git

AN INTRODUCTION

JOHN RELLIS

WEB DEVELOPMENT 2

What is git?

Imagine you're working on a big group project for your class. Everyone needs to contribute their part, but it's hard to keep track of who did what and when.

Git is like an ultra-organized, digital notebook that helps you and your classmates work together on your project without stepping on each other's toes.

What is git?

Git helps you keep track of every change you make to your project.

It's like taking snapshots of your work at different points in time.

So if something goes wrong, you can go back to a previous snapshot and fix it.

This makes git a “version control” system.

What is git?

Git allows multiple people to work on the same project at the same time.

It helps you merge everyone's changes together seamlessly.

It's like having a magic wand that combines all your classmates' contributions into one neat document.

What is git?

Git also acts as a backup for your project.

If your computer crashes or you accidentally delete something important, you can always get your work back from Git, especially if you have pushed it to a remote repository

What is git?

Think of Git as your trusty assistant that keeps your project organized, helps you work with others, and ensures you never lose your hard work.

So, to recap.

git is a version control system that enables collaboration and backup of (typically) code based projects.

What is a version control system?

Imagine you're writing a story on your computer.

Every time you make a change, like adding a new paragraph or deleting a sentence, you hit the "save" button.

But what if you want to go back to how your story looked a few days ago?

That's where version control comes in.

What is a version control system?

Version control is like a smart save button for your files.

It keeps track of every change you make, along with who changed it and when, creating a history of your work.

So, if you ever want to go back to a previous version of your story, you can!

What is a version control system?

But version control isn't just about saving different versions of your work.

It also helps you collaborate with others.

Let's say you're writing your story with a friend.

Version control allows both of you to work on the same document at the same time, without stepping on each other's toes.

It keeps track of who made which changes and helps you merge your work together.

Repository (or repo)

A repository is where git stores all the files and their complete history.

It's like a folder on your computer that git is watching closely.

When you initialize a repository in a folder (using git init), git starts tracking all the changes made to the files in that folder.

```
→  myproject ls -l
total 8
-rw-r--r--@ 1 john  staff  13  9 May 15:59 README.md
→  myproject cat README.md
# My Project
→  myproject
```

```
→  myproject git init
Initialized empty Git repository in /Users/john/Desktop/myproject/.git/
→  myproject git:(main) ✗ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
→  myproject git:(main) ✗
```

Add README.md to the staging area

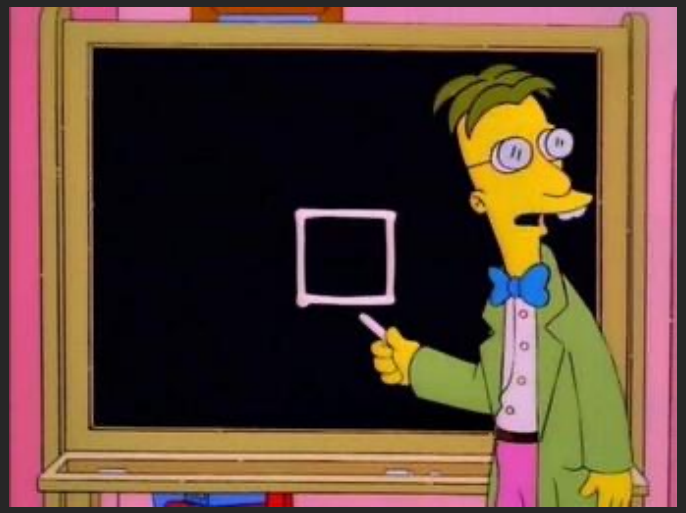
```
→  myproject git:(main) ✗ git add README.md
→  myproject git:(main) ✗ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

→  myproject git:(main) ✗
```

Commit our file to the repo

```
➜  myproject git:(main) ✗ git commit -m "Adding README.md"
[main (root-commit) 2cab2c2] Adding README.md
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

```
➜  myproject git:(main) git status
On branch main
nothing to commit, working tree clean
➜  myproject git:(main)
```

Staging area? Commit?

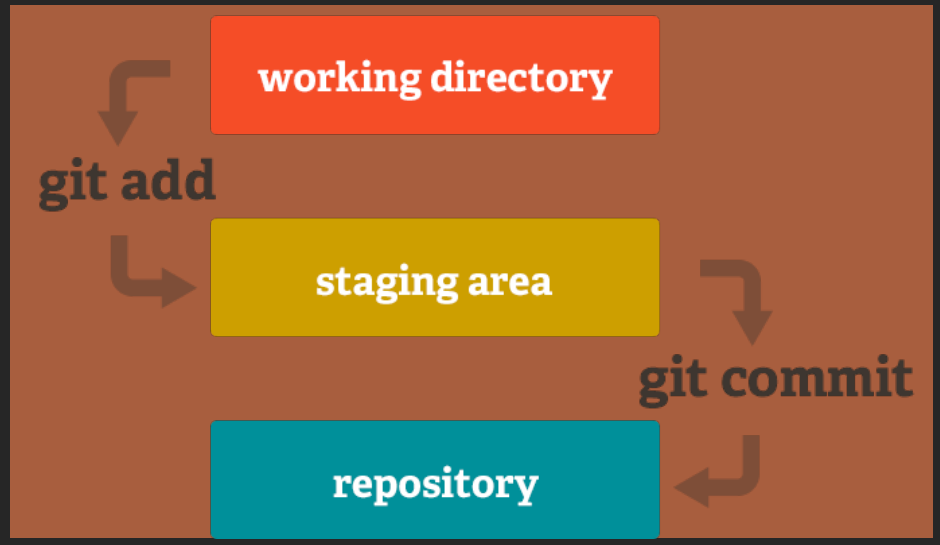


Staging area?

git has something called the "staging area" or "index".

This is an intermediate area where commits can be formatted and reviewed before completing the commit.

One thing that sets Git apart from other tools is that it's possible to quickly stage some of your files and commit them without committing all of the other modified files in your working directory or having to list them on the command line during the commit.



Staging area?

git also makes it easy to ignore this feature if you don't want that kind of control — just add a '-a' to your commit command in order to add all changes to all files to the staging area.

https://git-scm.com/about/staging-area

Commit?

We commit changes to our repository, these changes can include new files, updated files, deleted files, renamed files

Once a change (or a number of changes) are committed as part of a commit, these changes become part of the history of the repository, we can view the changes using `git log` and then `git show 2cab2c21f166cb01066f929d78eabd8755cf5794`

```
commit 2cab2c21f166cb01066f929d78eabd8755cf5794 (HEAD -> main)
Author: John Rellis <john[redacted]com>
Date:   Thu May 9 16:05:18 2024 +0100

    Adding README.md
(END)
```

```
Date:   Thu May 9 16:05:18 2024 +0100

    Adding README.md

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..a2beefd
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# My Project
(END)
```

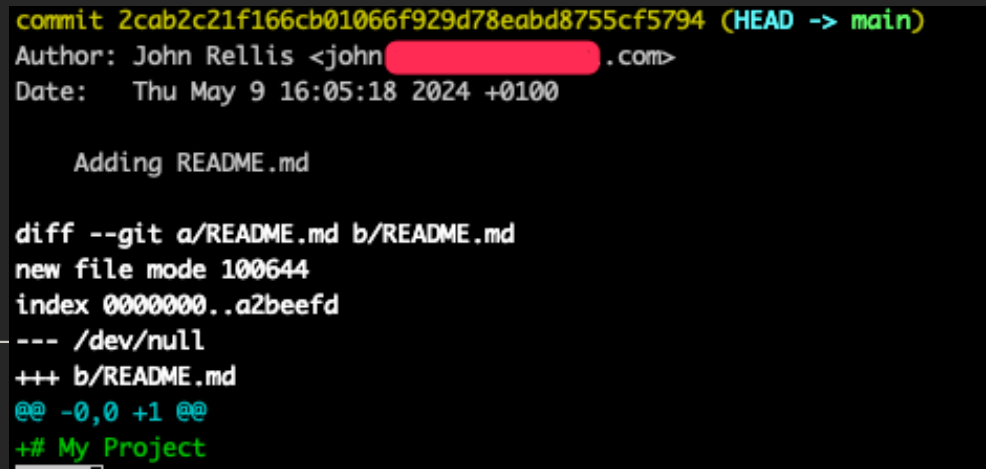
Commit?

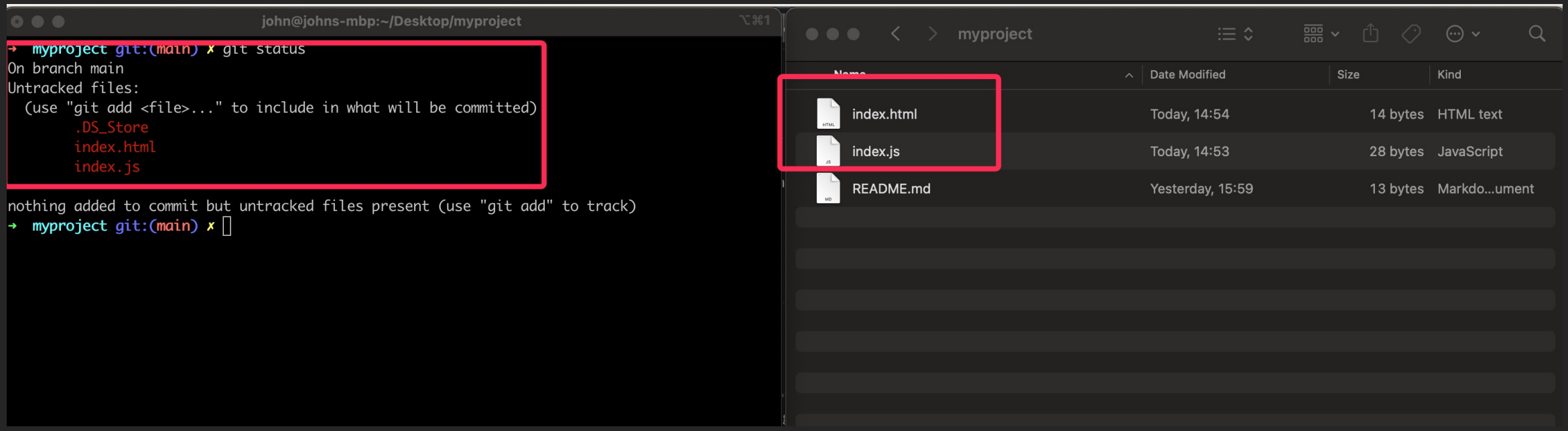
Therefore, a commit is a record of one or more changes committed to the repository

A commit is ID'ed via a hash within git itself, 2cab2c21f166cb01066f929d78eabd8755cf5794, in the last example.

These hashes can be shortened to the first 7 characters, the following are equivalent:

- git show 2cab2c21f166cb01066f929d78eabd8755cf5794
- git show 2cab2c2

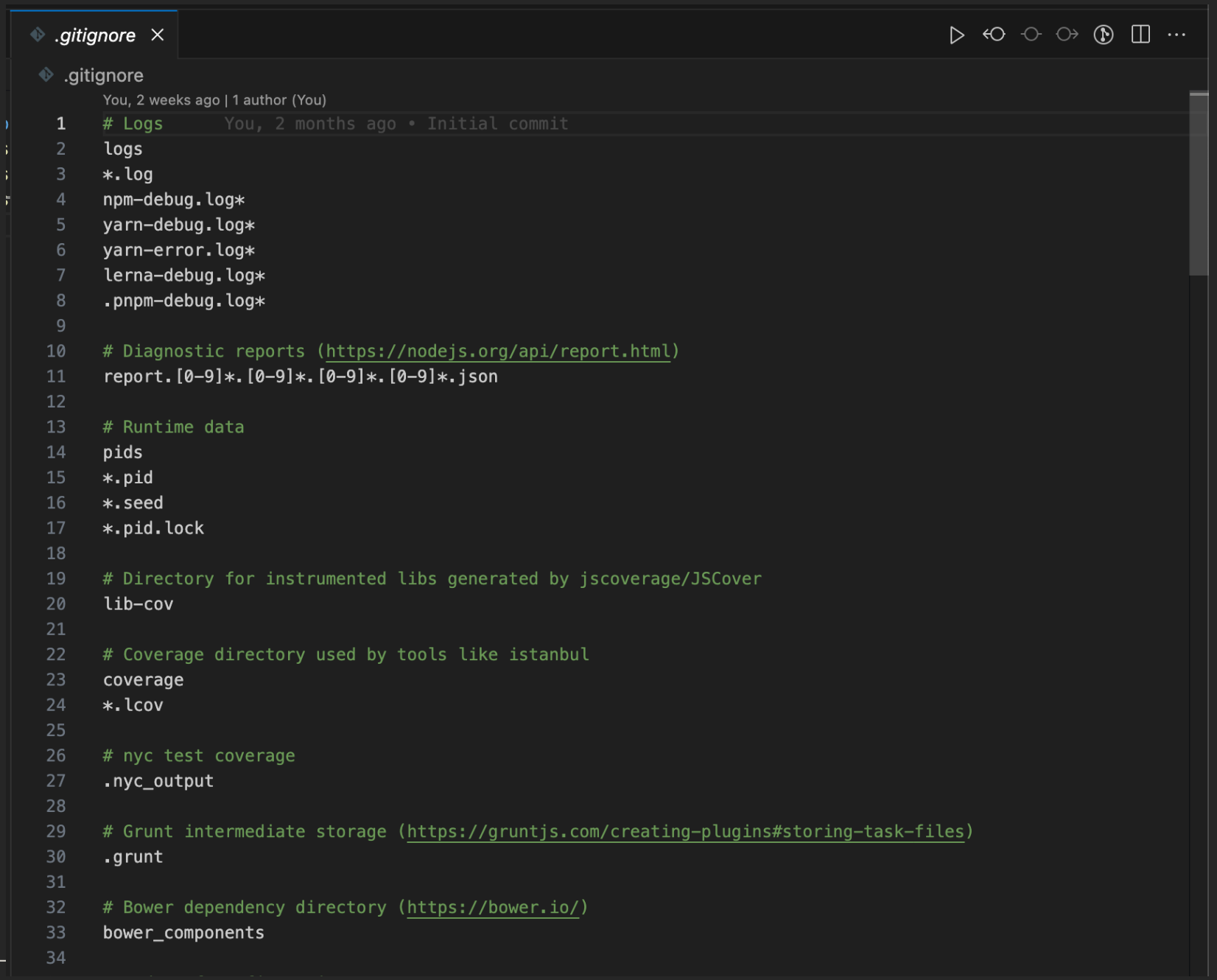




2 new files.....
Oh wait, what's .DS_Store?
It's a directory OSX uses to manage it's file system.
We do not want this in our repository

.gitignore

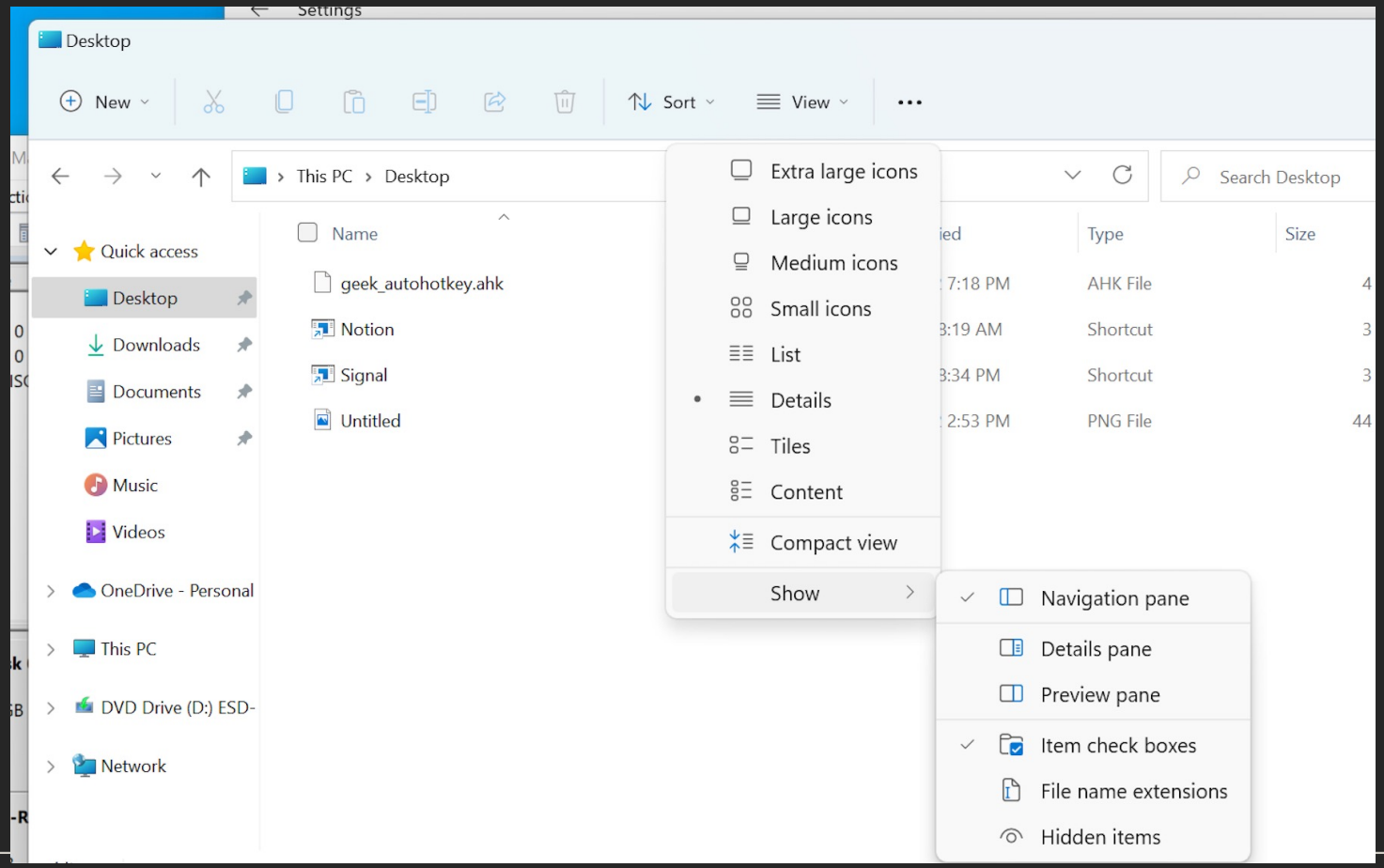
.gitignore (or git ignore) is a file that contains a list of files or directories to not track in our repository



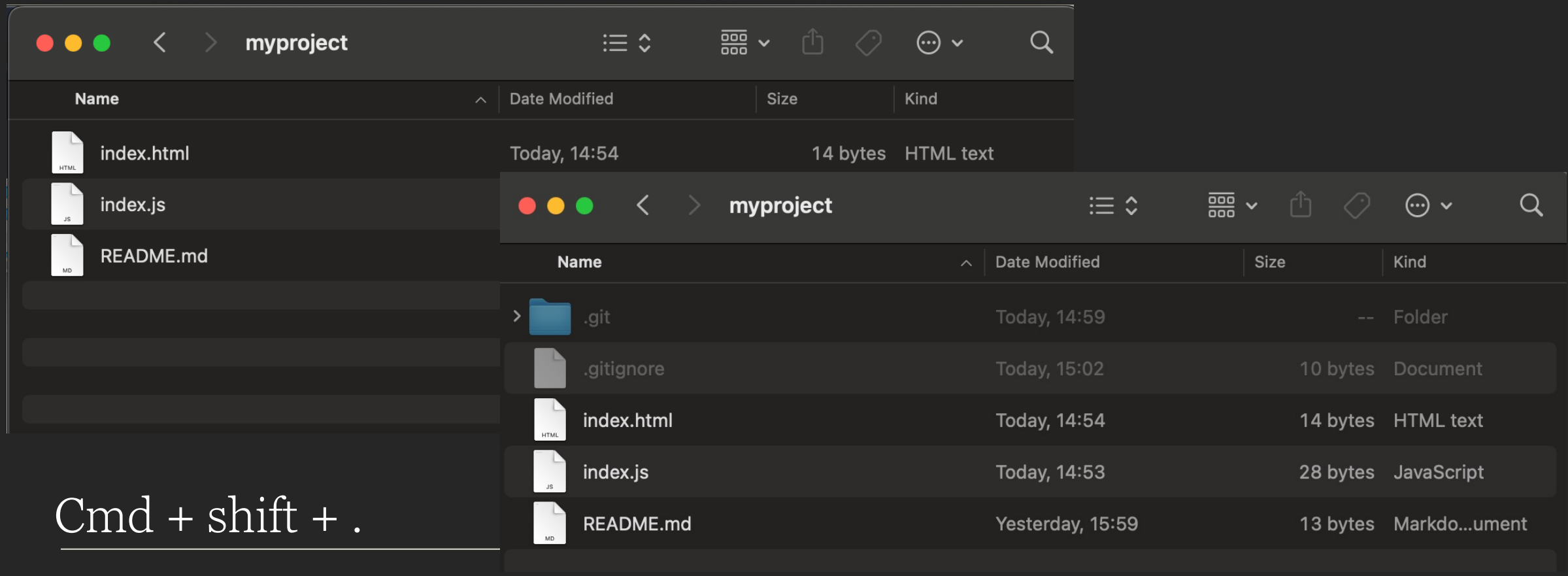
.gitignore

Note that it is a “hidden file” in most operating systems, that is, it is named with a`.` at the beginning

.gitignore


Settings
Desktop
New
Sort
View
This PC › Desktop
Search Desktop
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos
OneDrive - Personal
This PC
DVD Drive (D:) ESD-
Network
Name
geek_autohotkey.ahk
Notion
Signal
Untitled
Type
Size
7:18 PM
AHK File
4
3:19 AM
Shortcut
3
3:34 PM
Shortcut
3
2:53 PM
PNG File
44
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Tiles
Content
Compact view
Show
Navigation pane
Details pane
Preview pane
Item check boxes
File name extensions
Hidden items

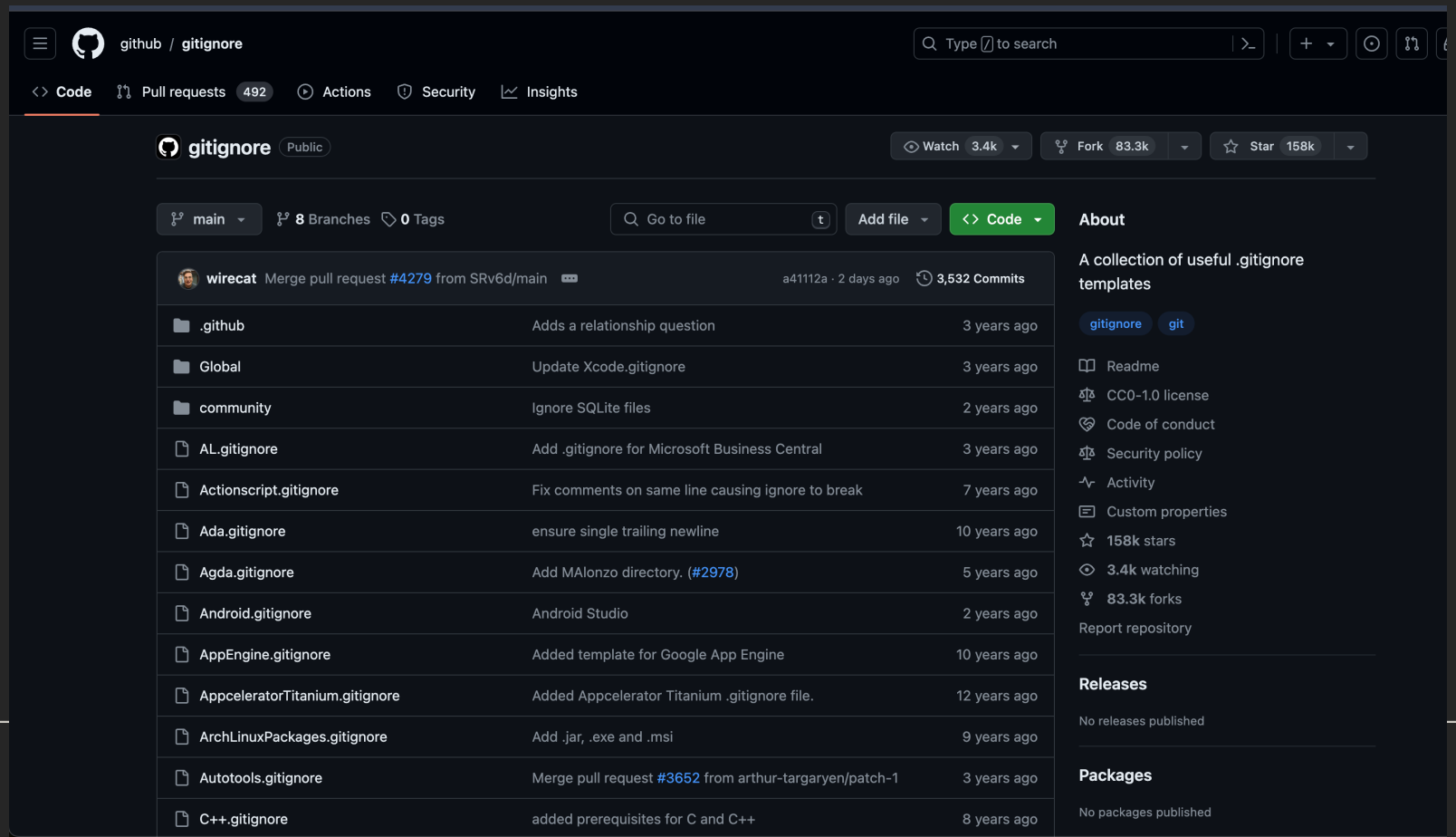
.gitignore



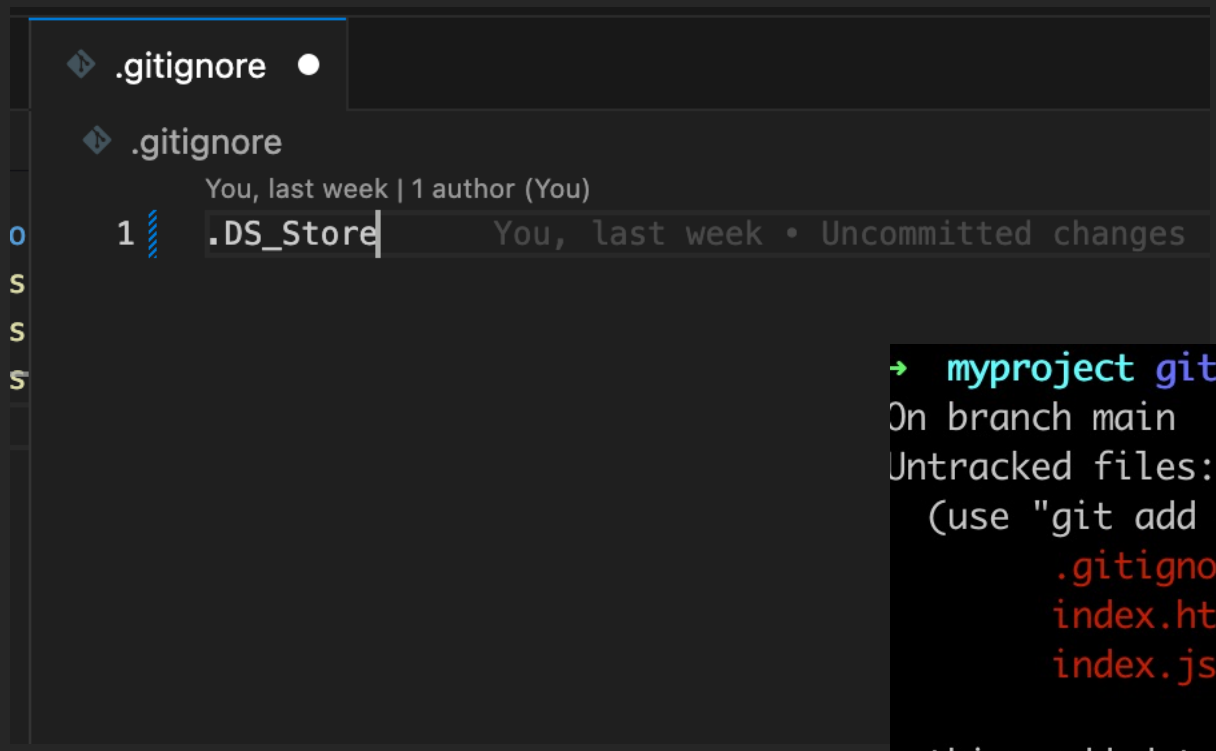
Cmd + shift + .

.gitignore

Useful .gitignores: https://github.com/github/gitignore



So, we add a .gitignore




```
→  myproject git:(main) ✗ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        index.html
        index.js

nothing added to commit but untracked files present (use "git add" to track)
```

Now we add and commit our new files

```
→  myproject git:(main) ✗ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        index.html
        index.js

nothing added to commit but untracked files present (use "git add" to track)
→  myproject git:(main) ✗ git add -A
→  myproject git:(main) ✗ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   index.html
        new file:   index.js

→  myproject git:(main) ✗ git commit -m "adding too many files"
[main 6f1d825] adding too many files
 3 files changed, 3 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 index.html
 create mode 100644 index.js
→  myproject git:(main) git status
On branch main
nothing to commit, working tree clean
→  myproject git:(main)
```

git log

```
commit 6f1d8251567fab9cda0662a0e58a073601e24ead (HEAD -> main)
Author: John Rellis <johr                    >
Date:   Fri May 10 15:27:18 2024 +0100

    adding too many files

commit 2cab2c21f166cb01066f929d78eabd8755cf5794
Author: John Rellis <john                .com>
Date:   Thu May 9 16:05:18 2024 +0100

    Adding README.md
(END)
```

git show 6f1d825

```
commit 6f1d8251567fab9cda0662a0e58a073601e24ead (HEAD -> main)
Author: John Rellis <johr                com>
Date:   Fri May 10 15:27:18 2024 +0100

    adding too many files

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..e43b0f9
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+.DS_Store
diff --git a/index.html b/index.html
new file mode 100644
index 0000000..18ecdcb
--- /dev/null
+++ b/index.html
@@ -0,0 +1 @@
+<html></html>
diff --git a/index.js b/index.js
new file mode 100644
index 0000000..6be0237
--- /dev/null
+++ b/index.js
@@ -0,0 +1 @@
+console.log('hello world');
(END)
```

.git folder

This is the history of your repository, you probably shouldn't be in here if you are new to git but it's good to know it exists

git init creates this folder

Use git log to see the commit history

Use git status to see the current status of the repository

```
→  .git git:(main) tree .
.
├── COMMIT_EDITMSG
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── fsmonitor-watchman.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-merge-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   ├── prepare-commit-msg.sample
│   ├── push-to-checkout.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── main
├── objects
│   ├── 08
│   │   └── 8876f26a914c0308dfff977a1189da5e218904
│   ├── 18
│   │   └── ecdcb795c33d6ab7bbb43f647947defca5634d
│   ├── 2c
│   │   └── ab2c21f166cb01066f929d78eabd8755cf5794
│   ├── 6b
│   │   └── e02374db118b9fb99cd98c6b403e5a558d0d57
│   ├── 6f
│   │   └── 1d8251567fab9cda0662a0e58a073601e24ead
│   ├── a2
│   │   └── beefd59223ea16000788d77e62f96bdaf23c7c
│   ├── e4
│   │   └── 3b0f988953ae3a84b00331d0ccf5f7d51cb3cf
│   ├── fa
│   │   └── 5e6af79e30fc26ab4acbd96388fde22b4c2f36
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── main
```

Recap

- We used git init to turn our directory into a repository
- We learned about the staging area that we can add files to to stage a commit
- We can then commit these files to the repository
- git log will show us the log of commits
- Each commit has a hash that can be shortened to the first 7 characters
- git show <commit hash> will show us information about the commit
- We used .gitignore to ignore files/folders within our project directory
- So far, this is all done locally on your machine