git

9.20.24

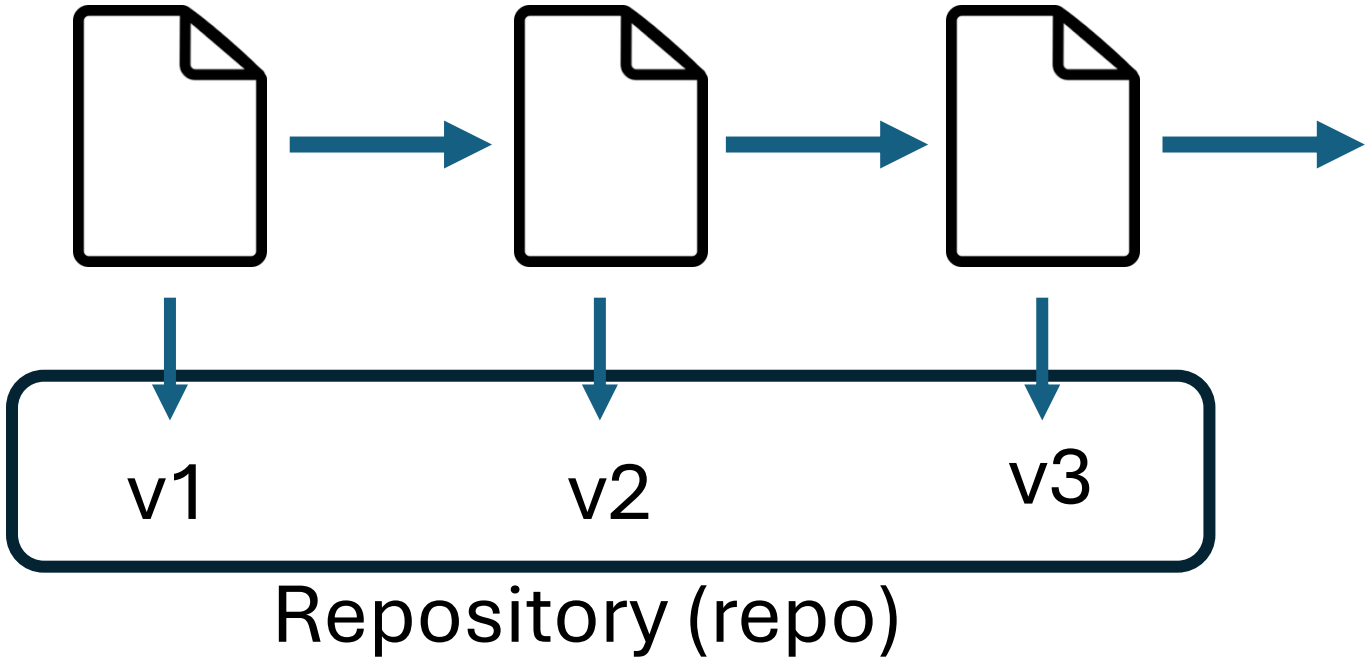
A system for managing code

“Version Control System”

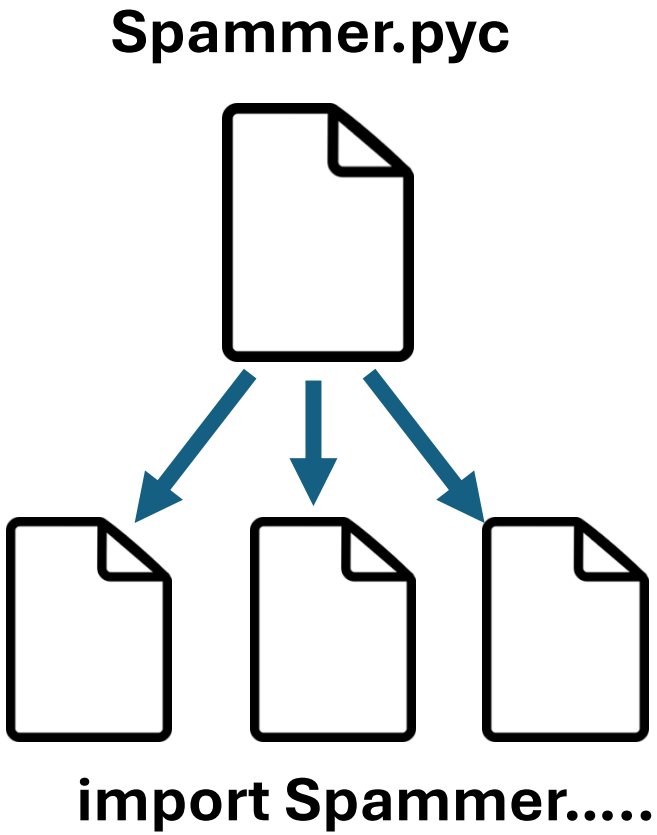
Website for storing code using git

Version Control System (VCS)

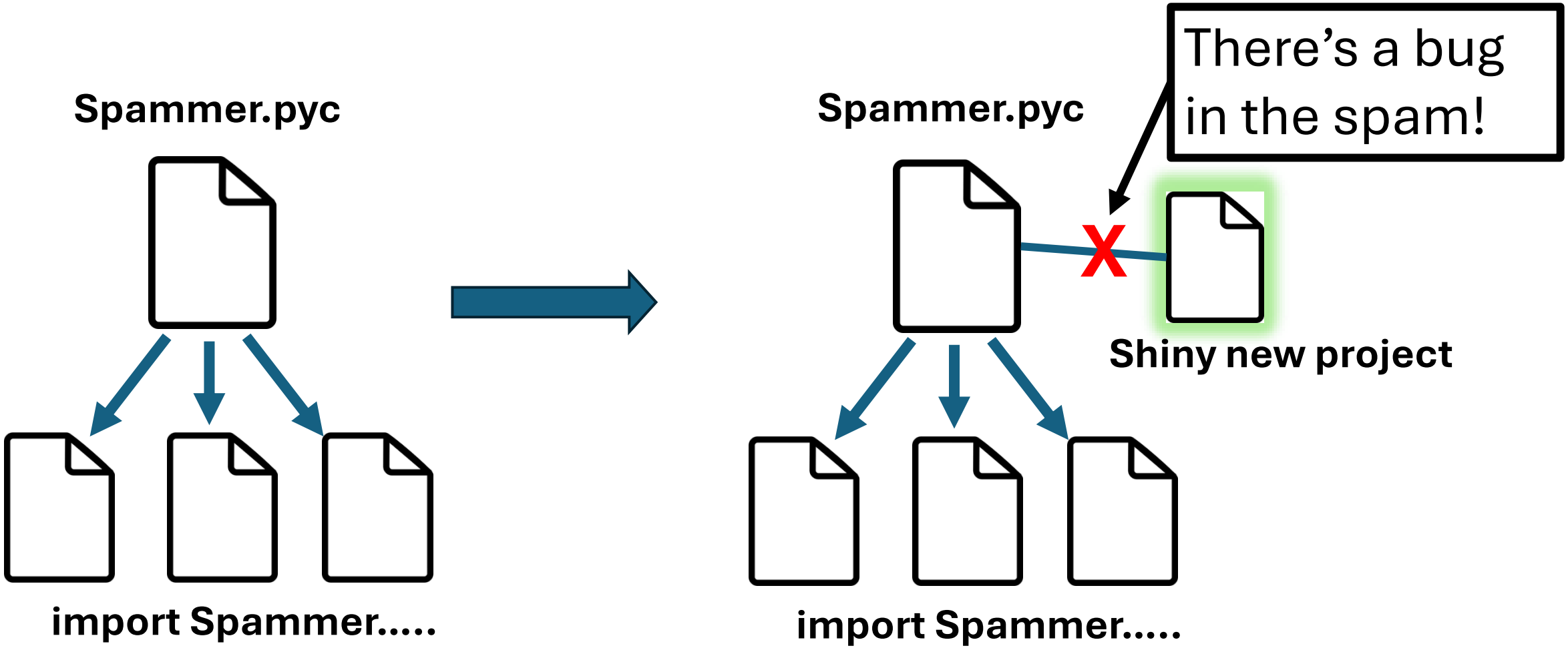
- Store each iteration of a file
- Let you revisit old versions
- Merge code from different people



Why Version-Control?

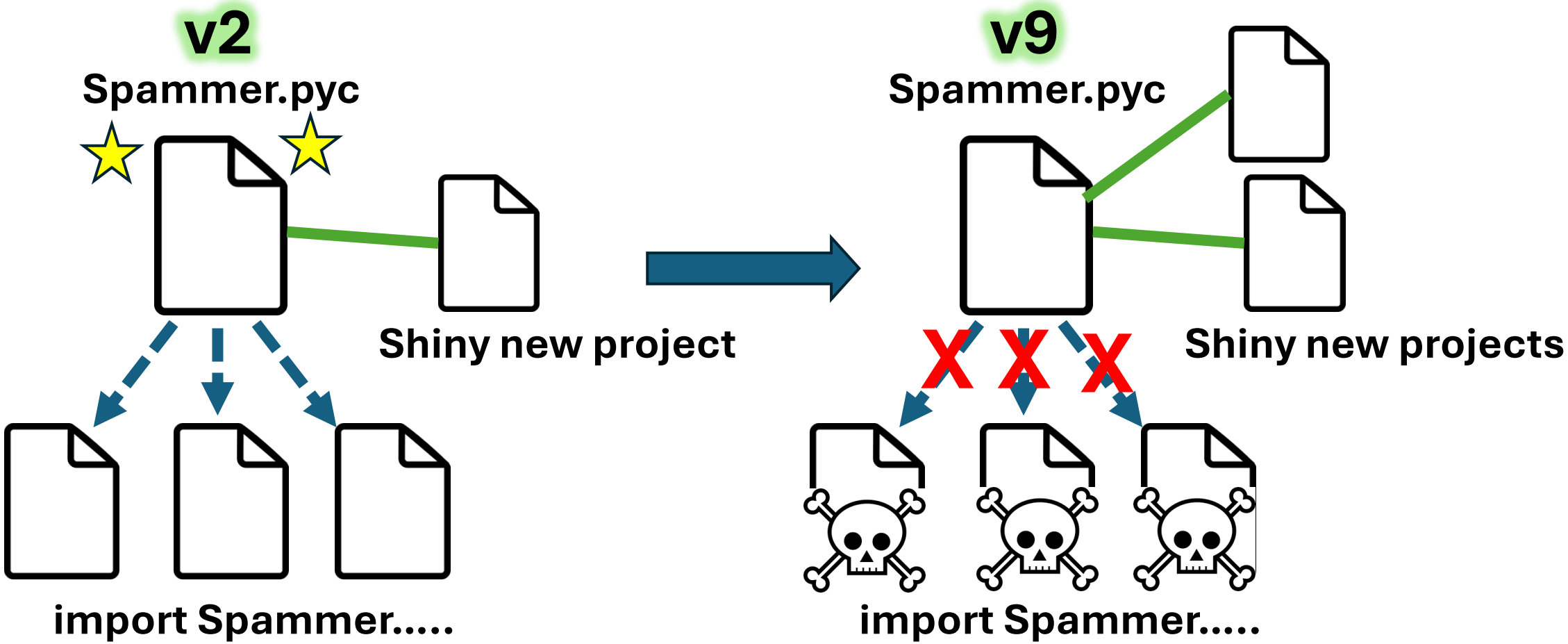


Why Version-Control?

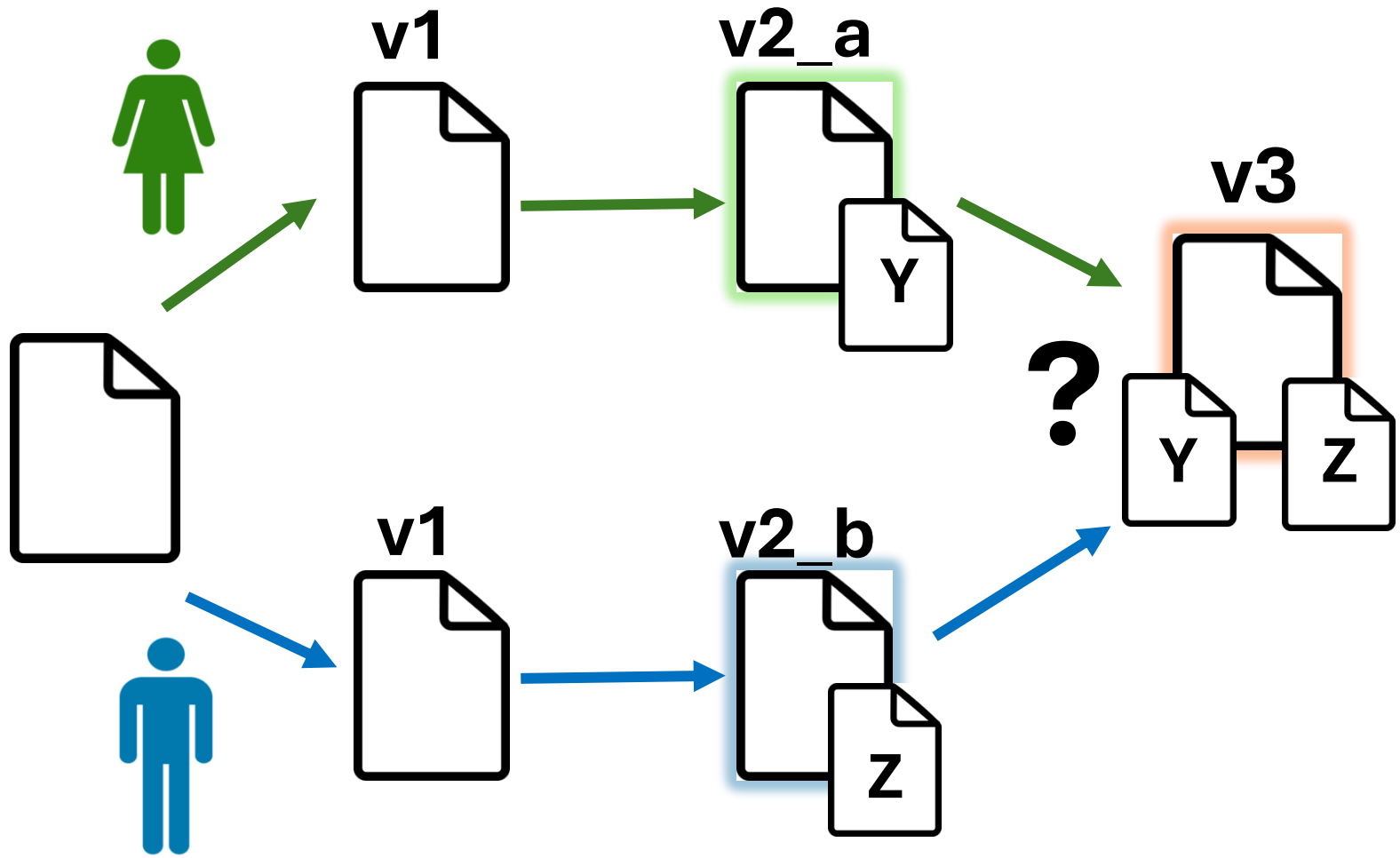


Why Version-Control?

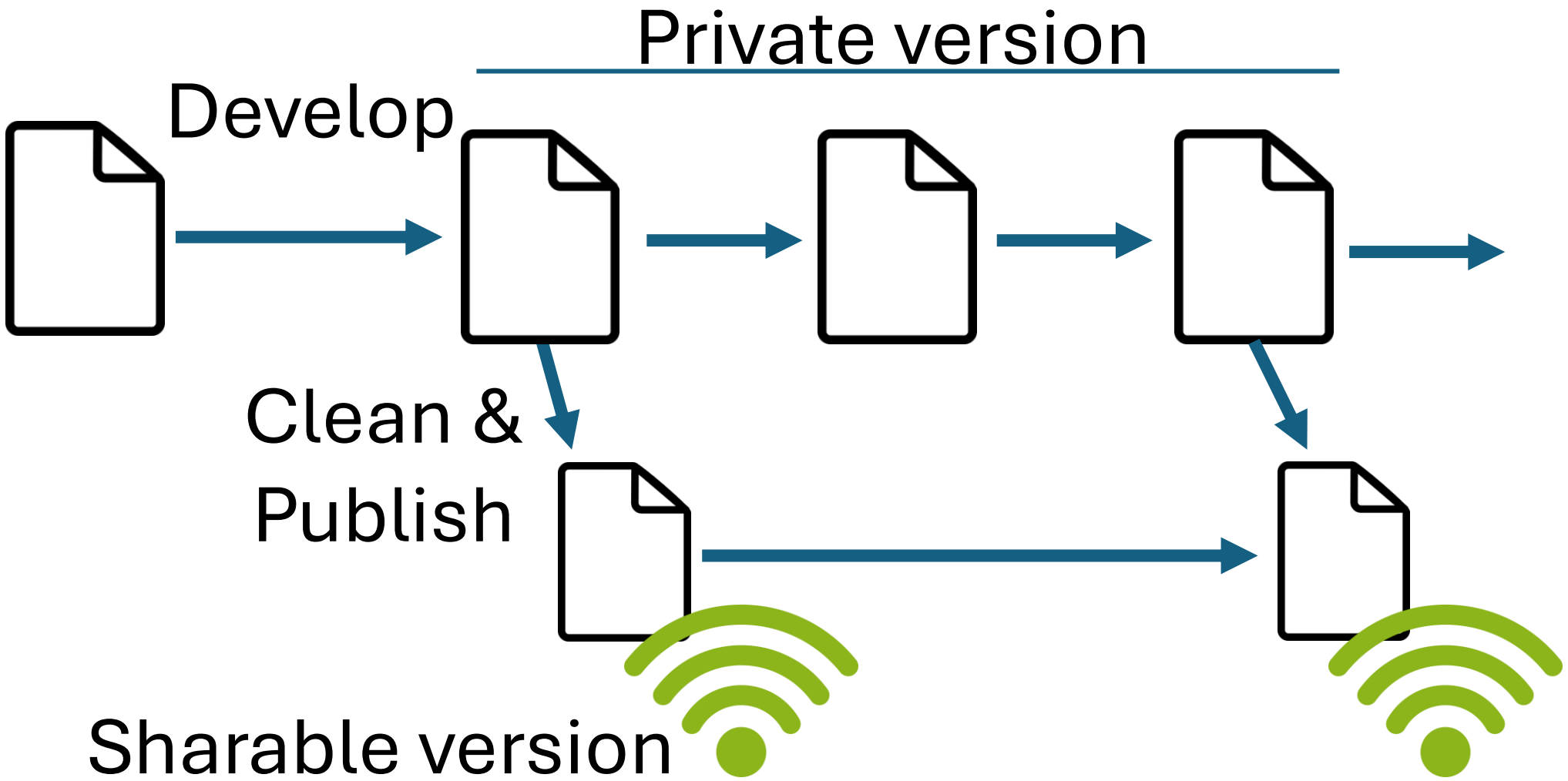
What did I change (why, when)?



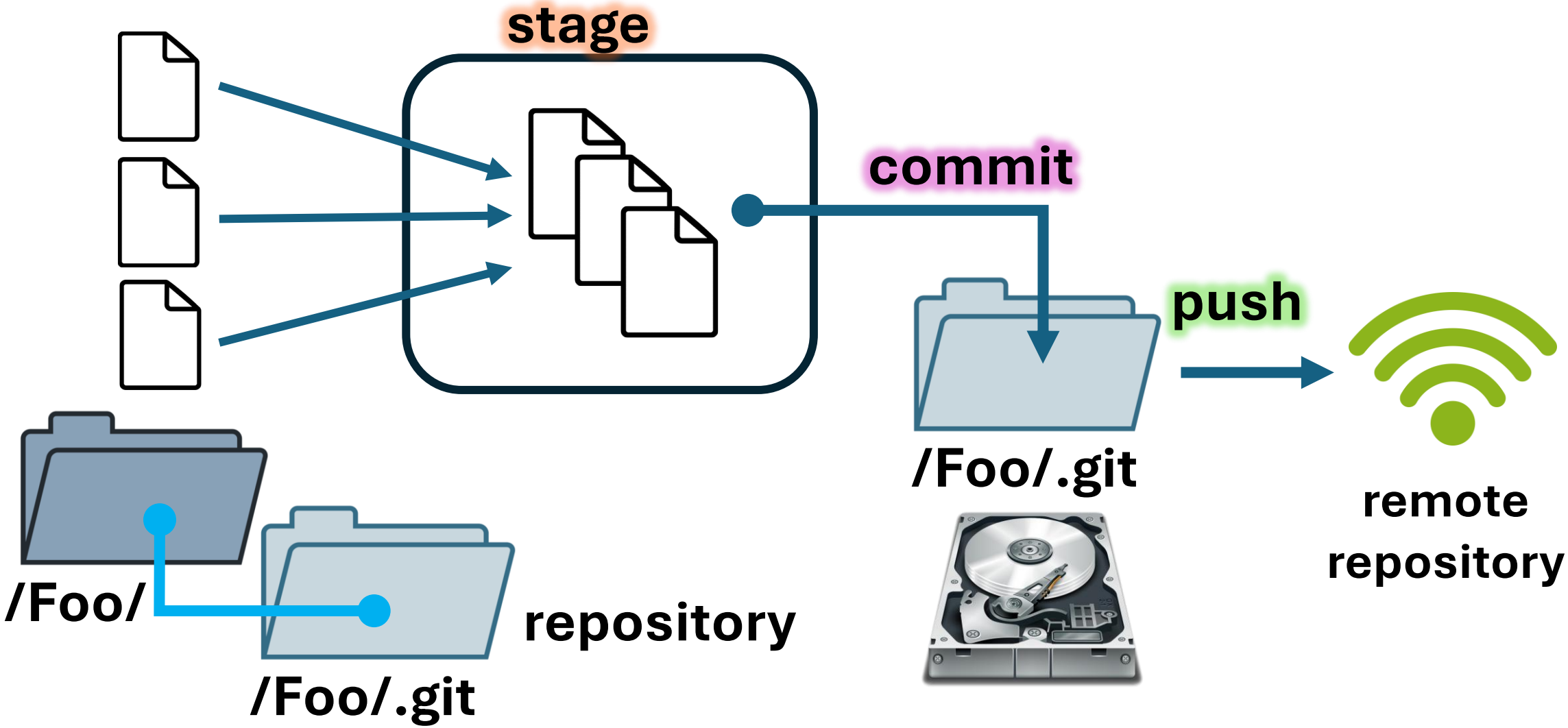
Why Version-Control?



Why Version-Control?



git Backup


stage
commit
push
/Foo/
repository
/Foo/.git
/Foo/.git
remote
repository

git Stores copies of changed files

- Every time you commit, git stores a copy/reference to the whole file (not just changes)

- Changed files: stored in-whole
- Unchanged files: stores a reference to the last updated copy in repo.

- Each commit saves the whole staging-area

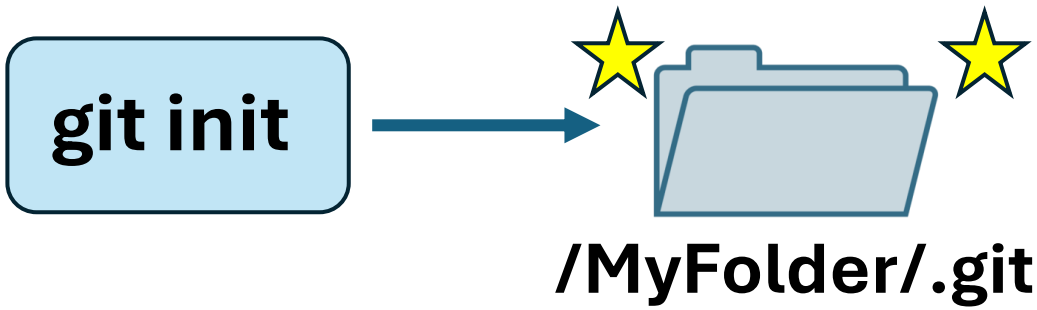
Setting up local git

- BASH terminal (default for UNIX), gitBash for Windows

ID
- **git config --global user.name** “samplename”
- **git config --global user.email** “abc@hotmail.com”

- Go to your project’s folder

cd C:/Users/…/MyFolder (note slash direction)



Staging Files

- Staging a file only prepares the current copy.
- If you change before committing, you'll need to restage.

  git add filename.py

  git add foldername

- To stage all files:

  git add . or **git add –A**

- Remove from the staging area:

  git restore filename.py or foldername

Check status before committing…

- Check status:

  git status

```
$ 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)
        new file:   tmpFolder/asdf.txt

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

Committing

- Commits all staged files at once
- Doesn’t stage/commit empty folders

- Include a useful description

git commit -m “A useful description of changes”

```
[main eff91a9] Test
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 tmpFolder/asdf.txt
```

git structure

- View history of commits using:

git log

```
$ git log
commit ee403f8d0dddd828e9af02e48bea254bb38b9519 (HEAD)
Author: burnerspam447 <stats447uiuc@gmail.com>
Date:   Thu Sep 19 21:58:58 2024 -0500

    newone

commit eff91a9f14fc935a4ea9ca61aff35bb8d87ae297
Author: burnerspam447 <stats447uiuc@gmail.com>
Date:   Thu Sep 19 21:23:38 2024 -0500

    Test
```

git log --oneline

```
$ git log --oneline
ee403f8 (HEAD) newone
eff91a9 Test
854d762 (origin/main, tmpBranch)
```

- HEAD denotes the commit git is currently “pointing” at
- (last commit by default)

Tagging

- git is natively addressed using a hash system
- This means your commits are named stuff like:

`854d762d59ed28644d7174e155d55a9310bdbc99`

  - Although you can (usually) reference with shortened: 854d76

- Tags make it easier to reference major timepoints

  git tag tagname (commitname)

- You can also reference chronologically

  HEAD~2 is 2 steps before current HEAD/ active commit

git Checkout

- “Checkout” restores your repository to the state at that commit
- Files will rollback
- Deleted files will be restored
- However, new files (since that commit) will remain unchanged

  git checkout commitname
- This also repoints HEAD to commitname

  You can repeat: **git checkout HEAD~1**

  to go back through commits one-at-a time

the git Multiverse

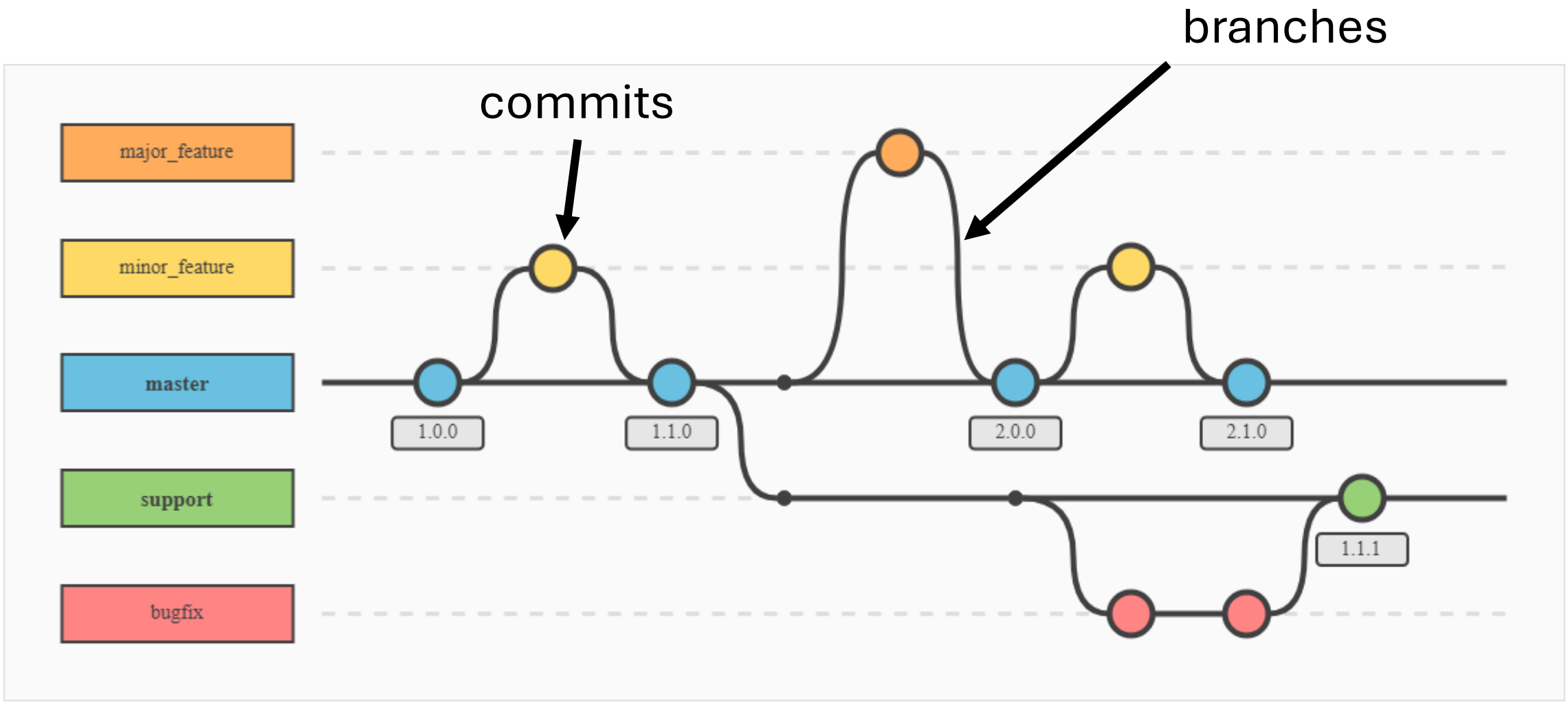
ChatGPT

is git checkout more similar to Rick & Morty time travel or to time travel in the Disney Avenger's cinematic universe?

Verdict:

`git checkout` is **more like the Avengers' approach** to time travel. You're revisiting points in time or branching off without causing chaos to the main timeline—unless you make significant changes, which can spawn a new branch of history (like an alternate universe in *Endgame*). Meanwhile, *Rick & Morty*'s time travel is a bit more akin to reckless exploration, akin to chaotic operations like force-pushing, rebasing, or resetting!

the git Multiverse



Creating a branch

- If you want to put your current progress in a new branch

  (splitting at last commit)

  git branch branchname

  git commit .....

- To start a branch at a specific point

  git checkout commitname **–b** branchname

- Switch branches

  git switch branchname

Merging

- Git merges are directed (A into B) rather than mixing A+B

  git switch branch1

  git merge branch2

- Conflicts will prevent merging:

  git merge -X ours branch2 defer to branch1

  git merge –X theirs branch2 defer to branch2

- To resolve each conflict in vimdiff:

  git mergetool –tool=vimdiff

gitHub concepts

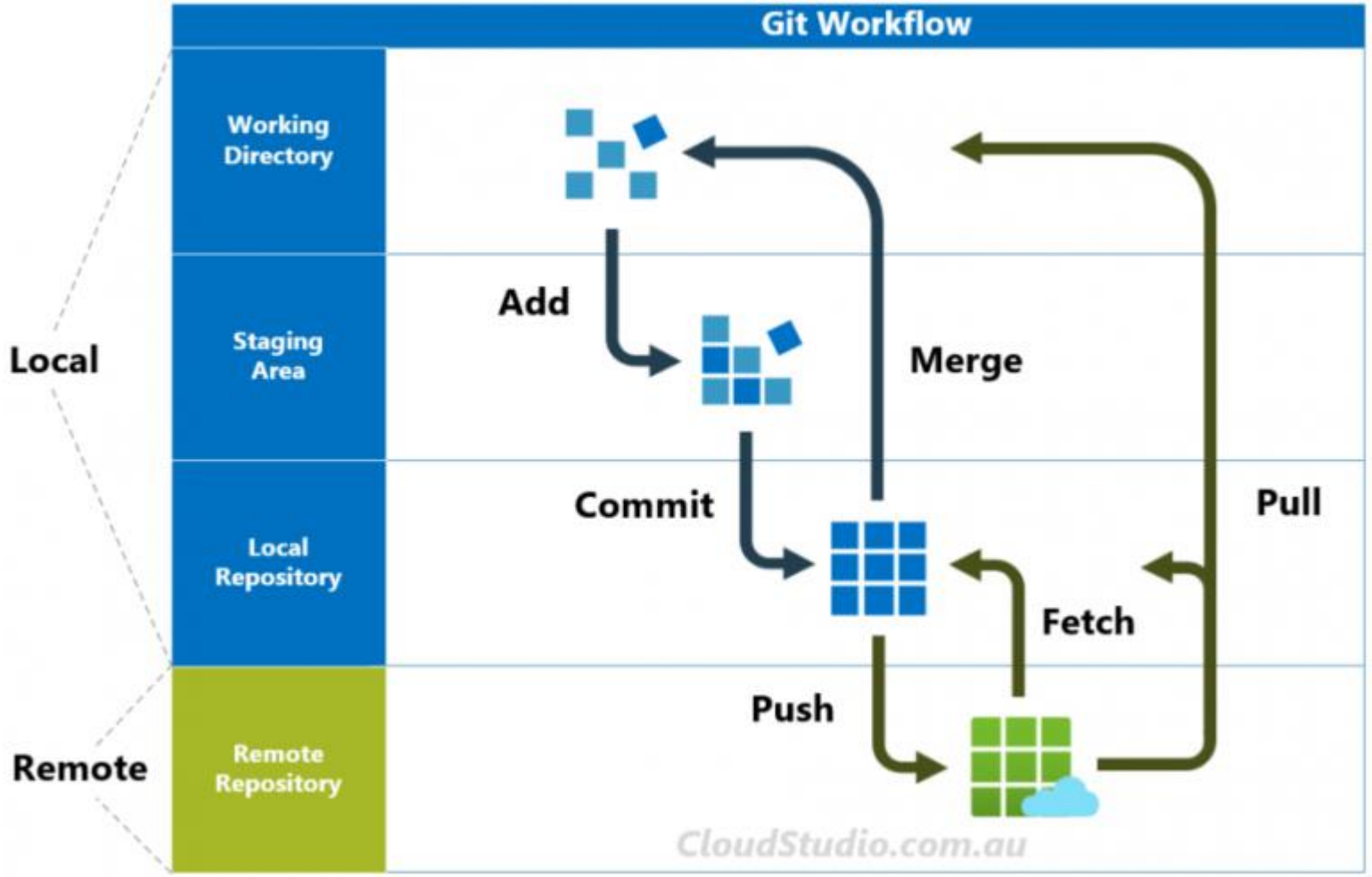
- Setup:
- Create SSH key & link

(Instructions online)

git remote add origin
git@github.com:....git~

- **Push:** upload a commit

  git push –u origin etc.
- **Pull:** switch & download
- **Fetch:** download
- **Clone:** download full repo



vsCode demo.....

Fin