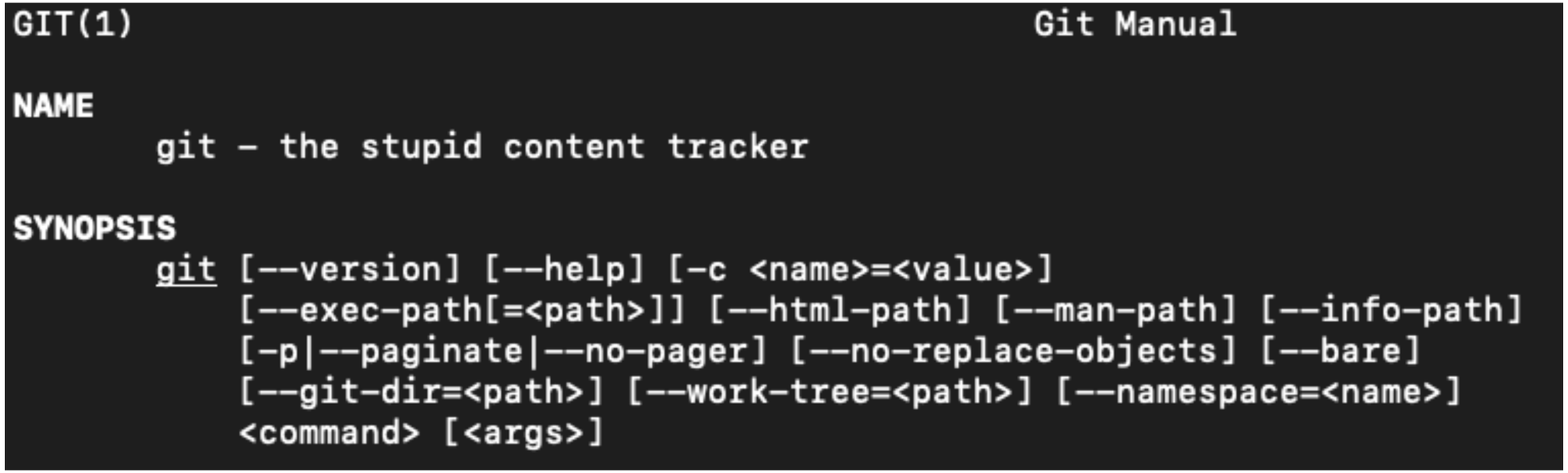
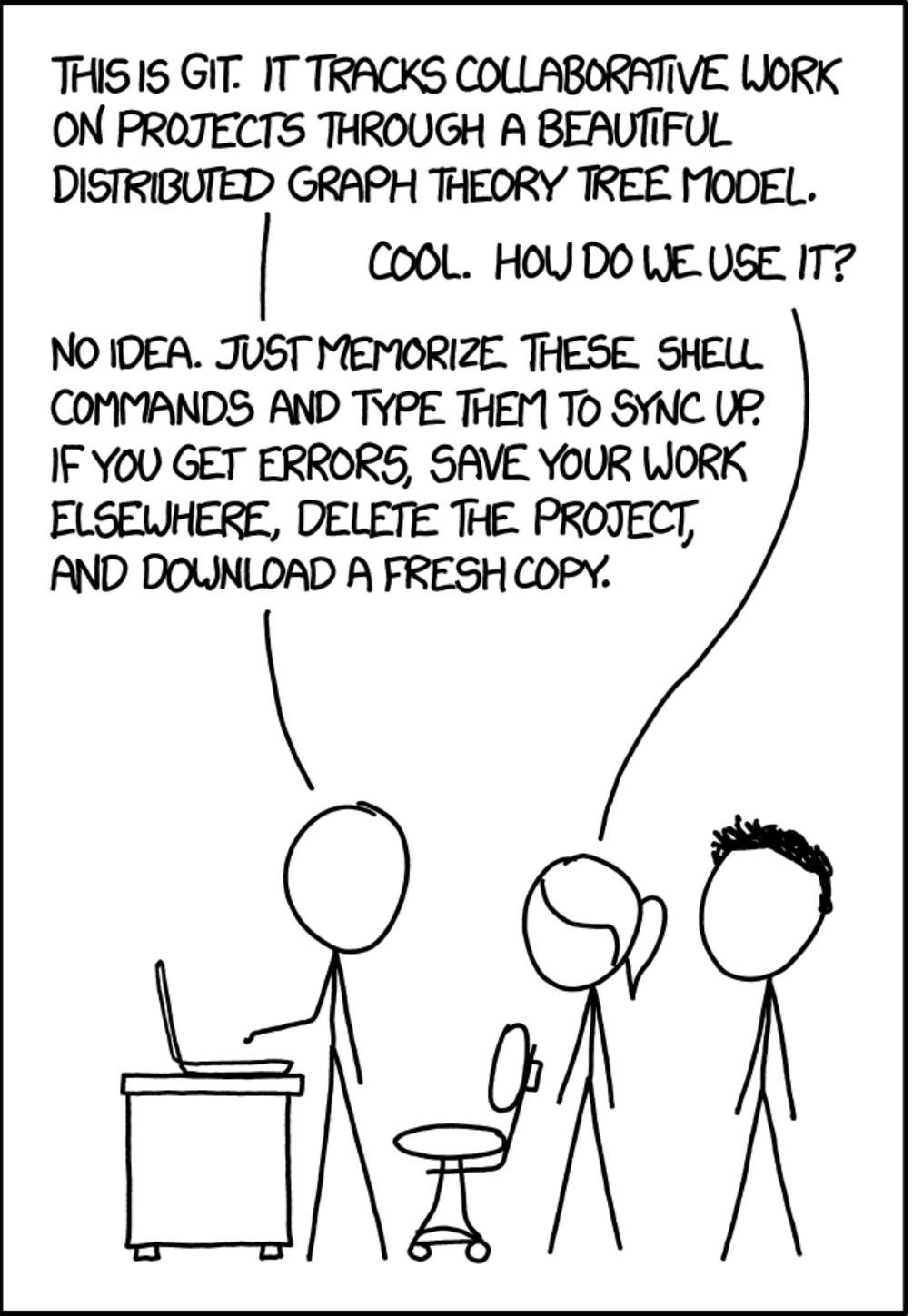


Git and Github

https://www.atlassian.com/git/tutorials

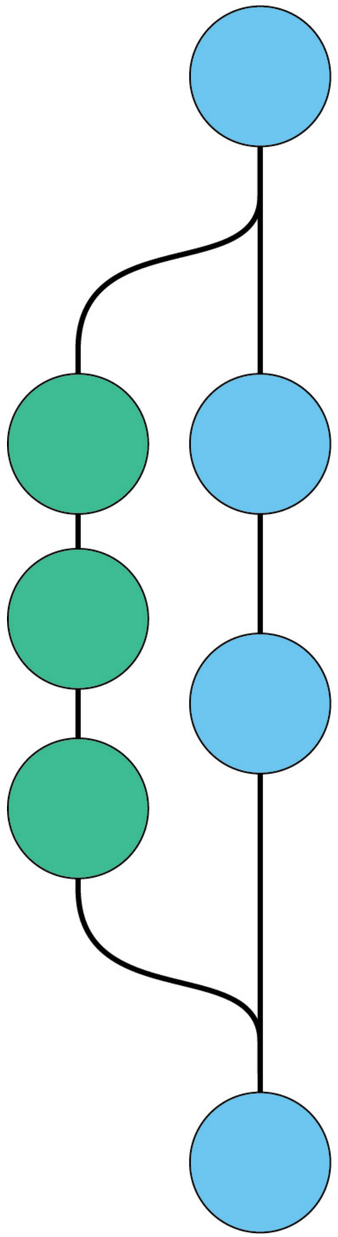
https://www.youtube.com/playlist?list=PL0lo9MOBetEHhfG9vJzVCTiDYcbhAiEqL



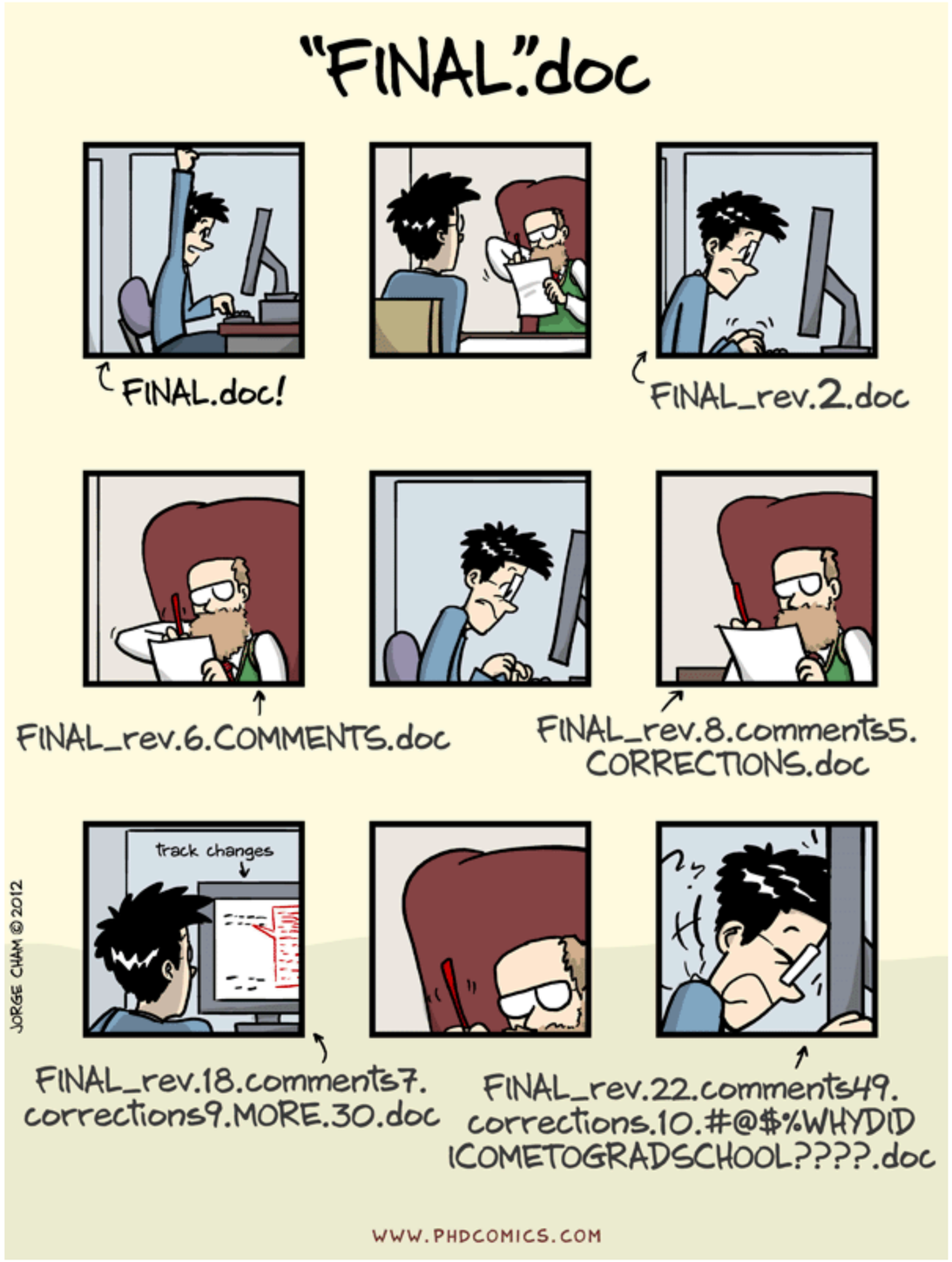
https://xkcd.com/1597/

What is git?

- Version control software
  - System that keeps records of your changes
  - Allows for collaborative development (when paired with a remote server, like GitHub)
  - Tracks **who** made **which** changes **when**
  - Allows you to revert to a previous state
- Started in 2005 by Linus Torvalds to help with Linux development
- Open source, free, widely used

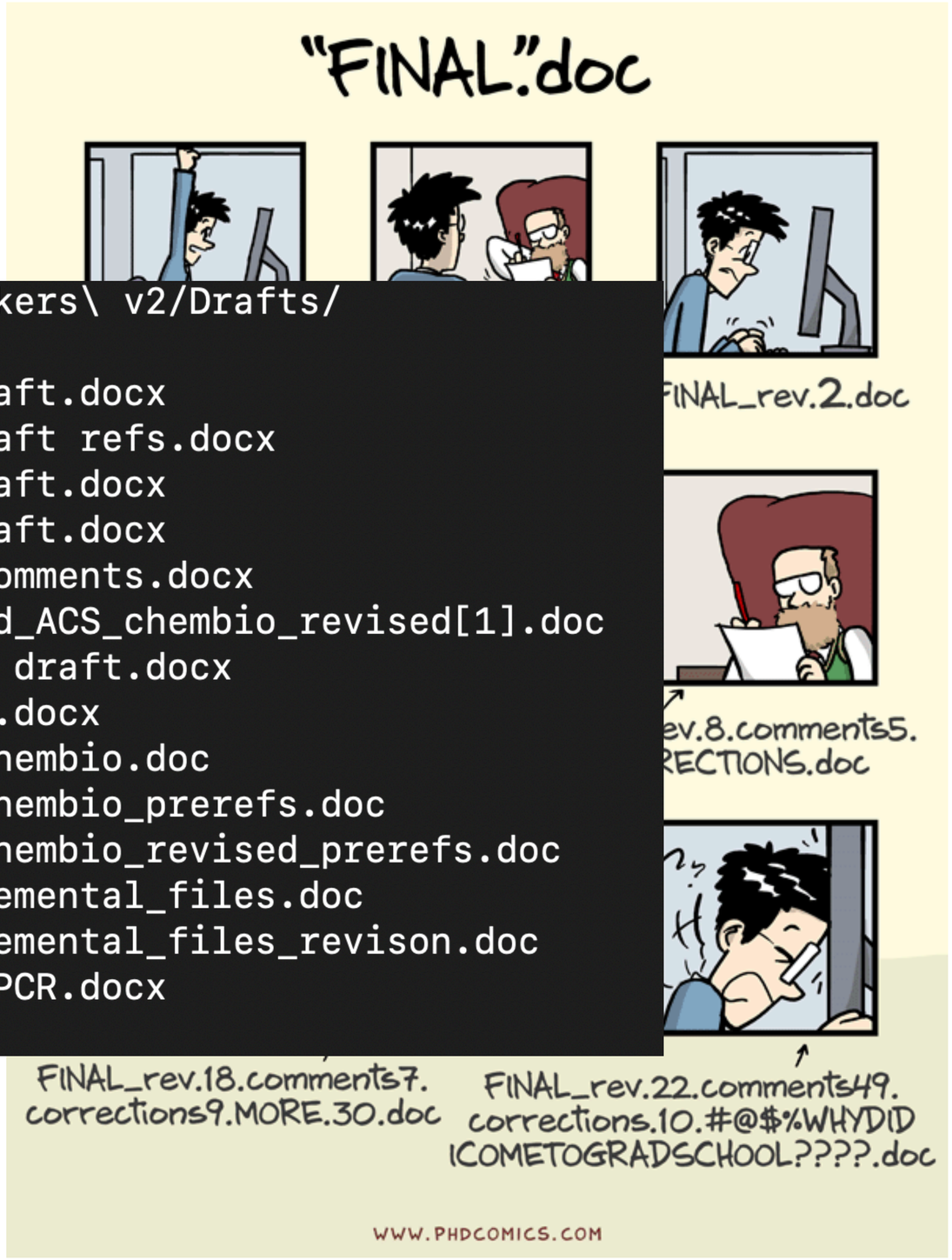


Why use git?


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Why use git?

```
[10:12:40 ~] $ cd ~/Documents/Manuscripts/ACS\ chembio/Linkers\ v2/Drafts/
[10:57:14 Drafts] $ ls
00 Linkers draft.docx                          04 Linkers draft.docx
00 Supplement.docx                             05 Linkers draft refs.docx
000_Coonrod_ACS_chembio_revised.doc            05 Linkers draft.docx
000_Coonrod_ACS_chembio_revised_prerefs.doc    06 Linkers draft.docx
01 Linker - figures only.docx                  AB Reviewer comments.docx
01 Linkers draft refs - MJB.docx               AB_000_Coonrod_ACS_chembio_revised[1].doc
01 Linkers draft refs.docx                     AB_05 Linkers draft.docx
01 Linkers draft.docx                          Aaron methods.docx
01 Supplement.docx                             Coonrod_ACS_chembio.doc
02 Linkers draft refs CT edits.docx            Coonrod_ACS_chembio_prerefs.doc
02 Linkers draft refs.docx                     Coonrod_ACS_chembio_revised_prerefs.doc
02 Linkers draft.docx                          Coonrod_Supplemental_files.doc
03 Linkers draft refs.docx                     Coonrod_Supplemental_files_revison.doc
03 Linkers draft.docx                          Heptamidine qPCR.docx
04 Linkers draft refs.docx
```

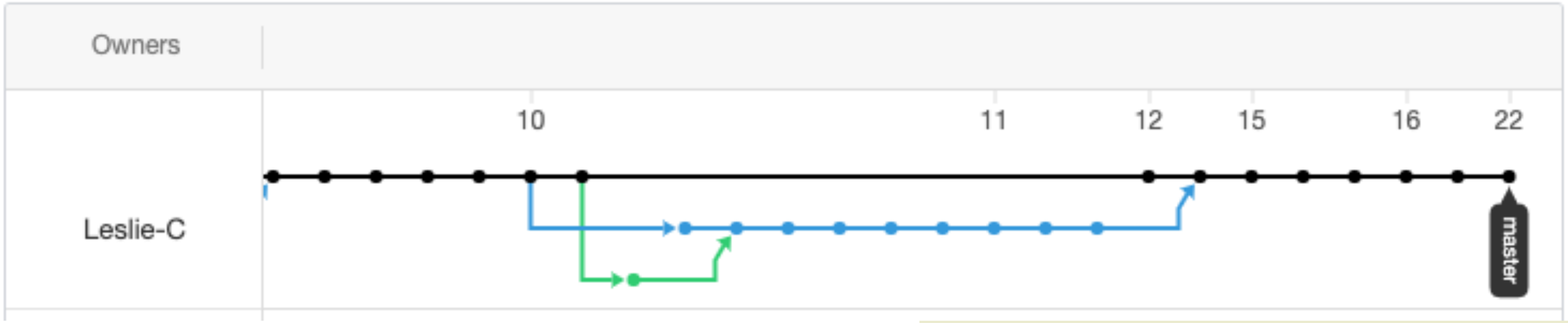


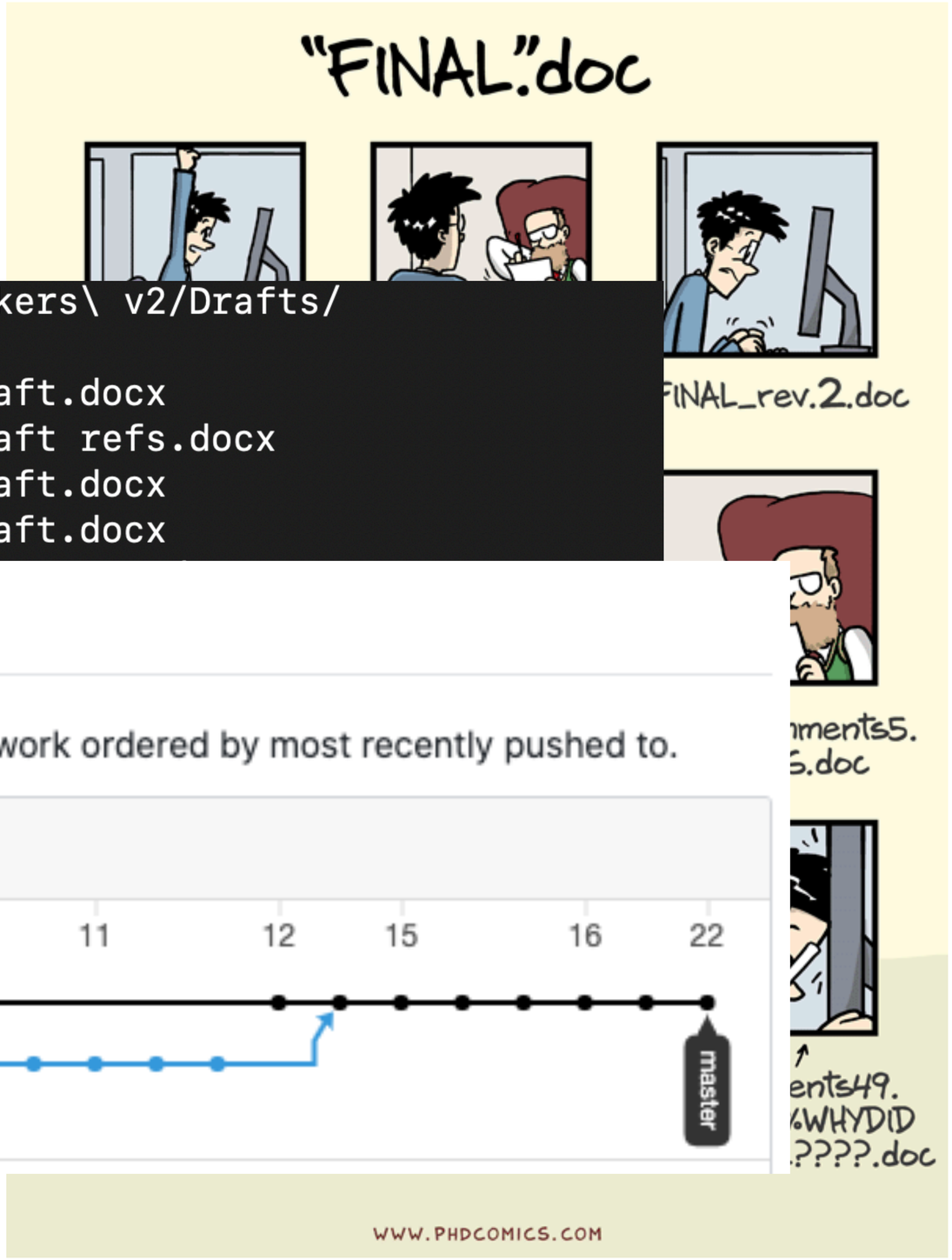
Why use git?

```
[10:12:40 ~] $ cd ~/Documents/Manuscripts/ACS\ chembio/Linkers\ v2/Drafts/
[10:57:14 Drafts] $ ls
00 Linkers draft.docx                          04 Linkers draft.docx
00 Supplement.docx                             05 Linkers draft refs.docx
000_Coonrod_ACS_chembio_revised.doc            05 Linkers draft.docx
000_Coonrod_ACS_chembio_revised_prerefs.doc    06 Linkers draft.docx
01 Linker
01 Linker
01 Linker
01 Linker
01 Supple
02 Linker
02 Linker
02 Linker
03 Linker
03 Linker
04 Linker
```

Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.





How does git work?
(on a superficial level)

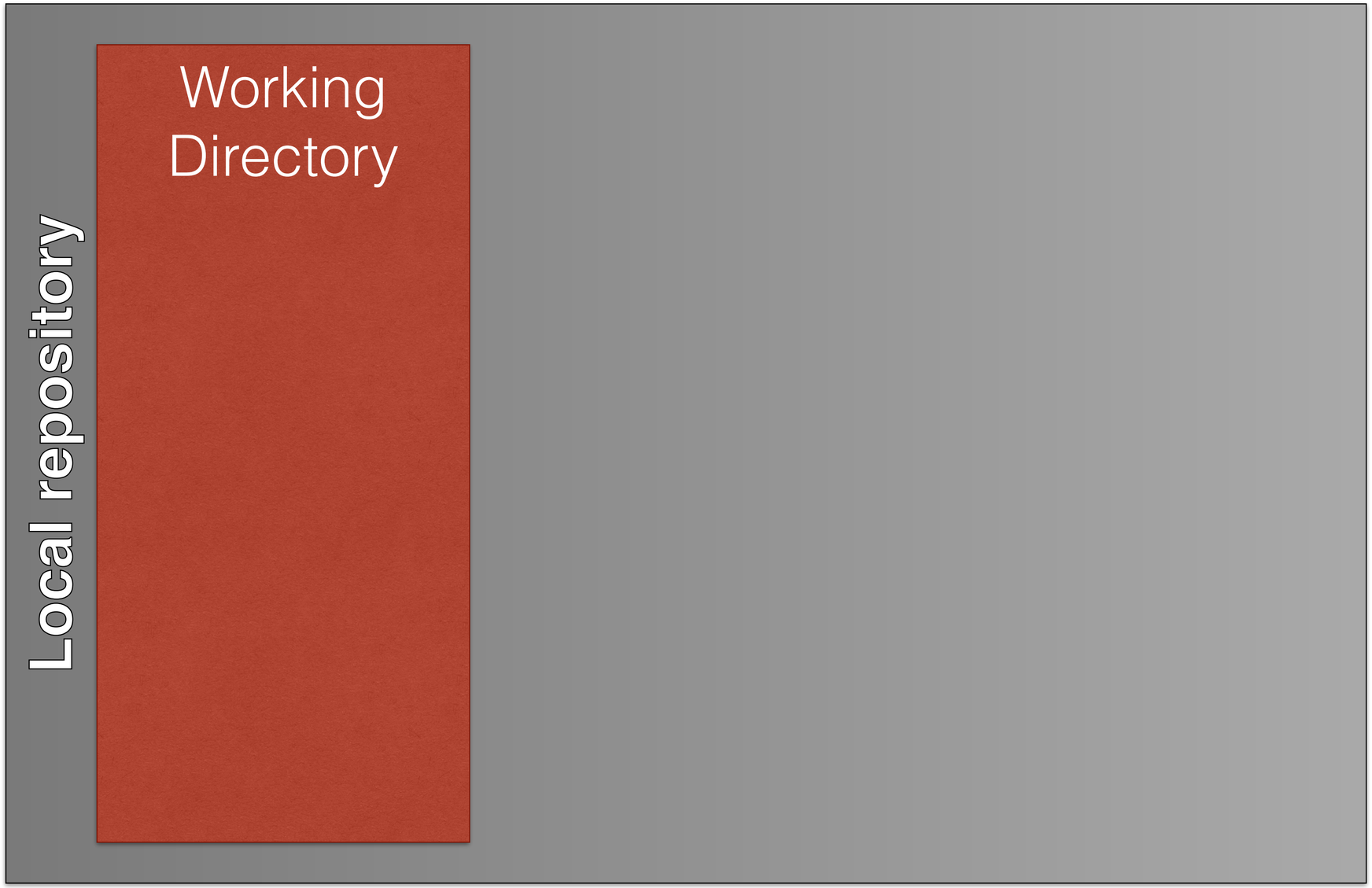
- Creates “snapshots” of your work – commit
  - Records what all files in a repository look like at a given point in time
  - User decides when to create a commit
  - Can move backwards (and forwards) through commits

How does git work?
(on a superficial level)

- Creates “snapshots” of your work – commit
  - Records what all files in a repository look like at a given point in time
  - User decides when to create a commit
  - Can move backwards (and forwards) through commits
- A commit contains:
  1. Information about how the files changed from the previous commit
  2. A reference to the previous commit
  3. A hash name
     `7845f1dbb92bcb190d6849e08e05ce49e5071cec`

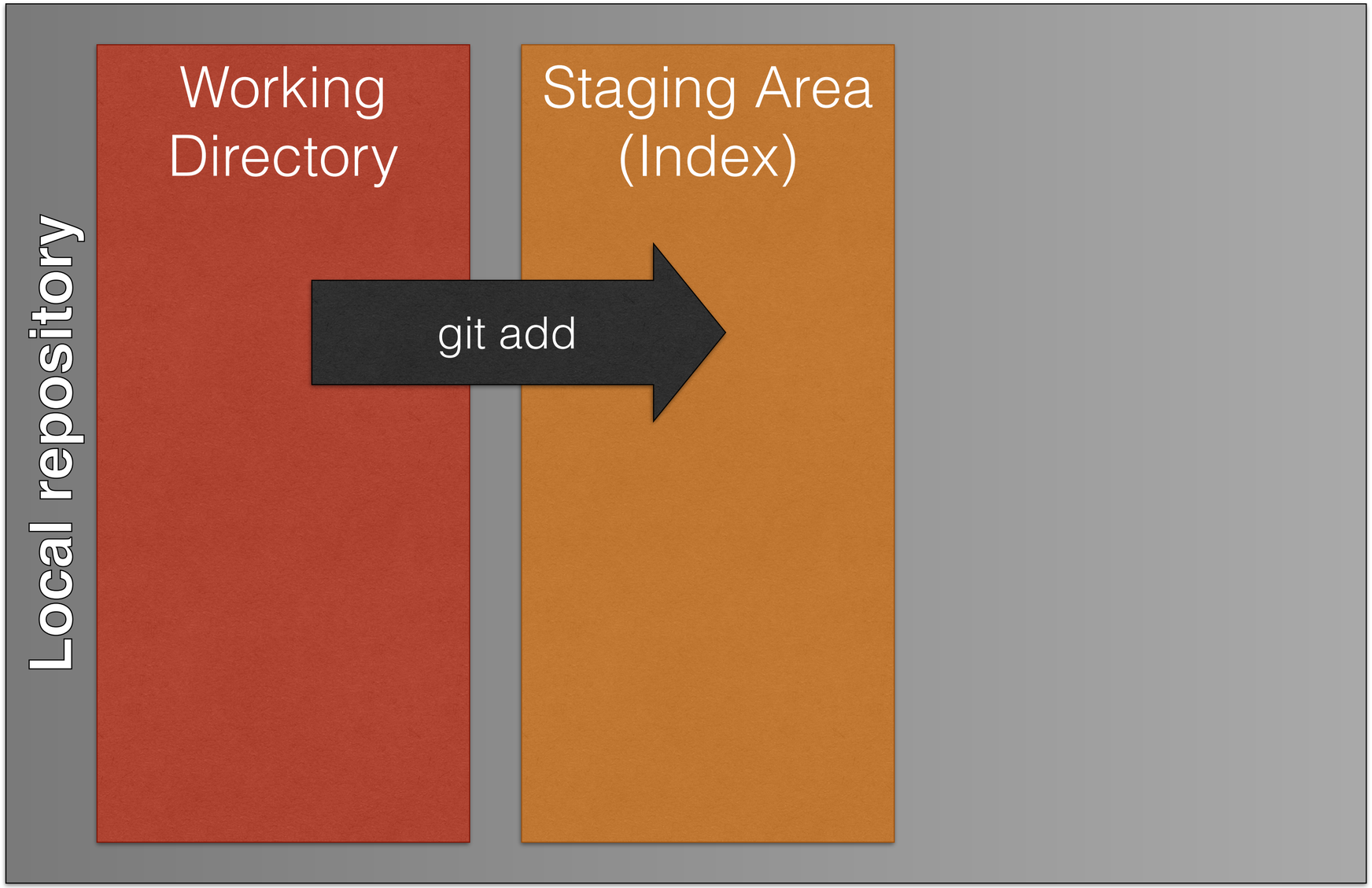
How does git work?

(on a superficial level)



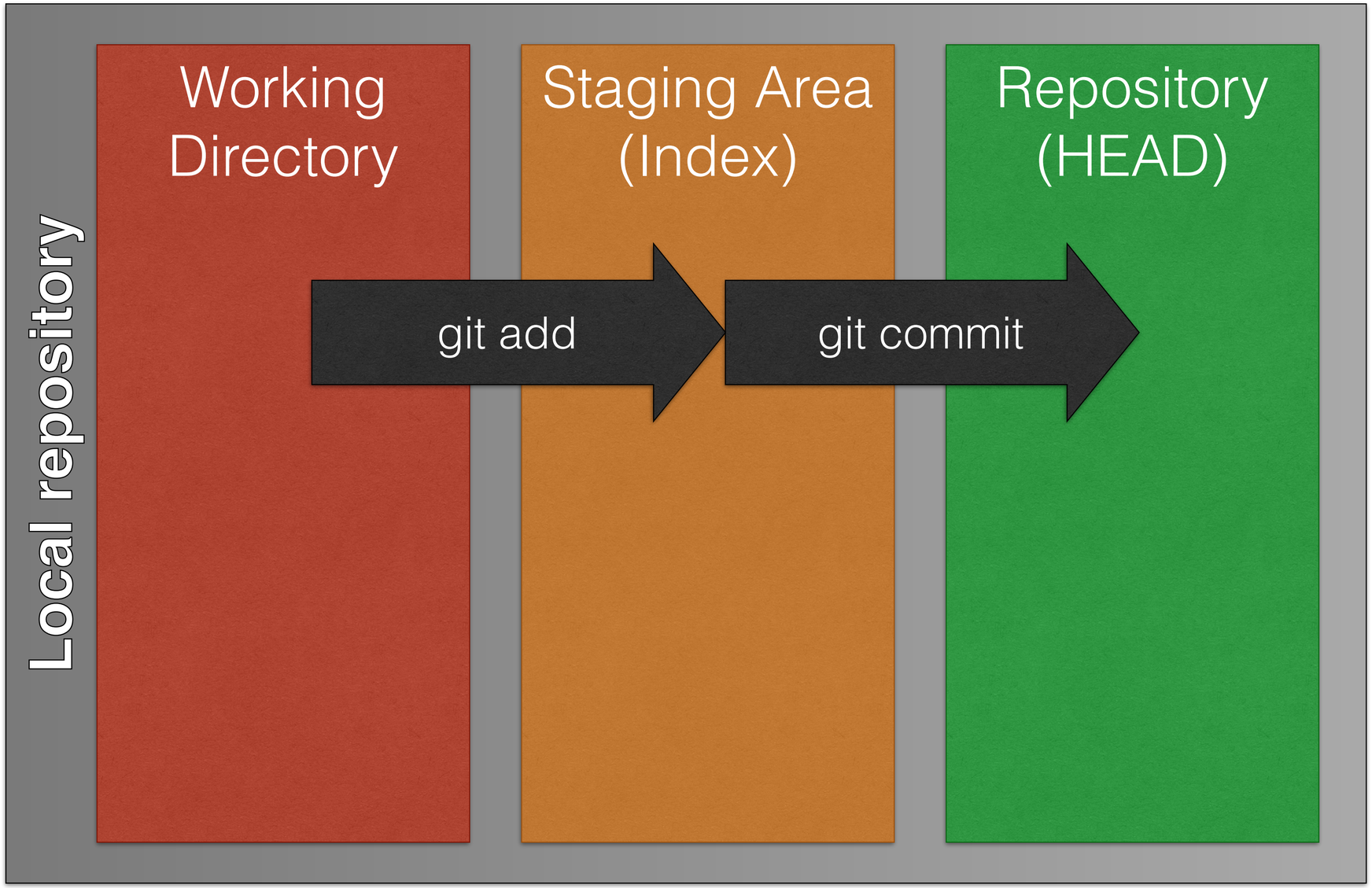
How does git work?

(on a superficial level)



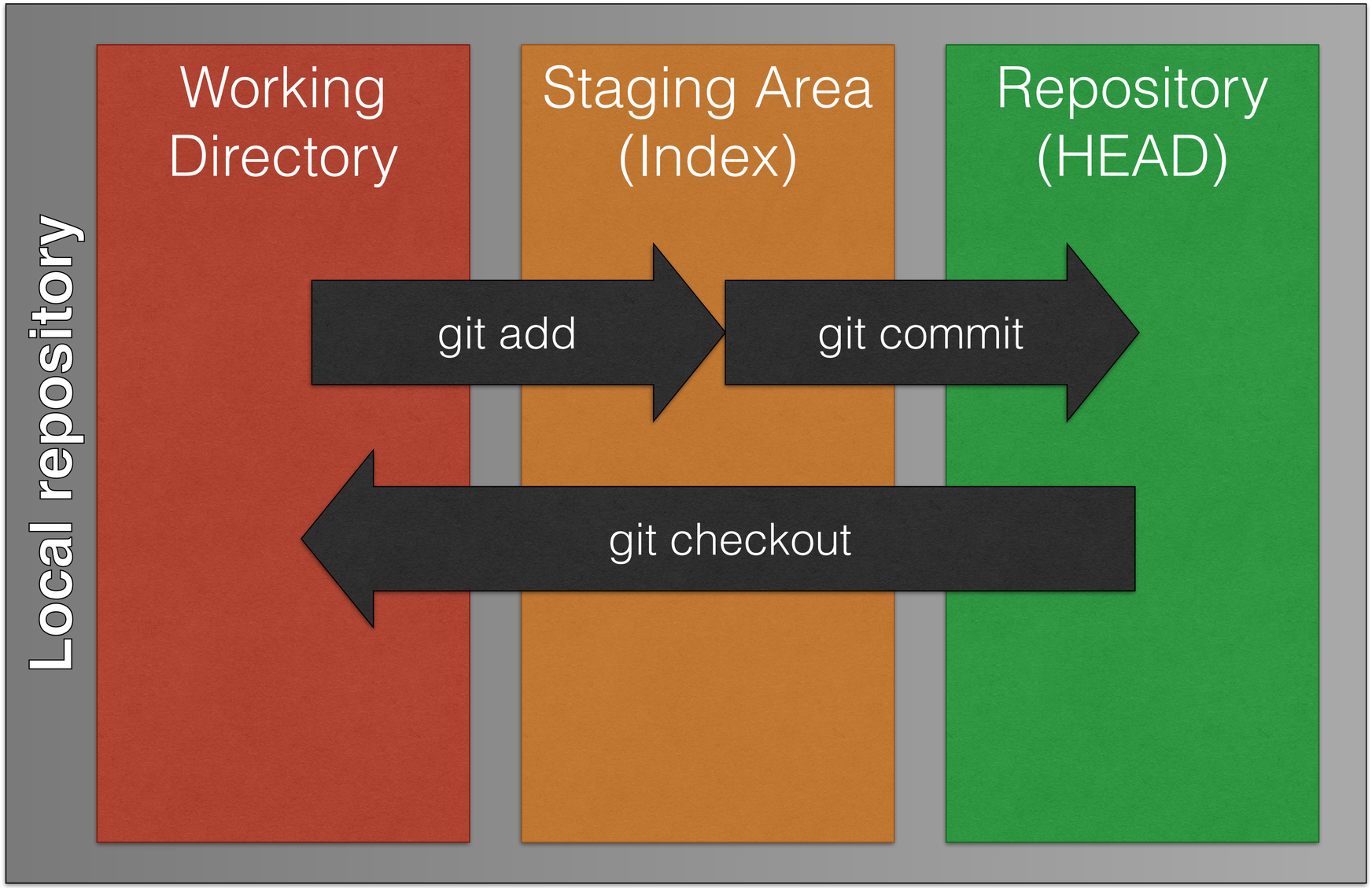
How does git work?

(on a superficial level)



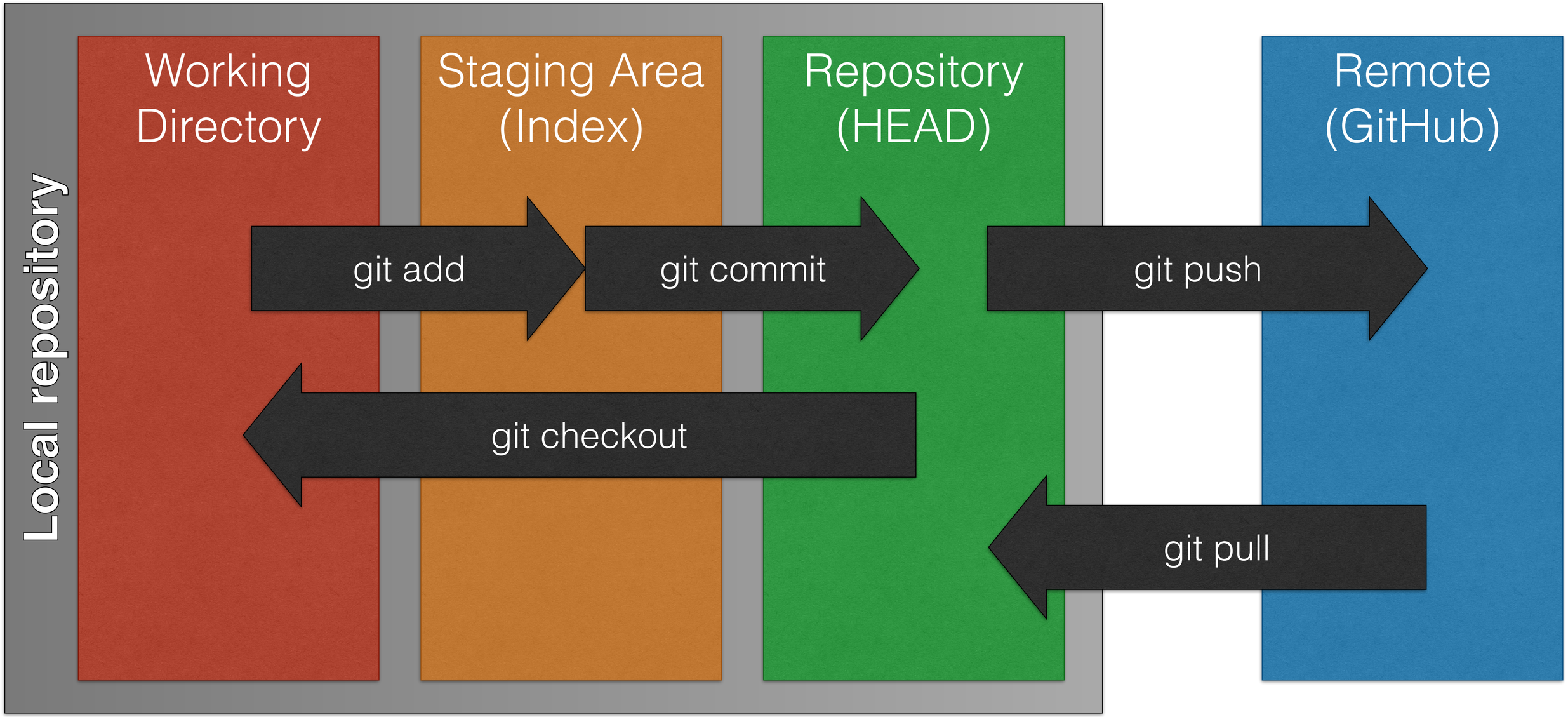
How does git work?

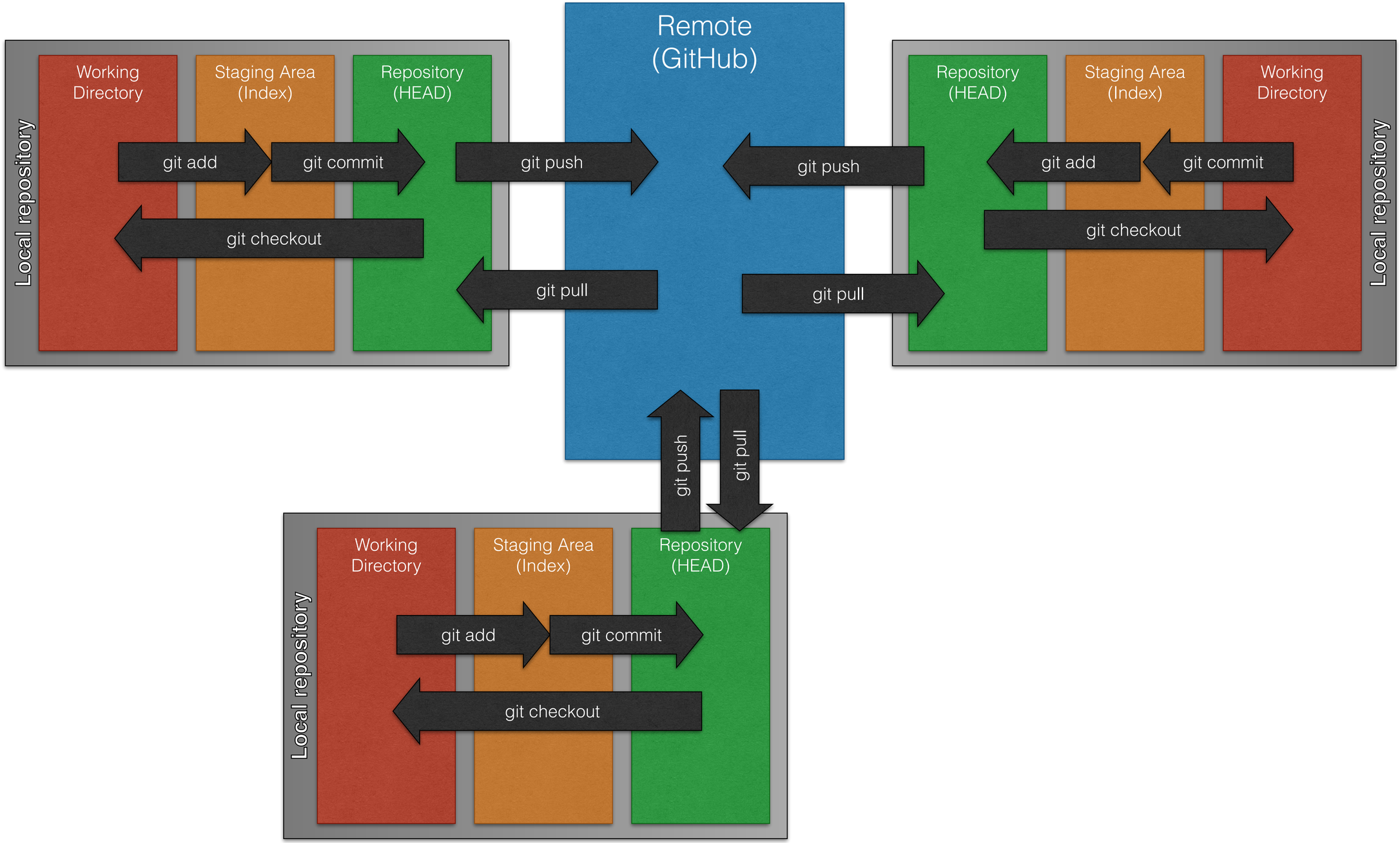
(on a superficial level)



How does git work?

(on a superficial level)

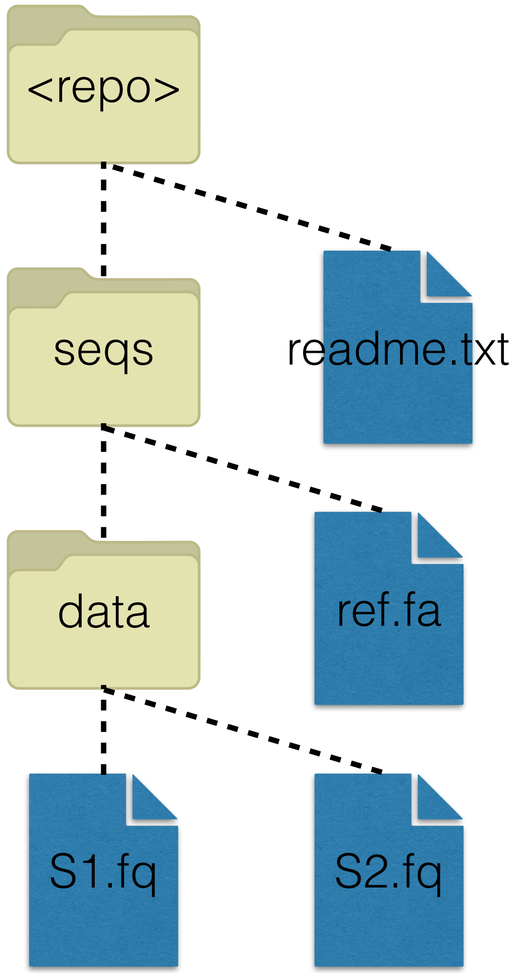



Remote
(GitHub)
Local repository
Working
Directory
Staging Area
(Index)
Repository
(HEAD)
git add
git commit
git checkout
git push
git pull
Repository
(HEAD)
Staging Area
(Index)
Working
Directory
Local repository
git add
git commit
git checkout
git push
git pull
Local repository
Working
Directory
Staging Area
(Index)
Repository
(HEAD)
git add
git commit
git checkout

Okay, that's cool and all...but how does git REALLY work?

How does git work?
(For real)

- Git models:
  - All the data within your repo as "trees" (folders) and "blobs" (files)
  - History – directed acyclic graph



Git data structures

OBJECTS	blob	array<bytes>
	tree	map{string[filename], location[tree or blob]}

	commit	custom_structure{
		parents: array<commit>
		author: string
		message: string
		snapshot: tree
		}

objects stored by		map{string[hash(o)], object}

Git data structures

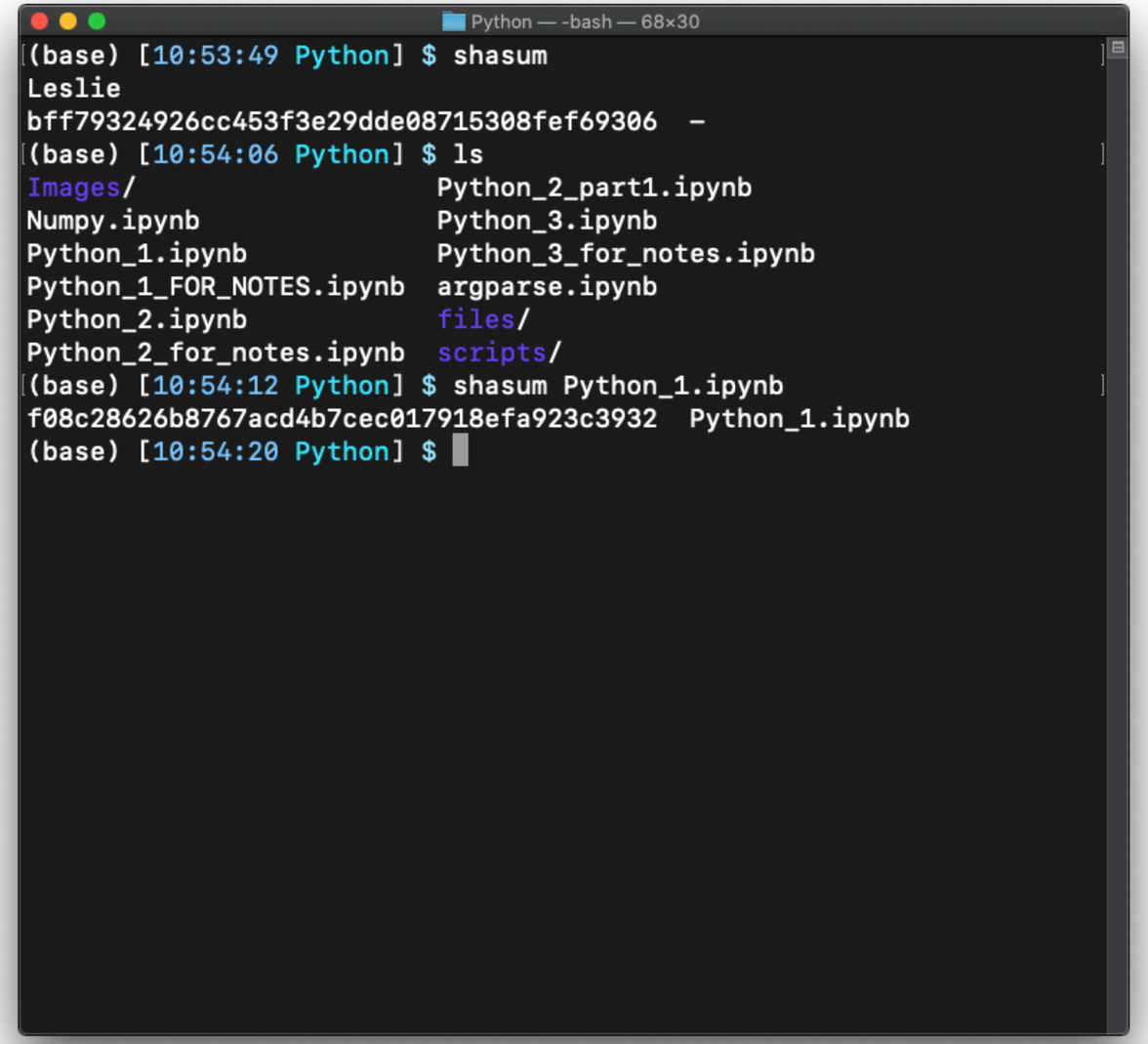
OBJECTS	blob	array<bytes>
	tree	map{string[filename], location[tree or blob]}

	commit	custom_structure{
		parents: array<commit_ID>
		author: string
		message: string
		snapshot: tree_ID
		}

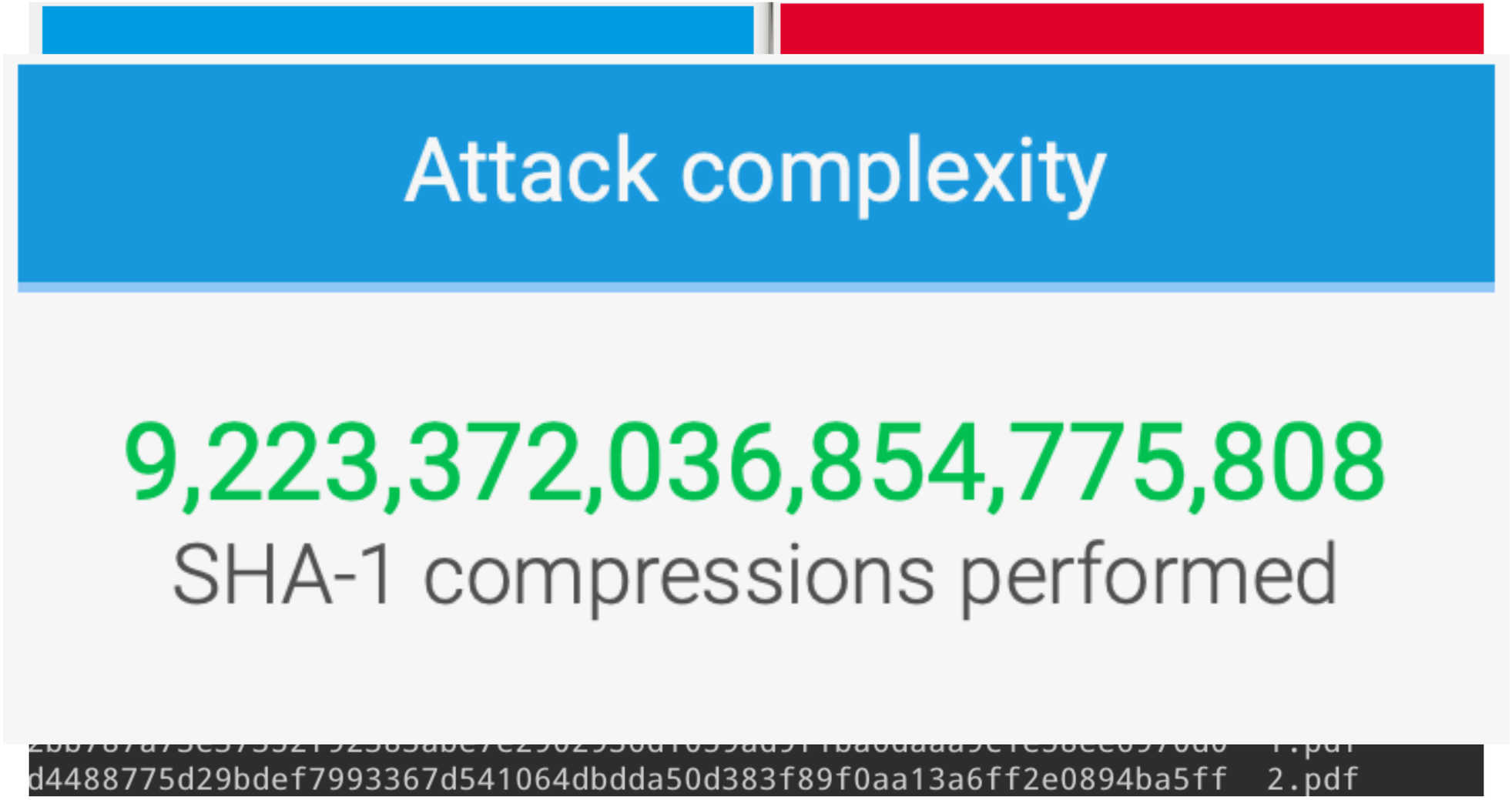
objects stored by		map{string[hash(o)], object}

Aside: SHA-1???

- SHA-1: Secure Hash Algorithm 1
- input → 160 bit value
- 40 digit long hexadecimal number
- 

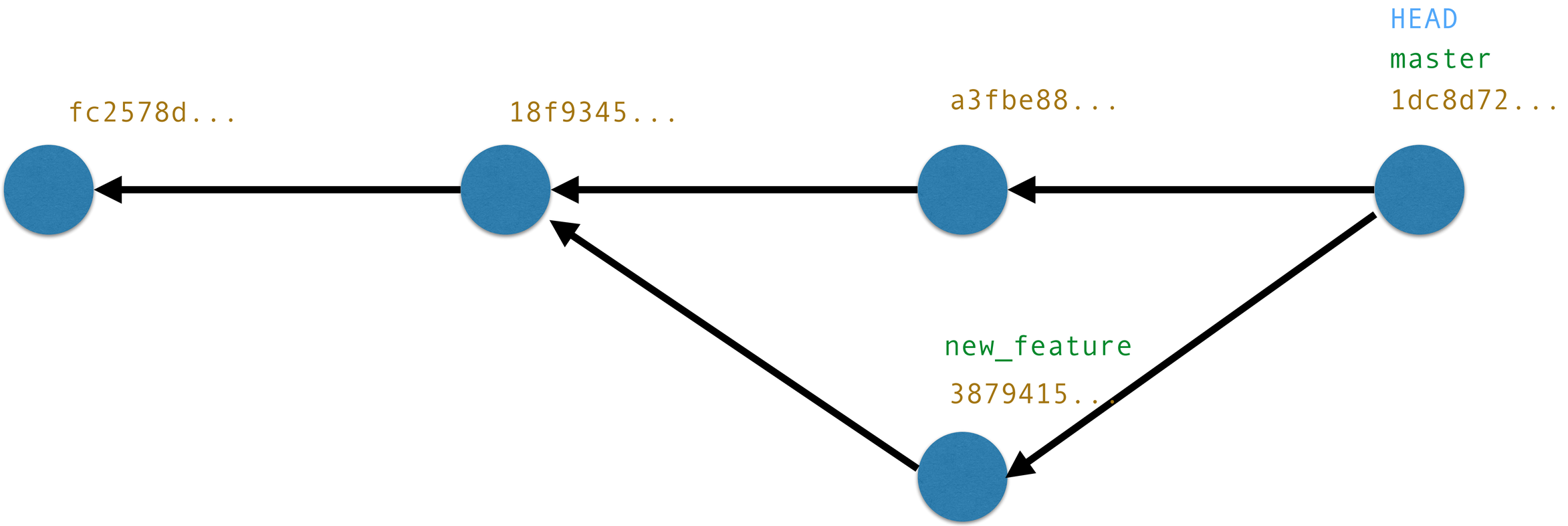


Aside of the aside: SHA-1 is broken



https://shattered.io/

Commit graph



Let's do it!

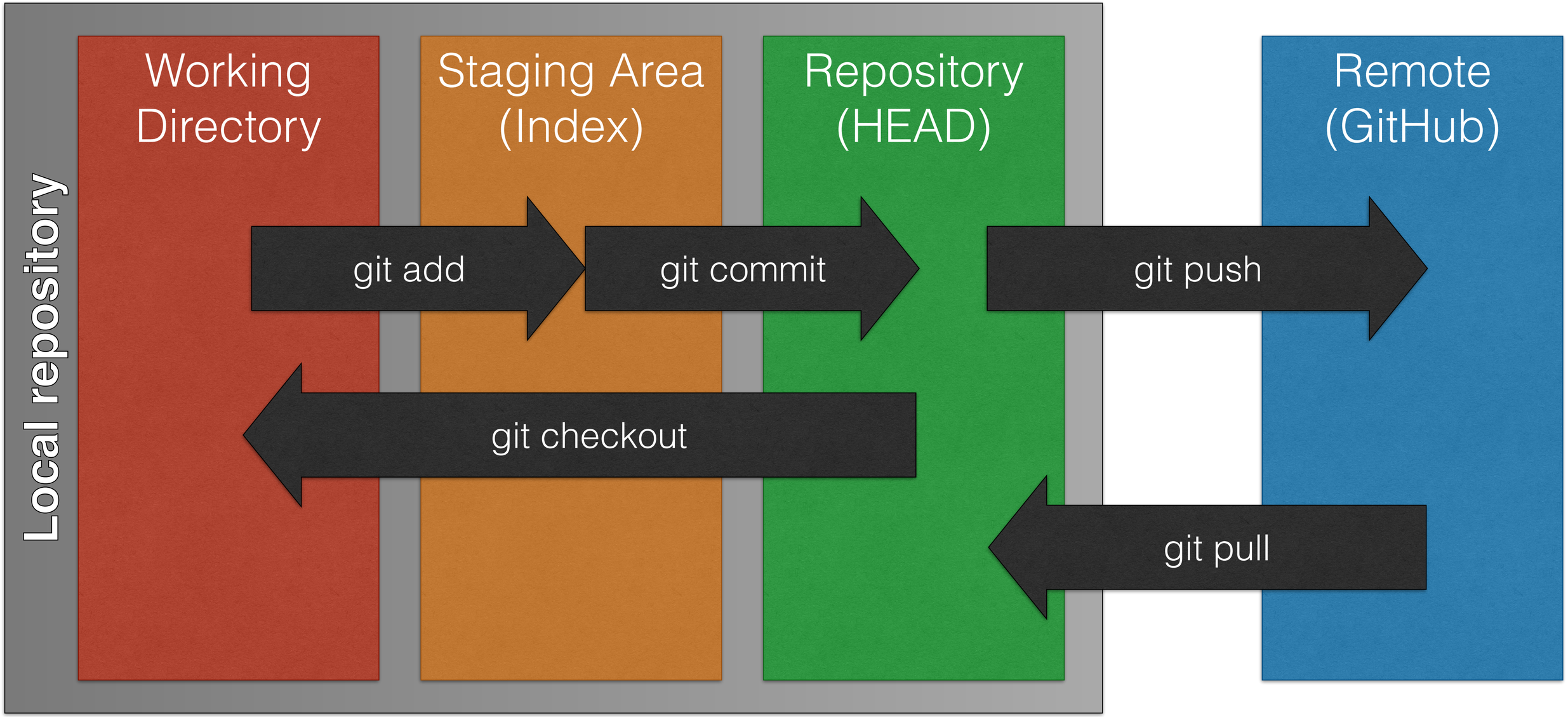
```
#create a repo
$ mkdir git-demo
$ cd git-demo
$ git init
Initialized empty Git repository in ~/git-demo/.git/

#check the status
$ git status
On branch master...

#create a file and add it to repo
$ echo "hello world" > hello.txt
$ git status
$ git add hello.txt
$ git status
$ git commit
[master (root-commit) 440250e] add hello
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
```

How does git work?

(on a superficial level)



WHO

made

WHICH

changes

WHEN

```
#check out the log
$ git log
commit 440250eb69de64d3a4f050a692f00a5905bbd498 (HEAD -> master)
Author: Leslie Coonrod <coonrod@uoregon.edu>
Date:   Wed Feb 2 13:03:25 2022 -0800

    add hello

#What's that crazy hex thing??? A commit object!
$ git cat-file -p 440250e
tree 68aba62e560c0ebc3396e8ae9335232cd93a3f60
author Leslie Coonrod <coonrod@uoregon.edu> 1643835805 -0800
committer Leslie Coonrod <coonrod@uoregon.edu> 1643835805 -0800

add hello

#we can drill down
$ git cat-file -p 68aba62
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad  hello.txt

$ git cat-file -p 3b18e5
hello world
```

```
#check out the log - not flattened!
$ git log --all --graph --decorate
#not super interesting with just 1 commit...add more!

#hop around in your history
$ git checkout <commit_hash>
Note: switching to...

#See that position of HEAD has changed
$ git log --all --graph --decorate

#move head pointer back
$ git checkout master

#compare differences between commits
$ git diff <commit_hash>
$ git diff <commit_hash> <file>
```

```
#Branching and merging! First create simple script and add/commit
$ git branch            #see all branches
$ git branch --v        #more verbose
$ git branch <name>     #create new branch with <name>
$ git log
$ git checkout <name>   #move pointer to branch <name>
$ git log

#do some stuff, get some branches

$ git merge <branch1>
$ git merge <branch2>

#look at conflict, fix!
$ nano <file_with_conflict>
$ git merge --continue
$ git log --all --graph --decorate --oneline

#be sure to detach HEAD at some point
```

```
#Take a look inside .git
$ cd .git
$ ls
$ cd objects/
$ ls

#cd into one of those folders

$ ls
$ git cat-file -p <sha>

#we can find ANYTHING that we're using git to track!
```

Remotes!

```
#create a GitHub repo with no readme
$ git remote add <name> <url>       #typically use origin for name
$ pit push <name> <local_branch>:<remote_branch>

#ON TALAPAS
T$ git clone <url> <name>

T$ git branch --set-upstream-to=origin/master
T$ git push

T$ git pull             #alternatively, git fetch; git merge
```

Other useful commands

- `git config` – https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config
- `git clone --depth=`n – avoid cloning entire history
- `git add -p <filename>` – stage smaller pieces for commit
- `git blame` – see who edited file
- `git stash` – save changes without committing
- `git stash pop` – get those changes back!
- `git reflog` – get those changes back
- `.gitignore`