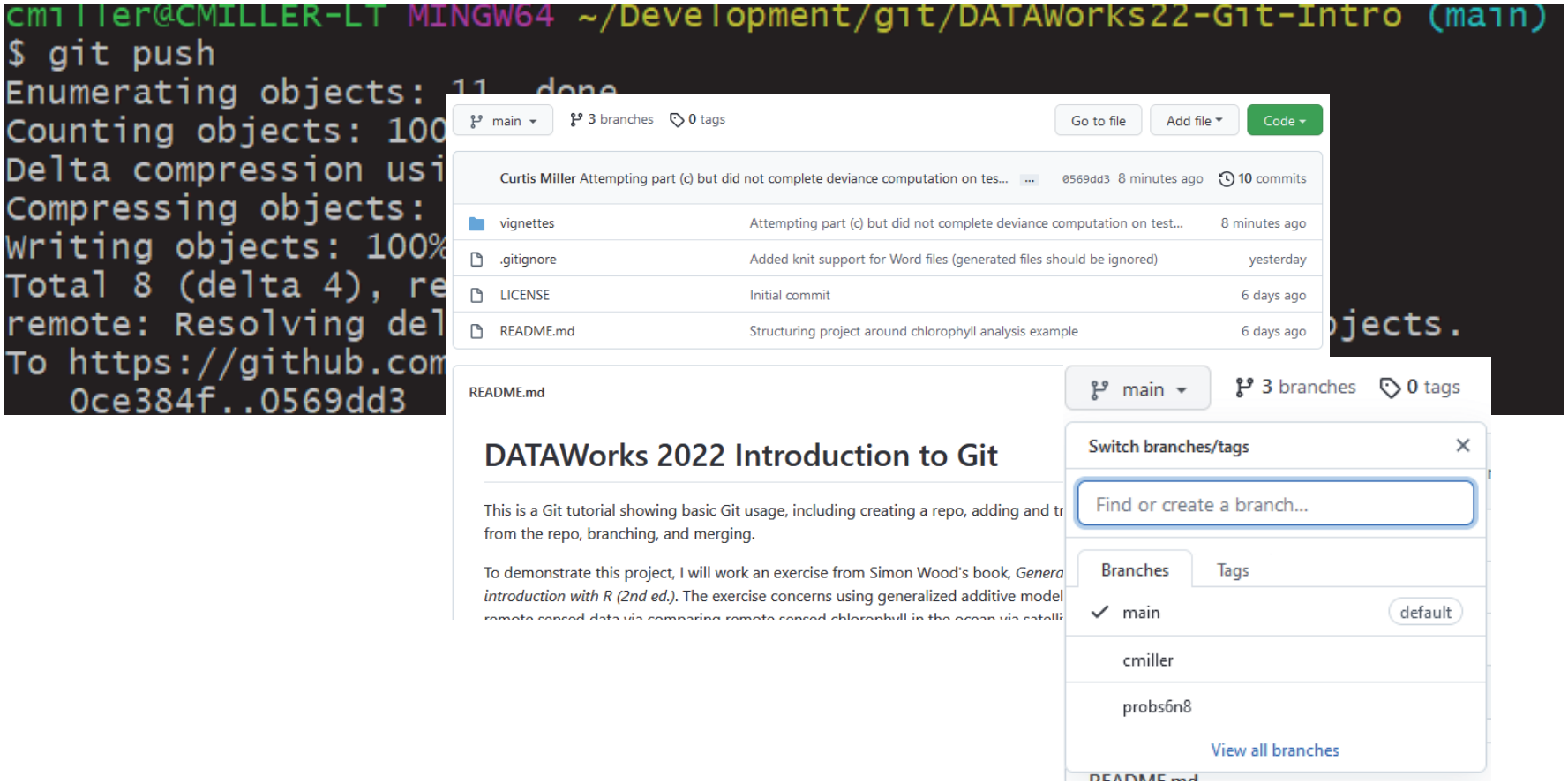
A merge may be simple...

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (cmiller)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git merge cmiller
Updating 0ce384f..0569dd3
Fast-forward
 vignettes/WoodCh7Exercises.Rmd | 47 ++++++++++++++++++++++++++++++++++++++++----
 1 file changed, 43 insertions(+), 4 deletions(-)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$
```

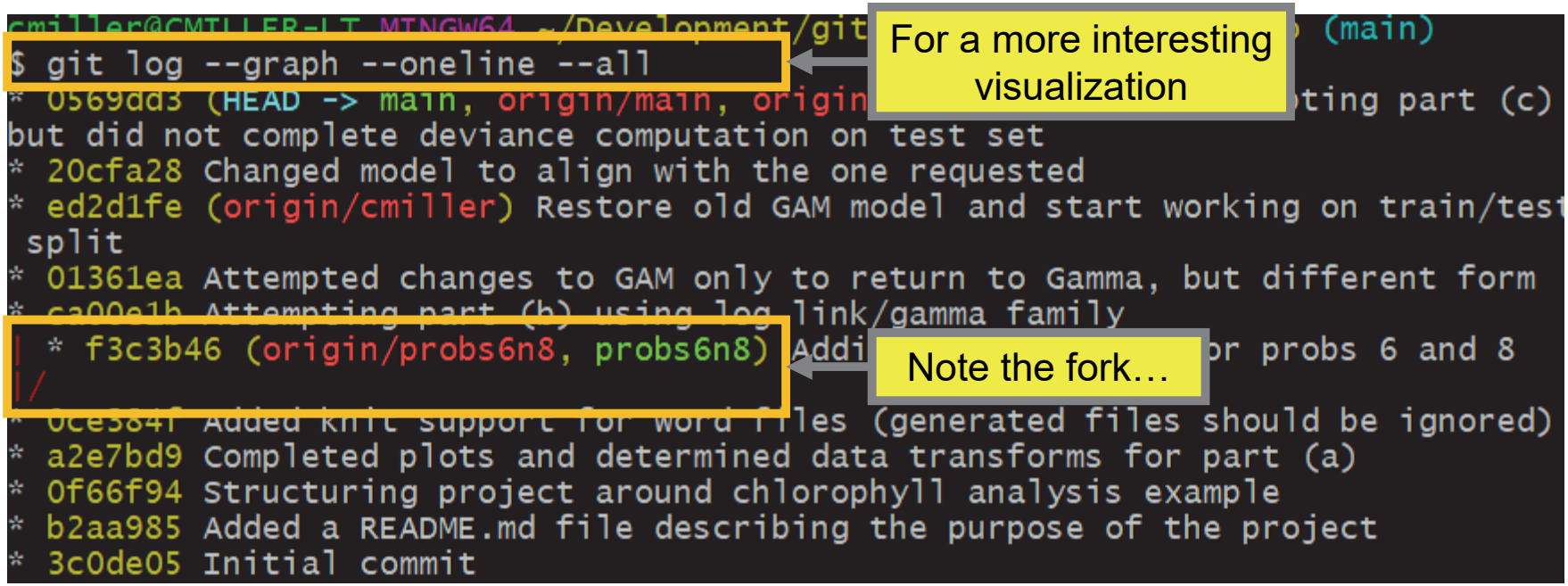
In this case, it was not too hard



A merger is treated like a new commit



Our log shows a more interesting structure now



Merge conflicts happen when Git does not know how to properly merge branches, and needs manual intervention

```
188  Now fit a GAM.
189
190  ```{r}
191  <<<<<<< HEAD
192  gam(chl ~ s(lat) + s(lon), data = chl, family = gaussian)
193  =======
194  chl_model <- chl ~
195                  s(tr.bath, k = 60, bs = 'cr') +
196                  s(jul.day, k = 120, bs = 'cr') +
197                  s(chl.sw, k = 60, bs = 'cr')
198
199  chl_fit <- gam(chl_model,
200                 data = chl_transform,
201                 family = Gamma(link = "log"),
202                 method = "REML"
203                )
204
205  summary(chl_fit)
206  >>>>>>> main
207  ```
208
209  ### Part (c)
210
211  *In this sort of modeling the aim is to improve on simply using
```

- Two users might edit the same line in two branches in different ways
- When those branches are merged, Git will throw an error and insert lines into files with unresolved differences
- After manually handling the conflicts with a text editor, commit the changes, then carry on; Git will view the conflicts as resolved

Let's introduce a potential merge conflict when we merge into our collaborator's branch...

```
165 ### Part (b)
166
167 *Using `mgcv::gam`, try modelling the data using a model of the sort suggested
168 (but with predictors transformed as in part (a)). Make sure that you use an
169 appropriate family. It will probably help to increase the default basis
170 dimensions used for smoothing, somewhat (especially for `jul.day`). Use the
171 `"cr"` basis to avoid excessive computational cost.*
172
173 ```{r}
174 gam(chl ~ s(lat) + s(lon), data = chl, family = gaussian)
175 ```
```

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (main)
$ git checkout probs6n8
Switched to branch 'probs6n8'
Your branch is up to date with 'origin/probs6n8'.

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git commit -m "Attempting initial solution to 9b"
[probs6n8 9391772] Attempting initial solution to 9b
 1 file changed, 1 insertion(+), 1 deletion(-)
```

The merger fails...

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git merge main
Auto-merging vignettes/WoodCh7Exercises.Rmd
CONFLICT (content): Merge conflict in vignettes/WoodCh7Exercises.Rmd
Automatic merge failed; fix conflicts and then commit the result.
```

... and we can see where

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git merge main
Auto-merging v                                                    .Rmd
CONFLICT (cont                                                    .
Automatic merg
```

```
188  Now fit a GAM.
189
190  ```{r}
191  <<<<<<< HEAD
192  gam(chl ~ s(lat) + s(lon), data = chl, family = gaussian)
193  =======
194  chl_model <- chl ~
195                   s(tr.bath, k = 60, bs = 'cr') +
196                   s(jul.day, k = 120, bs = 'cr') +
197                   s(chl.sw, k = 60, bs = 'cr')
198
199  chl_fit <- gam(chl_model,
200                 data = chl_transform,
201                 family = Gamma(link = "log"),
202                 method = "REML"
203                )
204
205  summary(chl_fit)
206  >>>>>>> main
207  ```
208
209  ### Part (c)
210
211  *In this sort of modeling the aim is to improve on simply using
```

... and we can see where

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git merge main
Auto-merging v
CONFLICT (cont                                                    .Rmd
Automatic merg
```

```
188  Now fit a GAM.
189
190  ```{r}
191  <<<<<<< HEAD
192  gam(chl ~ s(lat) + s(lon), data = chl, family = gaussian)
193
194  chl_model <- chl ~
195                   s(tr.bath, k = 60, bs = 'cr') +
196                   s(jul.day, k = 120, bs = 'cr') +
197                   s(chl.sw, k = 60, bs = 'cr')
198
199  chl_fit <- gam(chl_model,
200                 data = chl_transform,
201                 family = Gamma(link = "log"),
202                 method = "REML"
203                )
204
205  summary(chl_fit)
206  >>>>>>> main
207  ```
208
209  ### Part (c)
210
211  *In this sort of modeling the aim is to improve on simply using
```

Lines from this branch

Lines from incoming branch

Surprised that this is the *only* spot where a merge conflict happens? Git recognizes the differences elsewhere as intended. This allows for making changes to one master version of the project and propagating them to other deviations of the project.

... and we can see where

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git merge main
Auto-merging v
CONFLICT (cont                                                    .Rmd
Automatic merg                                                    .
```

```
188  Now fit a GAM.
189
190  ```{r}
191  <<<<<<< HEAD
192  gam(chl ~ s(lat) + s(lon), data = chl, family = gaussian)
193  =======
194  chl_model <- chl ~
195                   s(                  , k = 60,
196                   s(jul.day, k = 120,
197                   s(chl.sw,  k = 60, b
198
199  chl_fit <- gam(chl_model,
200                 data = chl_transform,
201                 family = Gamma(link =
202                 method = "REML"
203                )
204
205  summary(chl_fit)
206  >>>>>>> main
207  ```
208
209  ### Part (c)
210
211  *In this sort of modeling the aim is to improve on simply using
```

Indicates where a conflict happened and in what branch what text appears (HEAD refers to the checked-out commit of the currently checked-out branch).

To fix, edit the file to resolve the conflict...

```
188  Now fit a GAM.
189
190  ```{r}
191  chl_model <- chl ~
192                   s(tr.bath, k = 60, bs = 'cr') +
193                   s(jul.day, k = 120, bs = 'cr') +
194                   s(chl.sw, k = 60, bs = 'cr')
195
196  chl_fit <- gam(chl_model,
197                 data = chl_transform,
198                 family = Gamma(link = "log"),
199                 method = "REML"
200                )
201
202  summary(chl_fit)
203  ```
204
205  ### Part (c)
206
207  #In this sort of modeling the aim is to improve on simply
```

In this case we deleted lines from one commit, but we could have done anything we wanted to resolve the conflict, even adding brand new lines. Git doesn't care what you do to resolve the conflict.

... Then add and commit your changes (just like any other commit)

```
188  Now fit a GAM.
189
190  ```{r}
191  chl_model <- chl ~
```

```
cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8|MERGING)
$ git add --all

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8|MERGING)
$ git commit -m "Conflict resolved by using code from main (better solution)"
[probs6n8 cf0ee86] Conflict resolved by using code from main (better solution)

cmiller@CMILLER-LT MINGW64 ~/Development/git/DATAWorks22-Git-Intro (probs6n8)
$ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 1.33 KiB | 273.00 KiB/s, done.
Total 8 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 2 local objects.
To https://github.com/ntguardian/DATAWorks22-Git-Intro.git
   f3c3b46..cf0ee86  probs6n8 -> probs6n8
```

Basic branch management commands

Command	Description	Examples
`git branch`	Branch management command	`git branch -d somebranch  # Delete branch` `git branch -m old new  # Rename branch`
`git checkout`	Retrieve data from commits; discussed more later	`git checkout identifier`
`git merge`	Merge one branch into another branch	`git merge branchname`

Conclusion

Git can be an important part of making research reproducible

- As a research project carries on, the data and analysis techniques used are prone to change over time
- Results ultimately depend on the version of data and analysis used at any given time
- With Git, the evolution of the data and the analysis methods is tracked over time
- This allows for someone to see how the results of the project depend on the version of data and analysis used, and track down the specific version of these that made a particular analysis and what changed to produce a more current analysis

This is only the start of learning Git

- Git is powerful and frankly intimidating software
- It is not the most simple version control system out there, but it's popular for good reasons
- I often default to using Git for almost any project, regardless of whether I plan to share or not, or how big the project might be
- The more you use Git, the more familiar it will become
- Remember: `--help`, Google, and StackExchange are your friends

Commands I would love to talk about if I had time

Command	Description
`git log`	See commit history; setting options can allow for making useful displays
`git diff`	See what files changed between commits and in what ways
`git bisect`	Find the first "bad" commit (say where a bug was introduced)
`git stash`	"Stash" changes, then reset to the most recent commit
`git grep`	Search an entire repo using regular expressions

Cheat Sheet for Git

Command	Description	Examples
`git config`	Configure Git	`git config --global user.name "Jane Doe"` `git config --global user.email jdoe@ida.org`
`git init`	Create a new repo	`git init`
`git clone`	Clone an existing repository	`git clone "https:..."`
`git add`	Add changes to (or brand new) files to a staged commit	`git add myfile.txt` `git add --all`
`git commit`	Commit changes	`git commit -m "Message"`
`git pull`	For repos hosted remotely (such as GitHub), update repo with changes from host	`git pull`
`git push`	For repos hosted remotely (such as GitHub), send changes to host	`git push`
`git rm`	Remove a file from tracking	`git rm myfile.txt`
`git reset`	Git's "undo" command	`git reset --hard  # CAUTION: Deletes changes`
`git branch`	Branch management command	`git branch -d somebranch  # Delete branch` `git branch -m old new  # Rename branch`
`git checkout`	Retrieve data from commits	`git checkout identifier`
`git merge`	Merge one branch into another branch	`git merge branchname`

BACKUP

Navigating your computer using Bash

Command	Description	Examples

`ls`	List information about files or directories	`ls # List files`
		`ls -lh # List files with description`
		`ls -lha # List all files with description`

`cd`	Change the current working directory	`cd "My Documents" # Go to My Documents folder`
		`cd .. # Go to parent directory`
		`cd ~/ # Go to home directory`
		`cd - # Go to last visited`

`pwd`	Print the current working directory	`pwd`
`head/tail`	Show the first few lines/last few lines of a text file (resp.)	`head "My file.txt"` `tail my_file.txt`
`cat`	Can be used to show file contents (but has different purpose)	`cat "My file.txt"`
`mv`	Move a file; also used to rename files	`mv file.txt dir/file.txt` `mv oldname.txt newname.txt`
`cp`	Copy a file	`cp original.txt new.txt`
`mkdir`	Create a new directory	`mkdir my_new_repo`
`rm`	Remove a file	`rm filename.txt` `rm -r my_new_repo/* # Delete subcontents`
`rmdir`	Remove a directory	`rmdir my_new_repo`
`exit`	Exit the shell	`exit`

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
04-2022	Final	START DATE	END DATE Apr 2022

4. TITLE AND SUBTITLE
DATAWorks 2022 - Introduction to git

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
HQ0034-19-D-0001		
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
BD-09-2299	229990	

6. AUTHOR(S)
Miller, Curtis, G.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Institute for Defense Analyses 730 East Glebe Road Alexandria, Virginia 22305	NS D-33021 H 2022-000108

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER
Director, Operational Test and Evaluation 1700 Defense Pentagon Washington, DC 20301	DOT&E	

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Version control software manages, archives, and (optionally) different versions of files. Git is the most popular program for version control and serves as the backbone for websites such as Github, Bitbucket, and others. In this mini-tutorial we will introduce basics of version control in general, Git in particular. We explain what role Git plays in a reproducible research context. The goal of the course is to get participants started using Git. We will create and clone repositories, add and track files in a repository, and manage git branches. We also discuss a few Git best practices.

This tutorial was first given at DATAWorks Conference 2022 in Alexandria, Virginia.

15. SUBJECT TERMS
Git; Reproducible Research; reproducible analyses; Data Management; tutorial

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	SAR	143

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER
John T. Haman	703-845-2132