



Think Like Git

Eli Sander
Lead Data Scientist, Tempus Labs
PyData Global 2021



Times Git has ruined my day

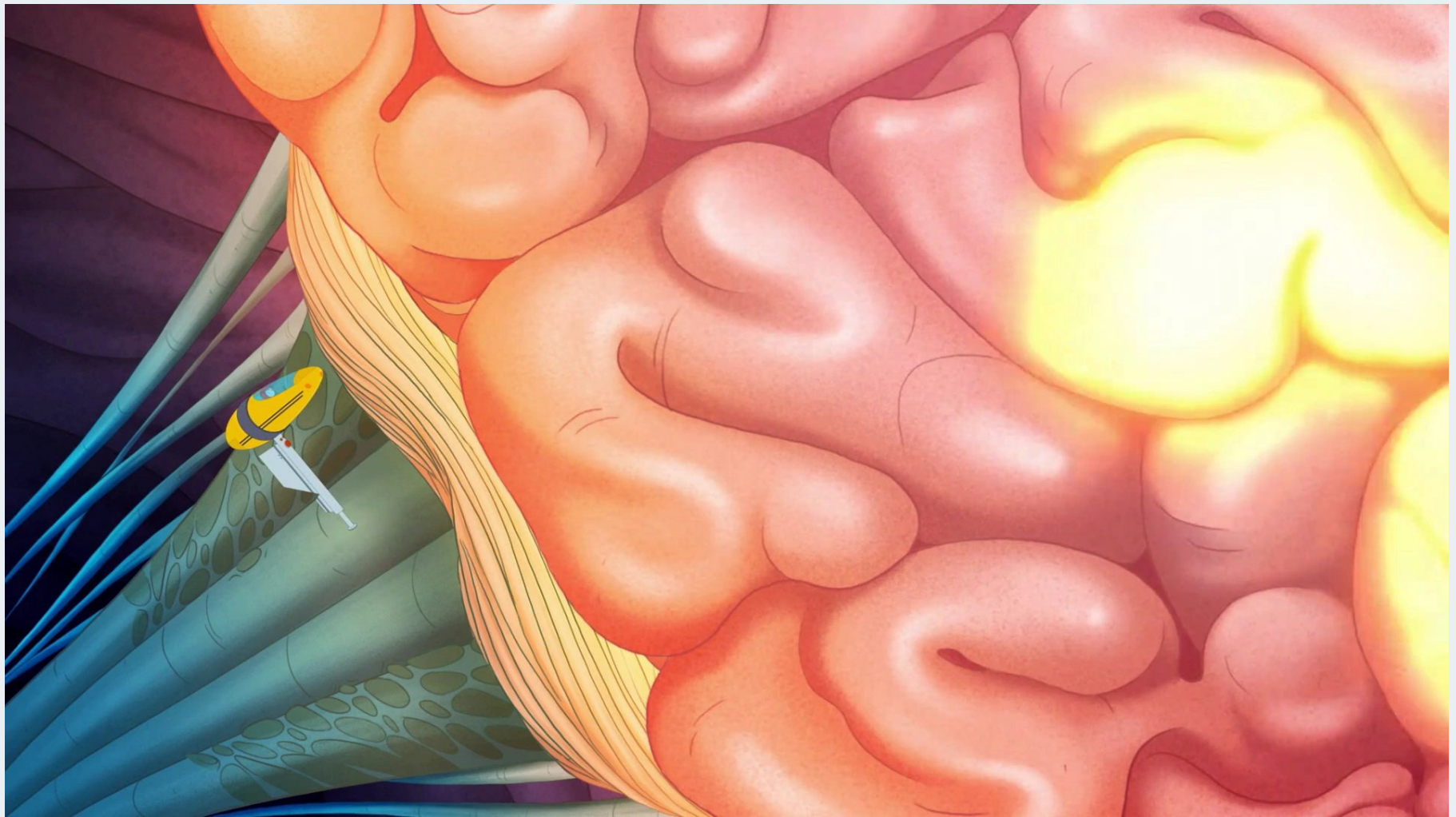
- The time I spent hours on a failed rebase
- The time I pushed so much data to Bitbucket, they locked my account
- The many times I just deleted the whole thing and started over



It doesn't have to be this way.

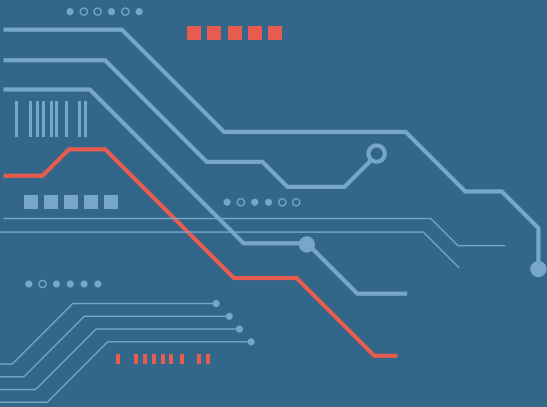


If you understand git, you can debug it!



Agenda

1. Design our own version control system
2. Learn how Git works
3. Solve all of our Git problems



01

| ||||| |

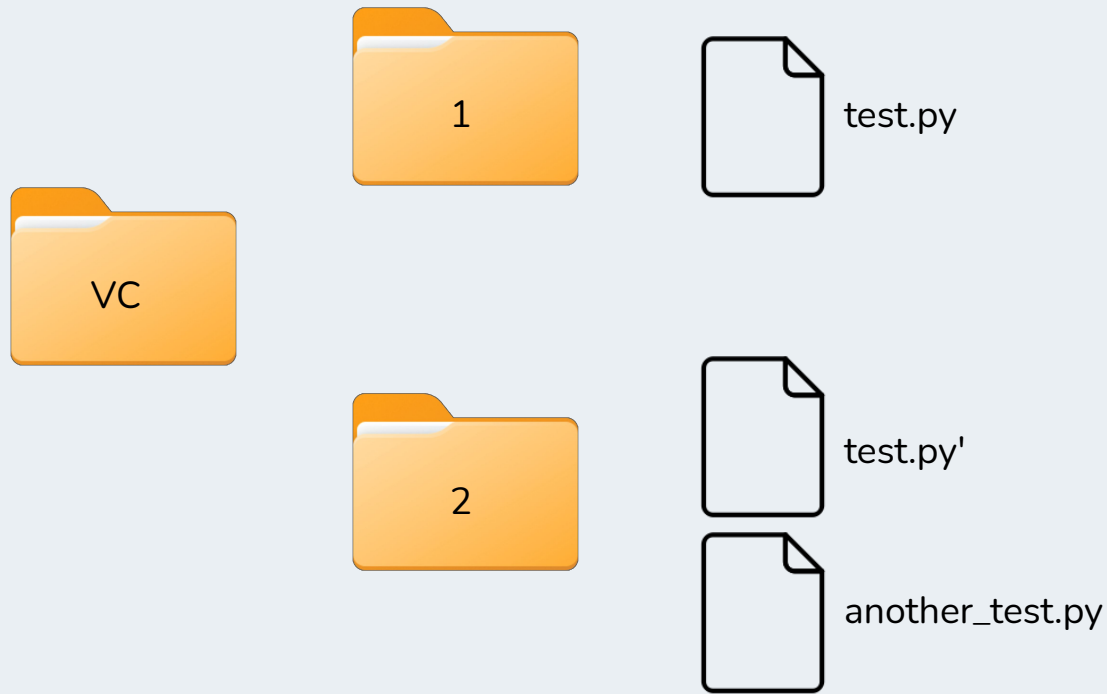
Version Control From Scratch

Problem 1: I need code checkpoints

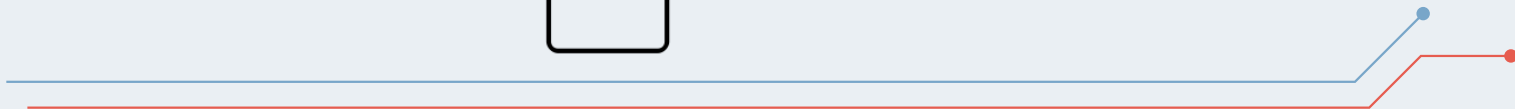
- Save a new copy of the code every time
 - Stash it in a "Version Control" folder that the user doesn't see
- How do we name the checkpoints?
 - Let's number them. It's like a built-in time record



Commits & Checkout

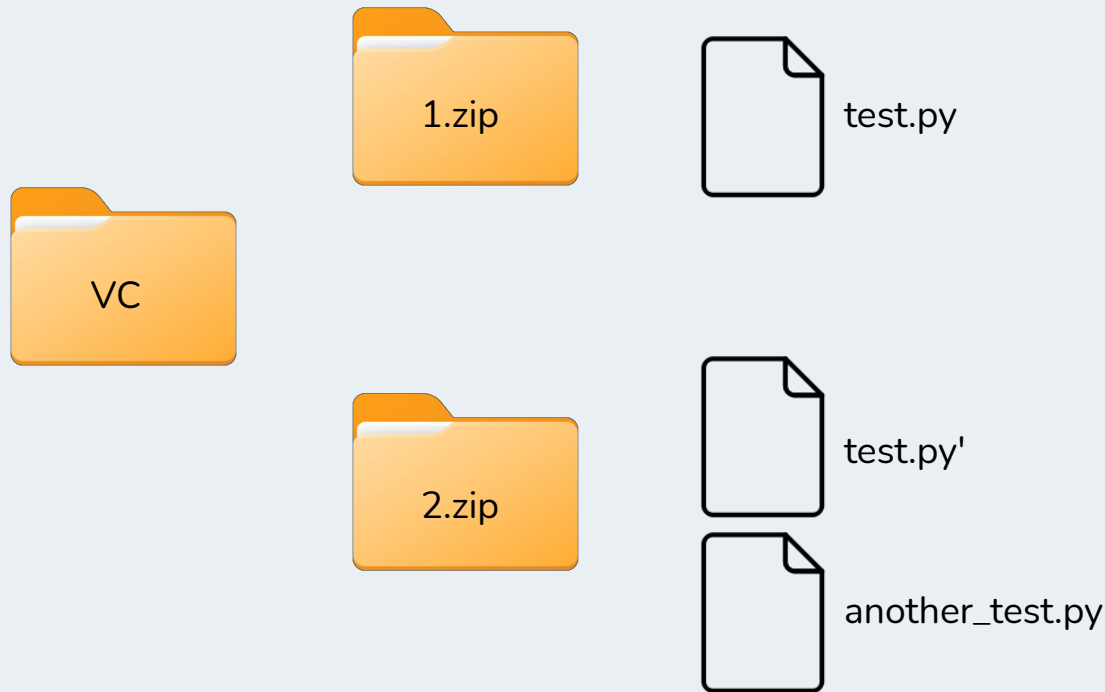


```
vc commit  
vc checkout 1
```



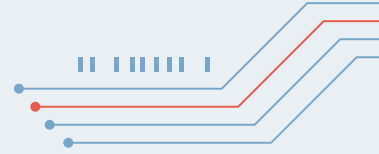
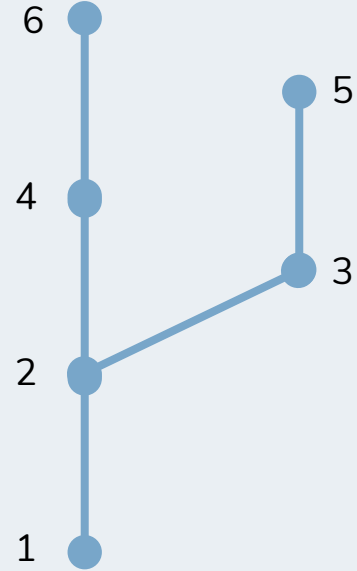
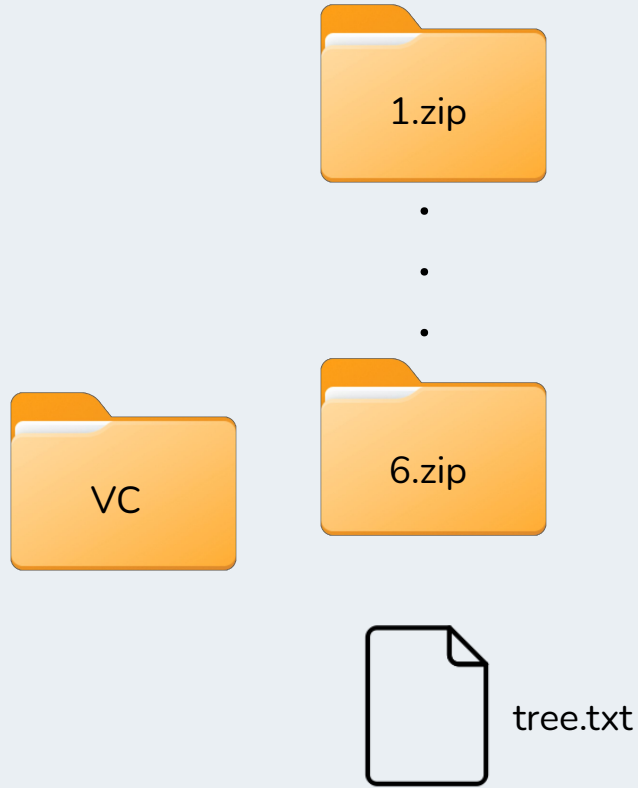


Problem 1.5: I'm out of disk space



Problem 2: I want to experiment

- Running multiple "experiments" at once means multiple "branches"
- Need some kind of tree structure
- Store this in its own file



tree.txt

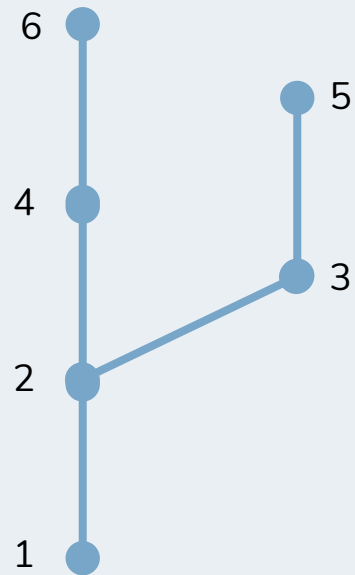
1>>2

2>>3

2>>4

3>>5

4>>6

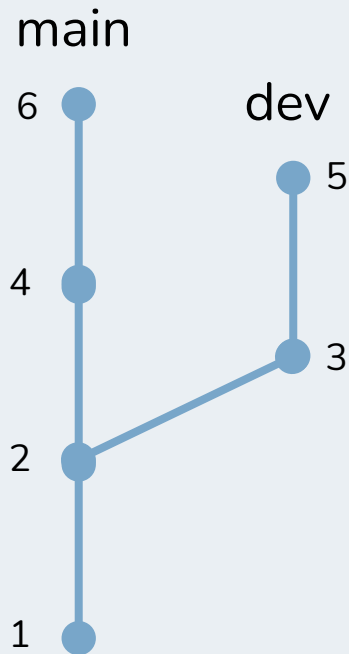


Problem 2.5: Staying organized

tree.txt

main: 1>>2>>4>>6

dev: 2>>3>>5



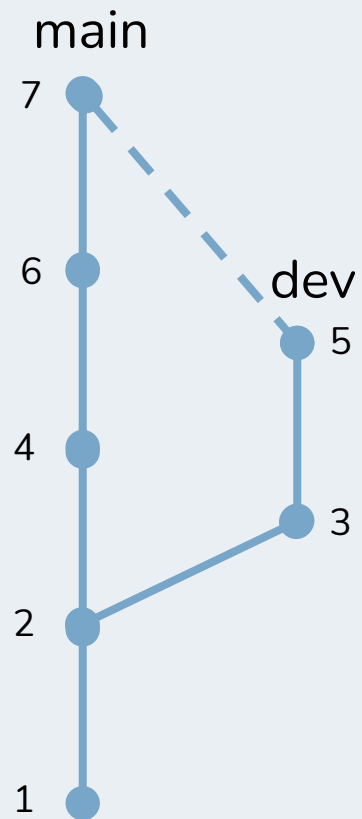
Problem 3: I want to combine code from different branches

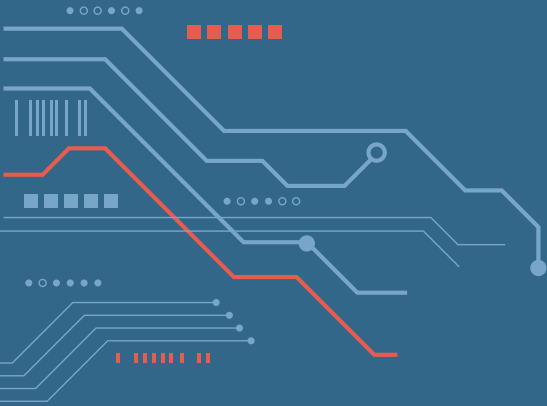
- How do we "merge"?
 - A merge is just another commit
 - Identify and resolve conflicts
 - Compare files via SHA1 hash
 - If the hashes match, so do the files
 - User needs to resolve files with same name but different hashes
- Merges are directional. Merging A into B creates a different commit history than B into A.

tree.txt

main: 1>>2>>4>>6>>7

dev: 2>>3>>5



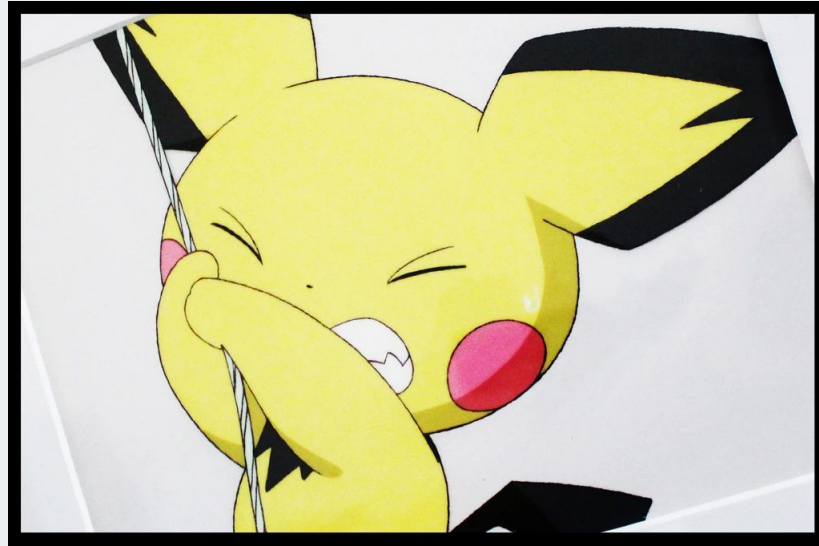


02

| | | | | | | |

Inside Git

Confession time



celbase.com

Git does what we did. It stores every file every time you commit.

Every committed version of every file is stored in the .git folder.

Every file.

In its entirety.

Problem 1: checkpointing

- Compress & hash every file in the repo
- Store the folder structure in a bunch of tree objects
 - Compress & hash the trees
- Point to the tree and parent in the commit, along with the user's info and message
 - Compress & hash the commit

Where does it all live?

- All objects (files, commits, trees) live in .git/objects
- Every object is named after its hash, so that no objects are duplicated
- Hash collisions are basically impossible.

What's in a commit?

```
tree d9f6ad74a2ab622b3a6f7b7b782ddd7e09a2291b
parent fde3b0478b64a5888a985389a6aecfa3245956ec
author Eli Sander <eli.sander@tempus.com> 1634068706 -0500
committer Eli Sander <eli.sander@tempus.com> 1634068706 -0500

test for pydata talk
```


What's in a tree?

```
100644 blob e6ff5952c107c17142adef4933ef2628f3ac33c1 .gitignore
100644 blob 761ce787301392904b97d57927121c94dbfb1039 CNAME
100644 blob 40e6030143cb33efbc9acc8b42411ba8b098d015 LICENSE
100644 blob b7b13d20c251c33a017e337d476dcb8d19f39e08 Research.md
100644 blob 7dbceac509f04d70ca6bab2579290ff383480527 _config.yml
040000 tree 007f736ccdab321d23503dbe2d9f20720d180907 _data
040000 tree fb8ff46fc3acb7ab6c784b7e1df3b2d62fbe7a43 _includes
040000 tree eae4d585905b8ed6983477b4ea949897b3d6823e _layouts
040000 tree 90f194398cd0e769eca02aaf82fbc38be6ee554f _posts
040000 tree 0ba27246b8b823ebf3176fece69533985732d54b _sass
100644 blob 2c7fc34171407ed88064d812ce934a0529db2078 about.html
100644 blob 52c33dff04cfd128fe29605d6518363c659bcefb about.md
100644 blob 3fb1c327d3ea2664d5384fdd451b45e1e9c609c3 archive.html
040000 tree a34020f377de9b46c3924d25eac60d16c3355224 blog
```

Problem 2: experimentation

- Let users create named branches
- Track the "head" of each branch
- There is no "canonical" branch, just naming conventions

```
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io/.git/refs/heads (test-dev)$ ls
master          test-dev
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io/.git/refs/heads (test-dev)$ cat master
788c055179e148dc95e1036b6ce03476260edd9f
```



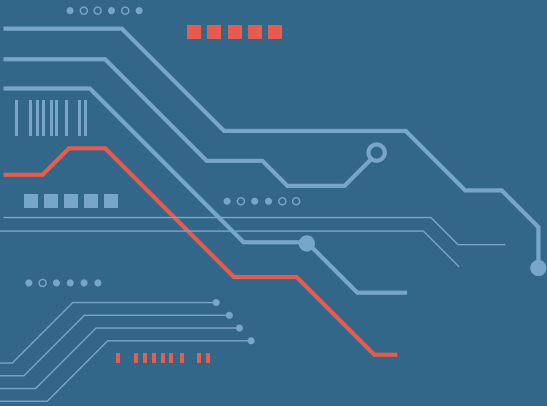
Commit hash for the latest
commit to master

Problem 3: merges

- Compare everything in the two trees
 - Accept any new files from either tree
 - For files with the same name, accept if hashes match
 - Otherwise, dive into the file to resolve

Problem 3: merges

- Compare everything in the two files
 - Find most recent common ancestor of the file
 - Accept any unique changes from either file
 - Otherwise, raise a "merge conflict" for the user to solve



03

| | | | | | | |

Git in Practice



The "working copy"

- Git ignores anything you haven't staged or committed
- If you checkout a different commit, it won't touch local changes
 - Uncommitted changes *follow you from branch to branch*
- If you checkout a commit that conflicts with your working copy, Git won't even try to resolve

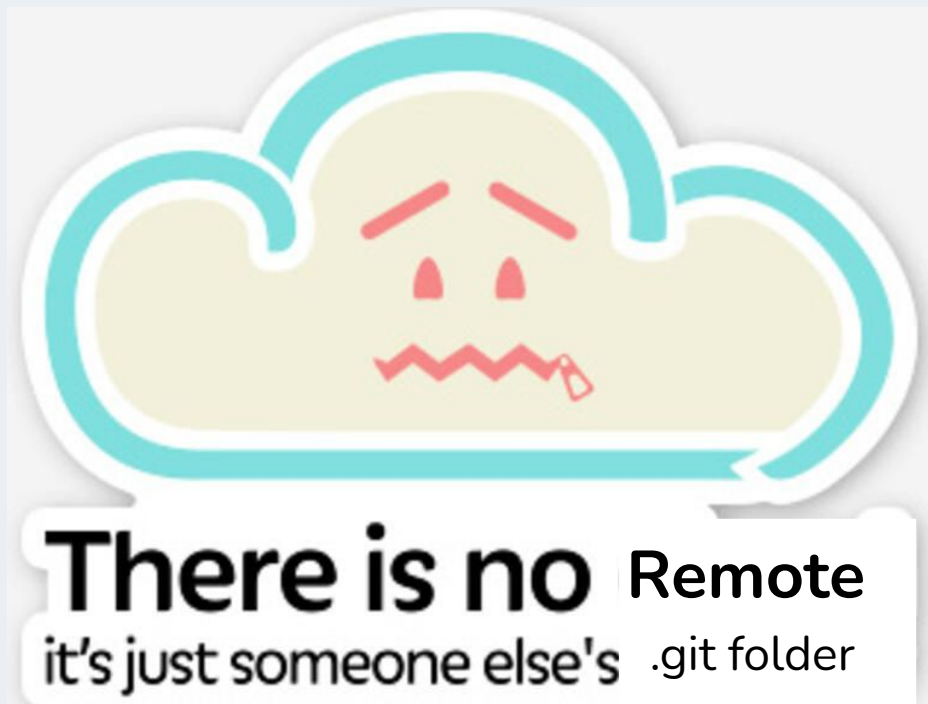
Take-Homes

- Always run `git status` before you commit/checkout
- Commit branch-specific changes before switching branches

Fetch & remotes



Fetch & remotes



Fetch & remotes

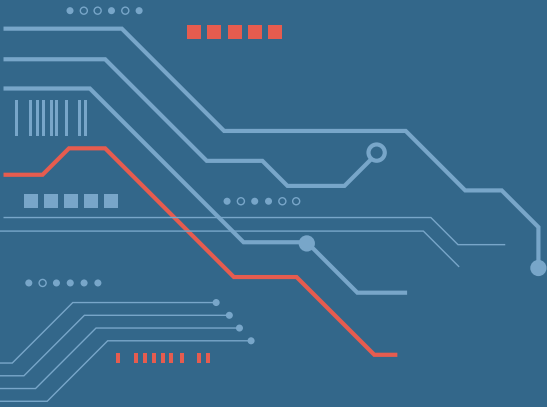
- `git fetch` is functionally identical to your coworker emailing you their `.git` folder
- Setting a remote is just more convenient than email!
- Remote names like `upstream` make it seem fancier than it is

GitHub

GitHub is just any other working copy. At its core is a (really good!) UI on top of the git CLI.

Some added conveniences:

- Lets you set a "default" branch
- Editing from GitHub lets you treat it like a real working copy
- Better visibility at the user level
- Concepts of org level, permissioning, etc.



04

| | | | | | | |

Final Thoughts

Final Thoughts

- Git is a complicated CLI on top of a simple tool
- When you hit an error, look at it from git's perspective
 - What is in .git?
 - What is in your working copy?
 - What is in the remote?
- A couple hours trying to build something like git will teach you **a lot** about how/why it works

Resources

- [Git Is Simpler Than You Think](#)
- [Git Internals](#) (a bit in the weeds, but useful reference)
- [The Recurse Center](#)

THANKS

Any questions?

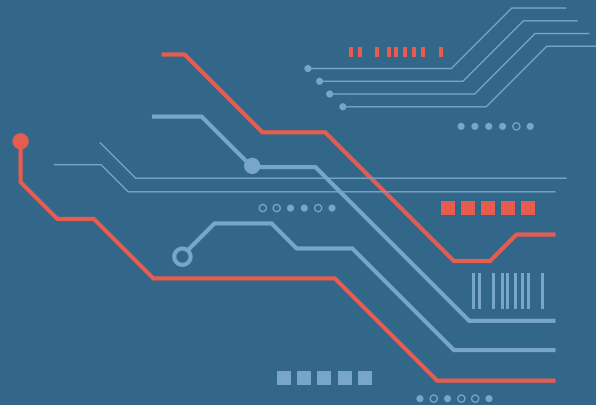


eli@elisander.com

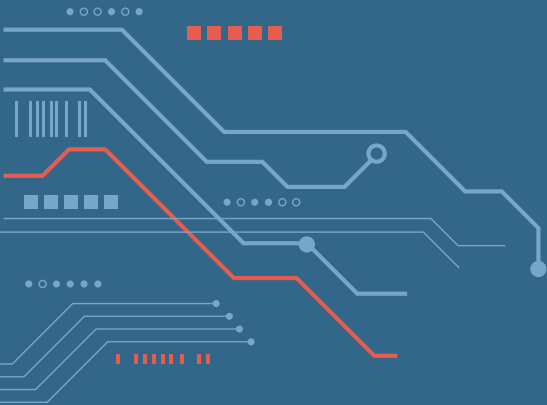


www.tempus.com

www.elisander.com



CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.



05

| | | | | | | |

Appendix

Making an example commit

```
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io (master)$ echo "test" >> test.txt
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io (master)$ git add test.txt
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io (master)$ git cm "test for pydata talk"
[master 788c055] test for pydata talk
1 file changed, 1 insertion(+)
create mode 100644 test.txt
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io (master)$ cd .git/objects/
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io/.git/objects (master)$ ls
78      9d      d9      info    pack
```


Find the commit hash

```
(base) \(^o^)/ elizabeth.sander:~/repos/elsander.github.io/.git/objects (master)$ git log
commit 788c055179e148dc95e1036b6ce03476260edd9f (HEAD -> master)
Author: Eli Sander <eli.sander@tempus.com>
Date:   Tue Oct 12 14:58:26 2021 -0500

    test for pydata talk
```

Extract the commit object

```
(base) \(\^o^\)/ elizabeth.sander:~/repos/elsander.github.io/.git/objects (master)$ git cat-file -p 788c055179e148dc95e1036b6ce03476260edd9f
tree d9f6ad74a2ab622b3a6f7b7b782ddd7e09a2291b
parent fde3b0478b64a5888a985389a6aecfa3245956ec
author Eli Sander <eli.sander@tempus.com> 1634068706 -0500
committer Eli Sander <eli.sander@tempus.com> 1634068706 -0500

test for pydata talk
```

Extract the "tree" object

```
(base) \(\^o\^)/ elizabeth.sander:~/repos/elsander.github.io/.git/objects (master)$ git cat-file -p d9f6ad74a2ab622b3a6f7b7b782ddd7e09a2291b
```

```
100644 blob e6ff5952c107c17142adef4933ef2628f3ac33c1 .gitignore
100644 blob 761ce787301392904b97d57927121c94dbfb1039 CNAME
100644 blob 40e6030143cb33efbc9acc8b42411ba8b098d015 LICENSE
100644 blob b7b13d20c251c33a017e337d476dcb8d19f39e08 Research.md
100644 blob 7dbceac509f04d70ca6bab2579290ff383480527 _config.yml
040000 tree 007f736ccdab321d23503dbe2d9f20720d180907 _data
040000 tree fb8ff46fc3acb7ab6c784b7e1df3b2d62fbe7a43 _includes
040000 tree eae4d585905b8ed6983477b4ea949897b3d6823e _layouts
040000 tree 90f194398cd0e769eca02aaf82fbc38be6ee554f _posts
040000 tree 0ba27246b8b823ebf3176fece69533985732d54b _sass
100644 blob 2c7fc34171407ed88064d812ce934a0529db2078 about.html
100644 blob 52c33dff04cfd128fe29605d6518363c659bcefb about.md
100644 blob 3fb1c327d3ea2664d5384fdd451b45e1e9c609c3 archive.html
040000 tree a34020f377de9b46c3924d25eac60d16c3355224 blog
100644 blob 41db46b927155336d15fa8d3f1a7001447abe050 category.html
040000 tree 99f3a745de62851b092be825d624bd85327885f4 css
100644 blob a6628bd842af95a7f423155dd95510941d3a78dc feed.xml
040000 tree 14032aab85b43a058cfc7025dd4fa9dd325ea97 fonts
```

Extract test.txt

```
(base) \(^o^)/ elizabeth.sander:~/repos/elsander.github.io/.git/objects (master)$ git cat-file -p 9daeafb9864cf43055ae93beb0afd6c7d144bfa4  
test
```

Summary of how Git stores data

- Commits point to the parent commit, and a "tree" containing all files/folders in the repo at time of commit
- All objects (files, commits, trees) are stored in `.git/objects` in your repo
- All objects are compressed, hashed, and named after their hashes
- The most recent commit hash for each branch is listed in `.git/refs/heads`

That's it!