

**NAME**

git - the stupid content tracker

**SYNOPSIS**

```
git [--version] [--help] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]
```

# Git and Github

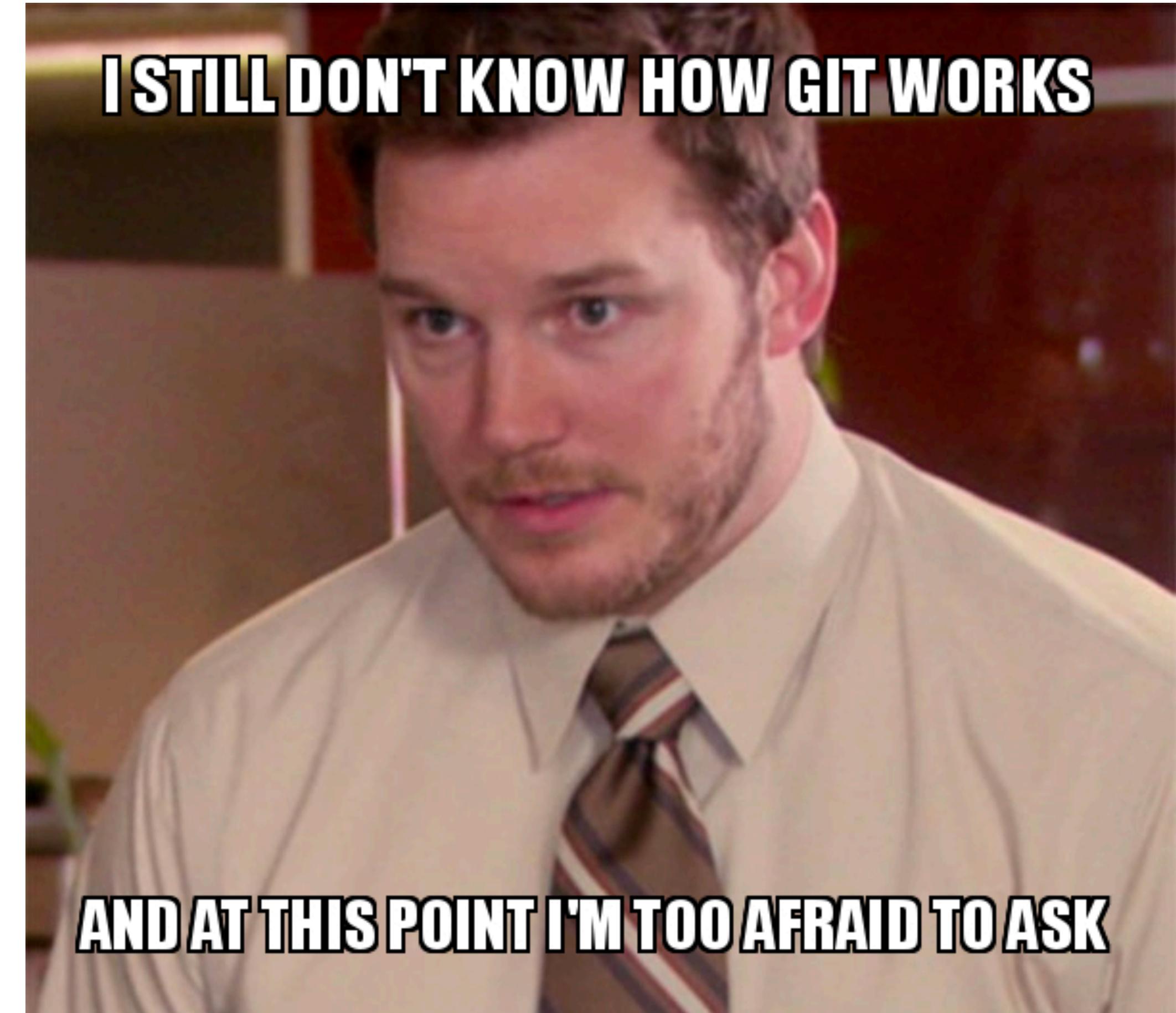
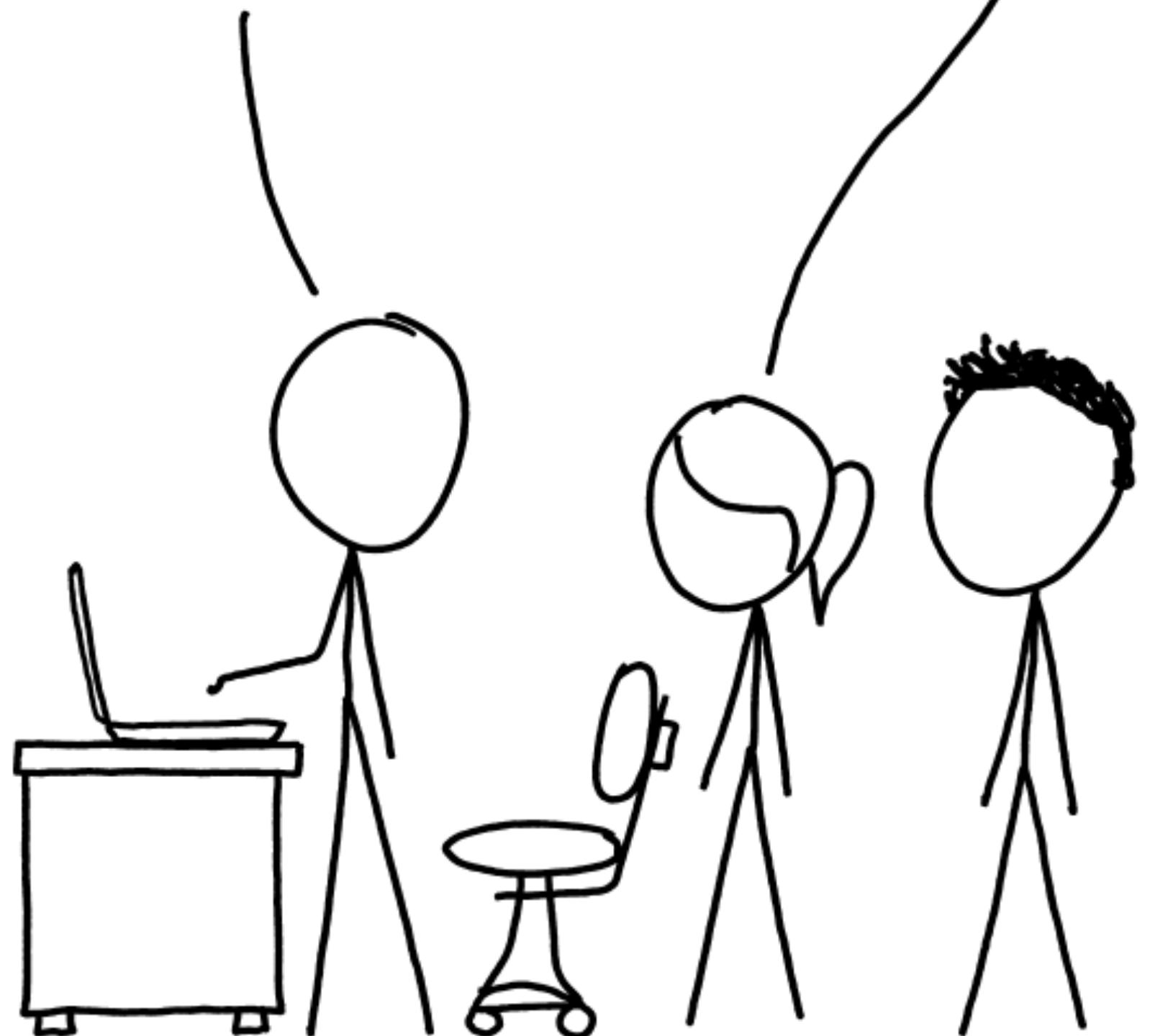
<https://www.atlassian.com/git/tutorials>

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

THIS IS GIT. IT TRACKS COLLABORATIVE WORK  
ON PROJECTS THROUGH A BEAUTIFUL  
DISTRIBUTED GRAPH THEORY TREE MODEL.

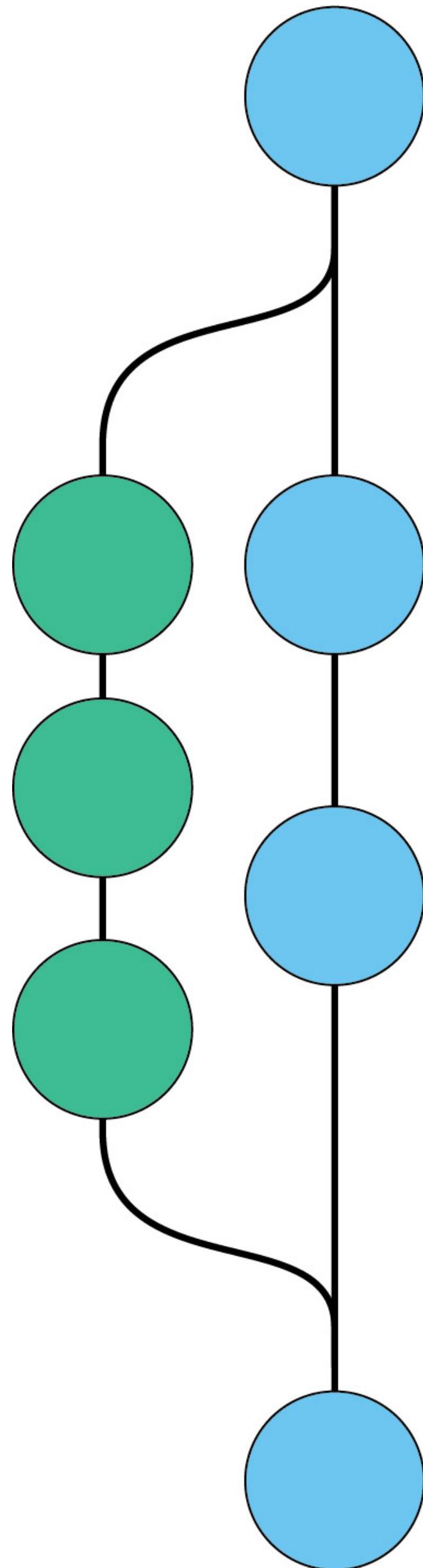
COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZIZE THESE SHELL  
COMMANDS AND TYPE THEM TO SYNC UP.  
IF YOU GET ERRORS, SAVE YOUR WORK  
ELSEWHERE, DELETE THE PROJECT,  
AND DOWNLOAD A FRESH COPY.



# 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

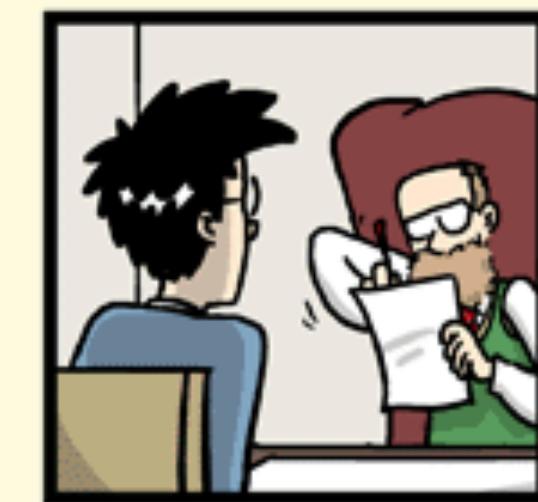


# Why use git?

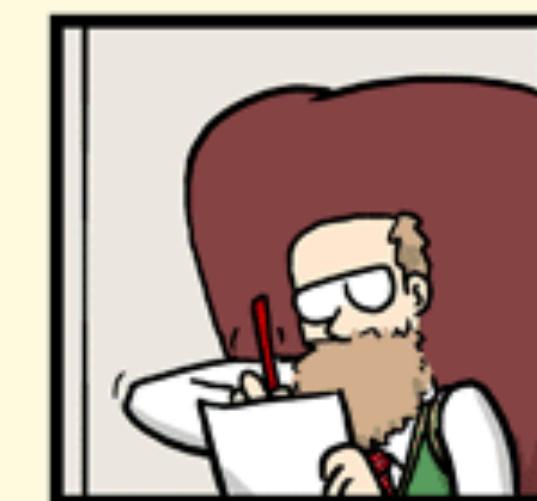
"FINAL".doc



FINAL.doc!



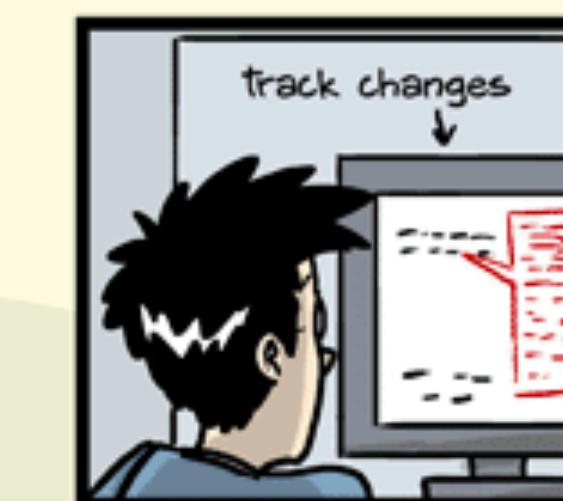
FINAL\_rev.2.doc



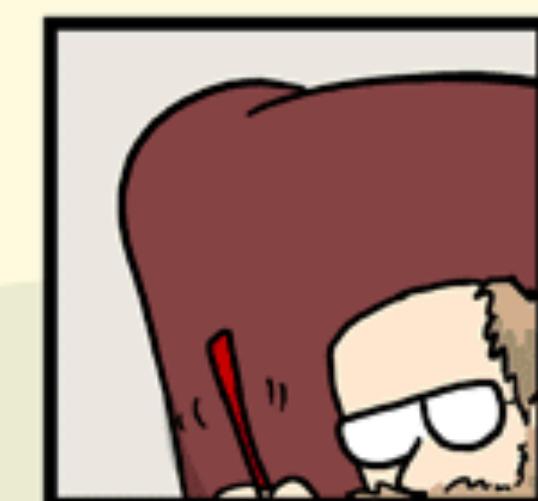
↑  
FINAL\_rev.6.COMMENTS.doc



↑  
FINAL\_rev.8.comments5.CORRECTIONS.doc



↑  
FINAL\_rev.18.comments7.corrections9.MORE.30.doc



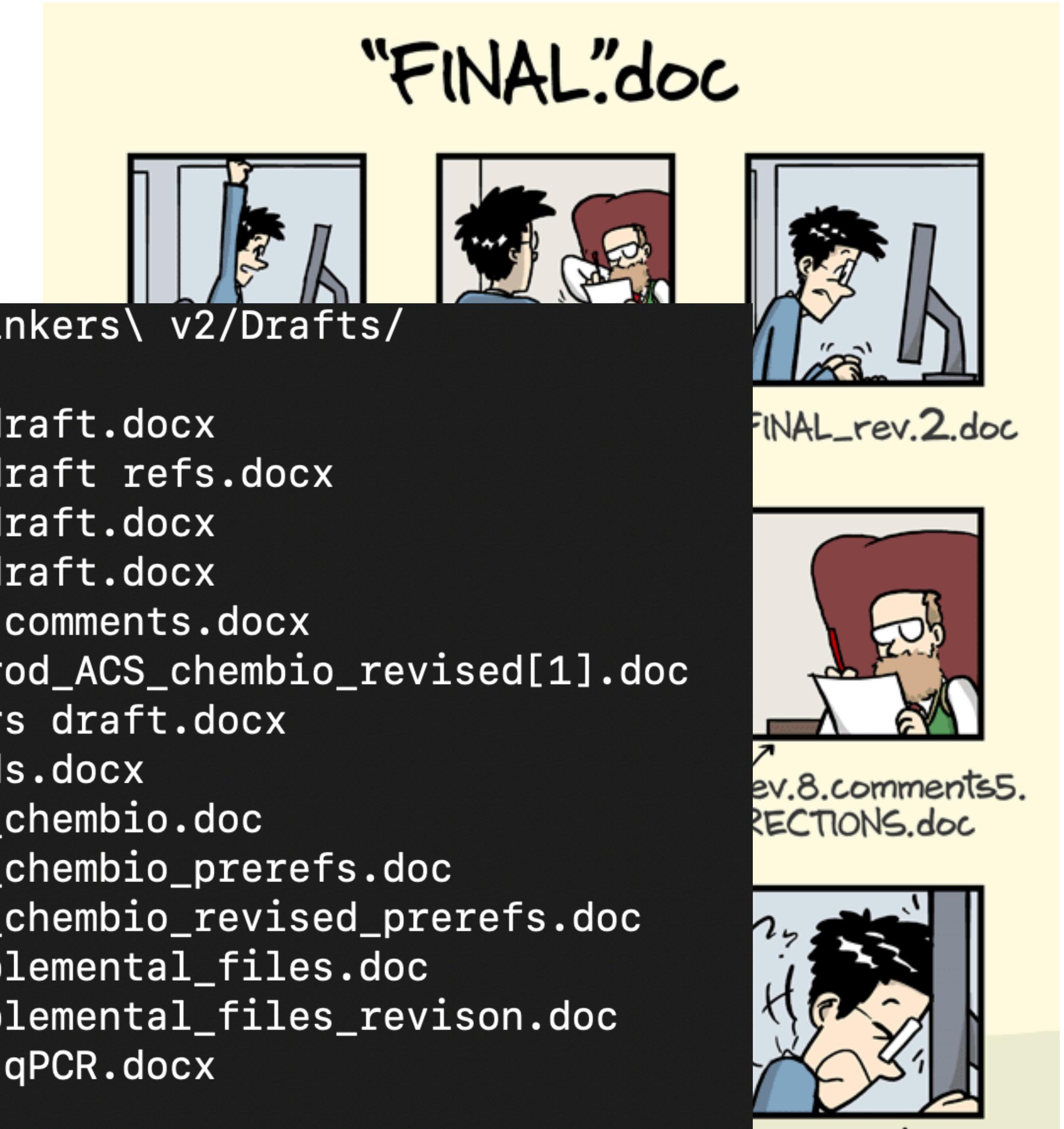
↑  
FINAL\_rev.22.comments49.corrections.10.#@\$%WHYDID  
ICOMETOGRAD SCHOOL????.doc

JORGE CHAM © 2012

# Why use git?

```
[10:12:40 ~] $ cd ~/Documents/Manuscripts/ACS\ chembio/Linkers\ v2/Drafts/  
[10:57:14 Drafts] $ ls  
00 Linkers draft.docx  
00 Supplement.docx  
000_Coonrod_ACS_chembio_revised.doc  
000_Coonrod_ACS_chembio_revised_prrefs.doc  
01 Linker - figures only.docx  
01 Linkers draft refs - MJB.docx  
01 Linkers draft refs.docx  
01 Linkers draft.docx  
01 Supplement.docx  
02 Linkers draft refs CT edits.docx  
02 Linkers draft refs.docx  
02 Linkers draft.docx  
03 Linkers draft refs.docx  
03 Linkers draft.docx  
04 Linkers draft refs.docx
```

```
04 Linkers draft.docx  
05 Linkers draft refs.docx  
05 Linkers draft.docx  
06 Linkers draft.docx  
AB Reviewer comments.docx  
AB_000_Coonrod_ACS_chembio_revised[1].doc  
AB_05 Linkers draft.docx  
Aaron methods.docx  
Coonrod_ACS_chembio.doc  
Coonrod_ACS_chembio_prrefs.doc  
Coonrod_ACS_chembio_revised_prrefs.doc  
Coonrod_Supplemental_files.doc  
Coonrod_Supplemental_files_revision.doc  
Heptamidine qPCR.docx
```

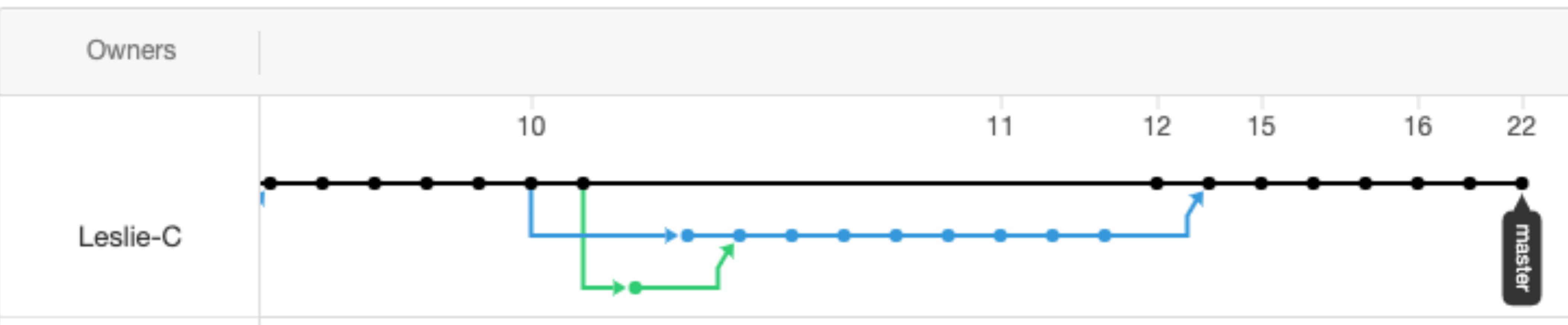


# Why use git?

```
[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_prrefs.doc 06 Linkers draft.docx
```

## Network graph

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



"FINAL".doc



ments5.  
5.doc



ents49.  
WHYDID  
????.doc

# 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

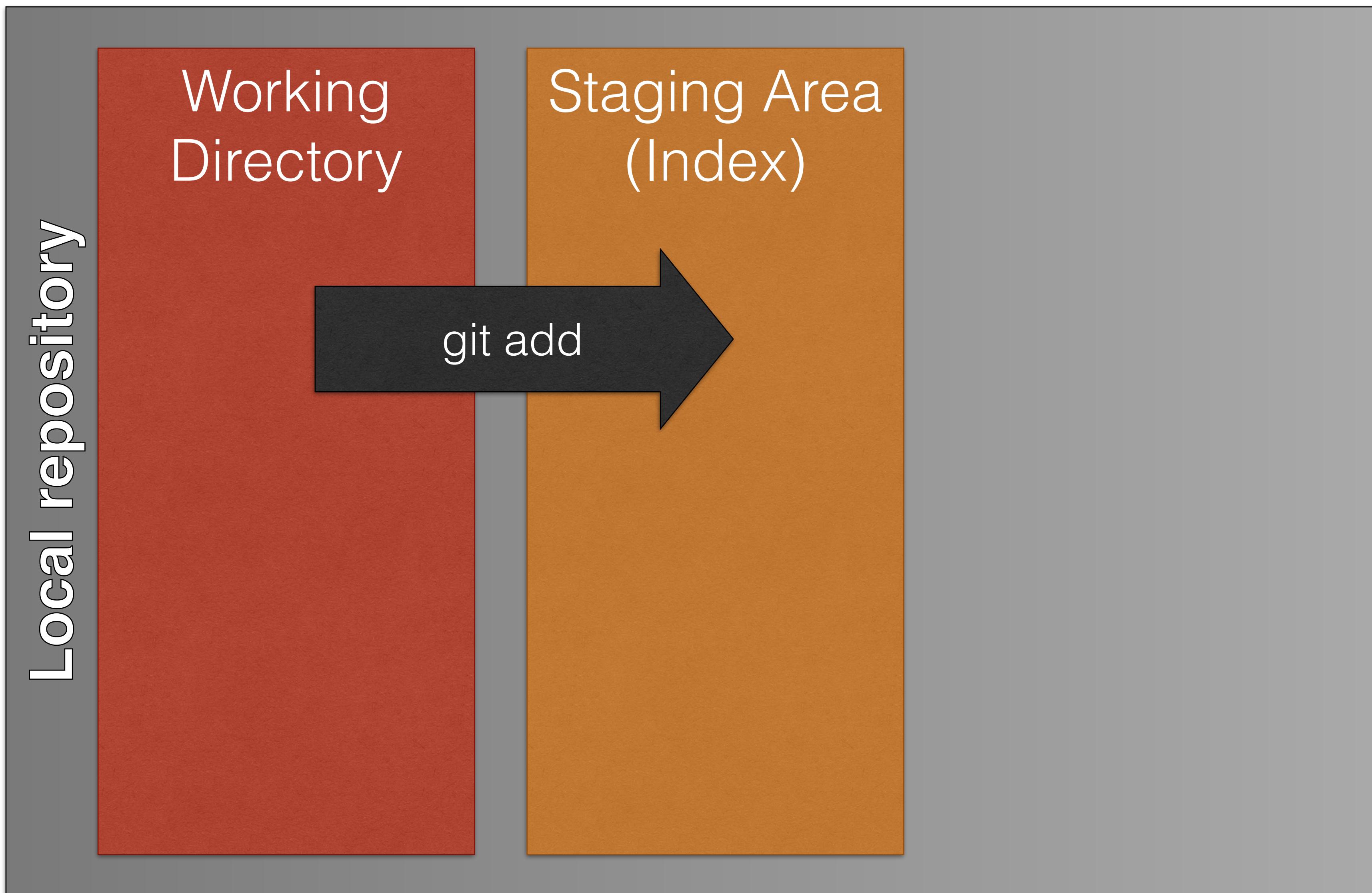
# How does git work? (on a superficial level)

A diagram illustrating a local repository structure. On the left, a vertical dark gray bar contains the text "Local repository" rotated 90 degrees counter-clockwise. To its right is a large, solid orange rectangle labeled "Working Directory" in white text. To the right of the orange rectangle is a large, solid gray rectangular area.

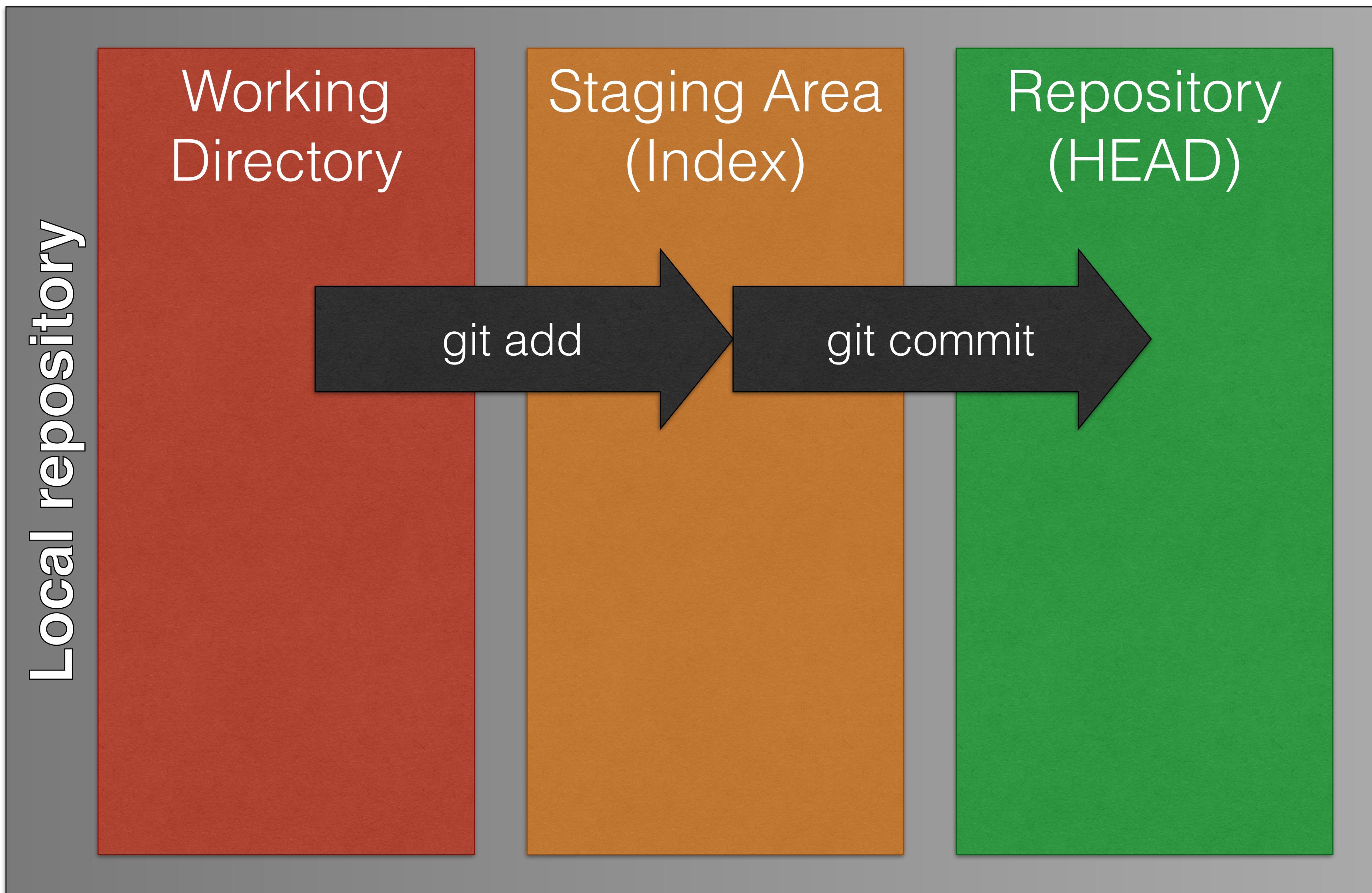
Working  
Directory

Local repository

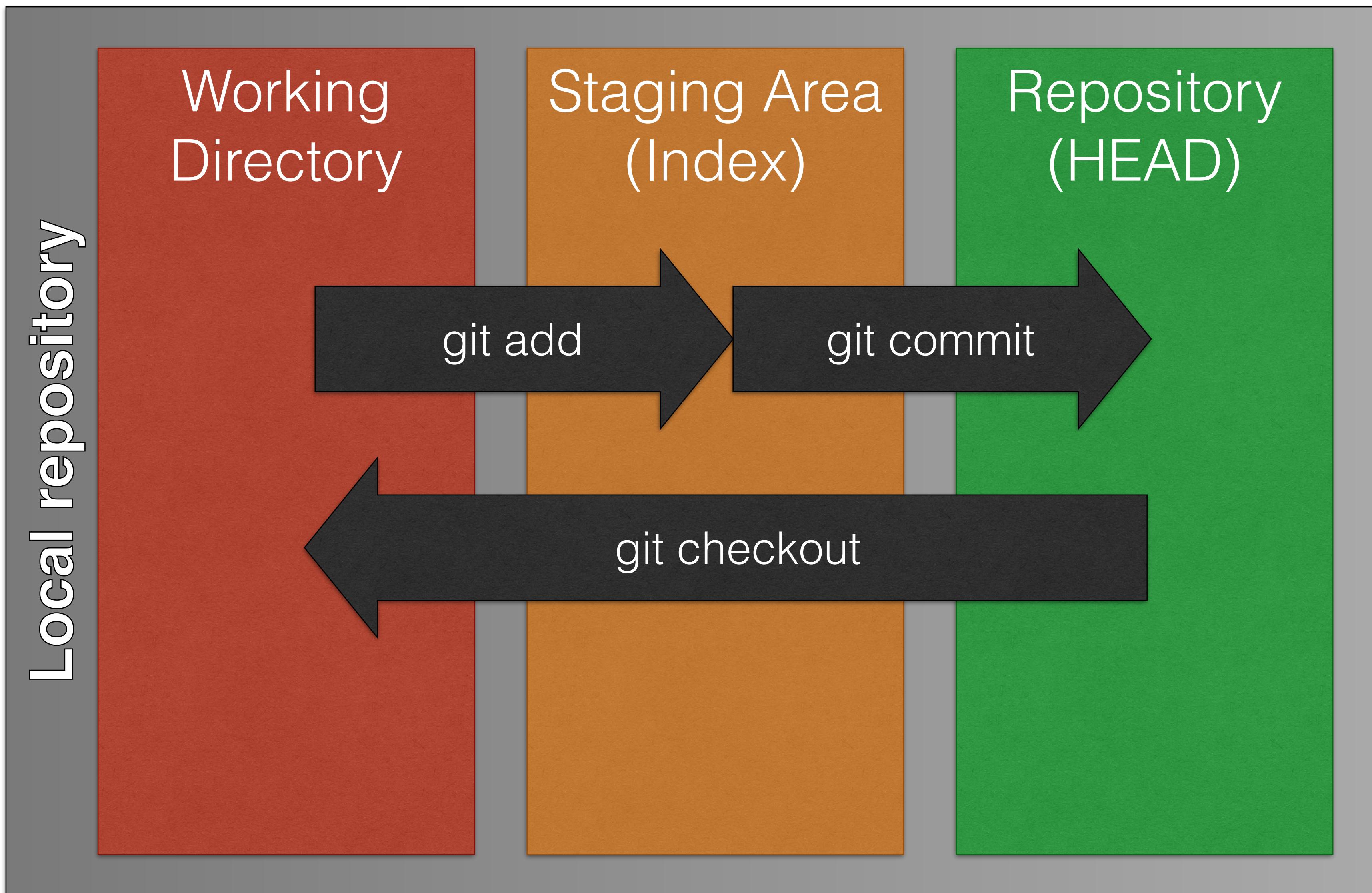
# How does git work? (on a superficial level)



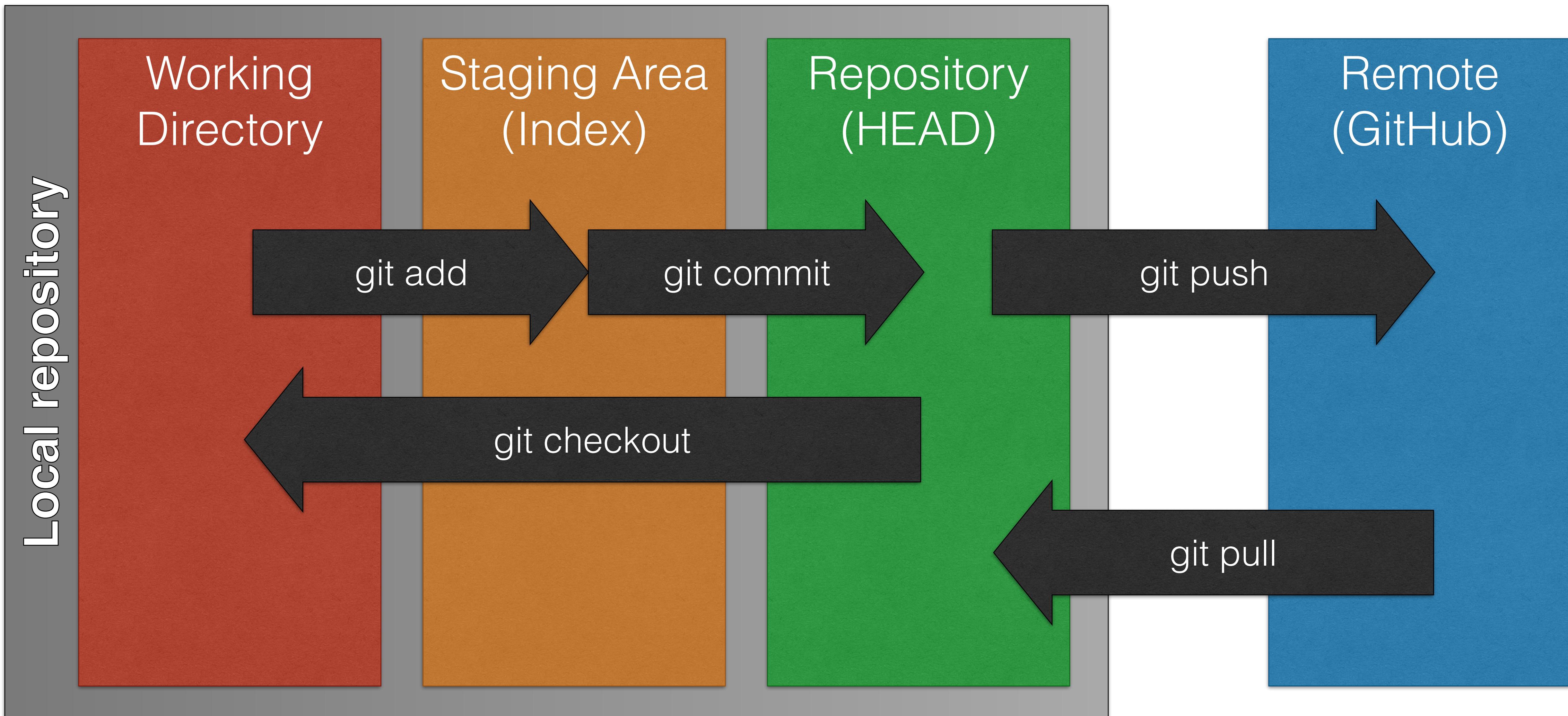
# How does git work? (on a superficial level)

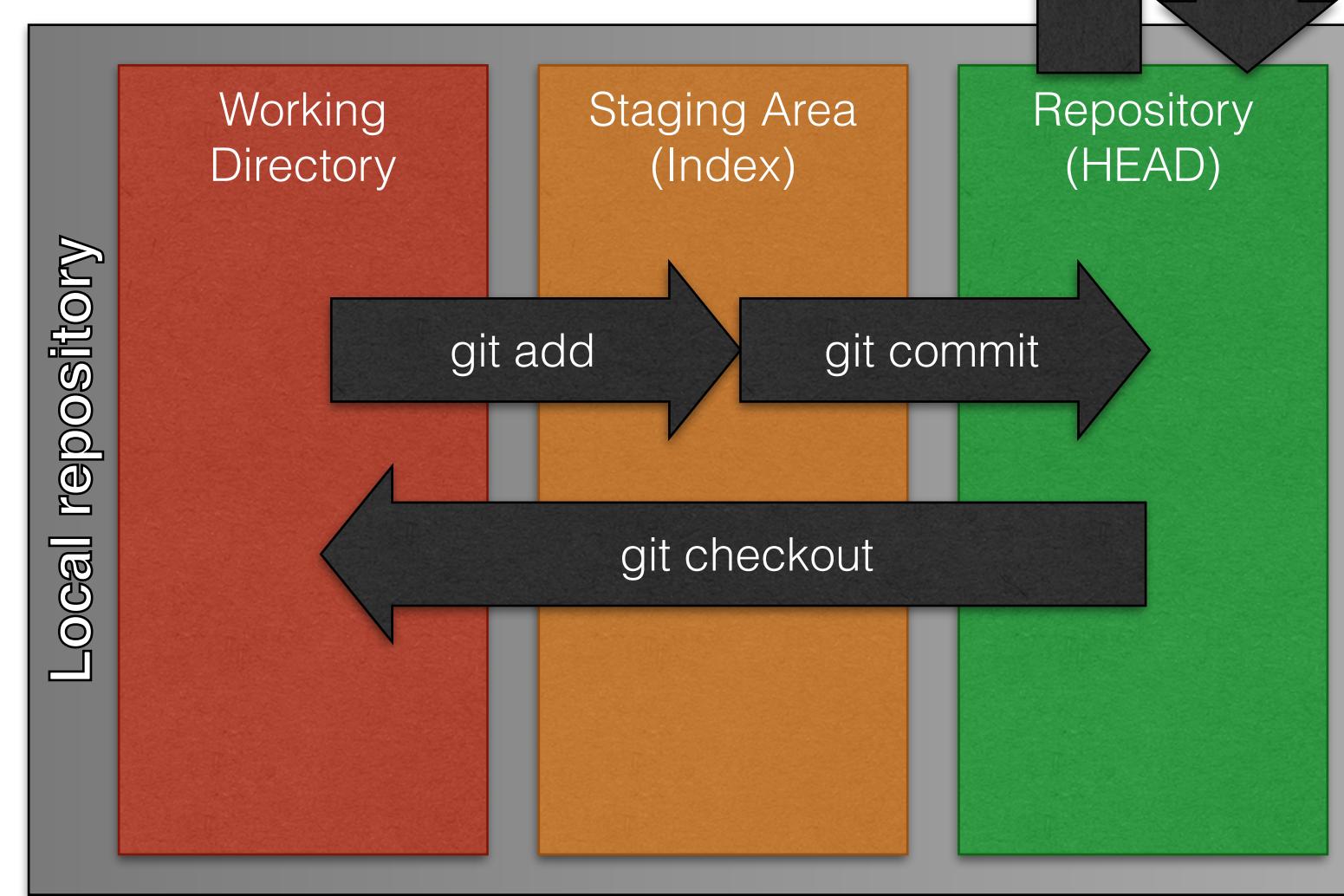
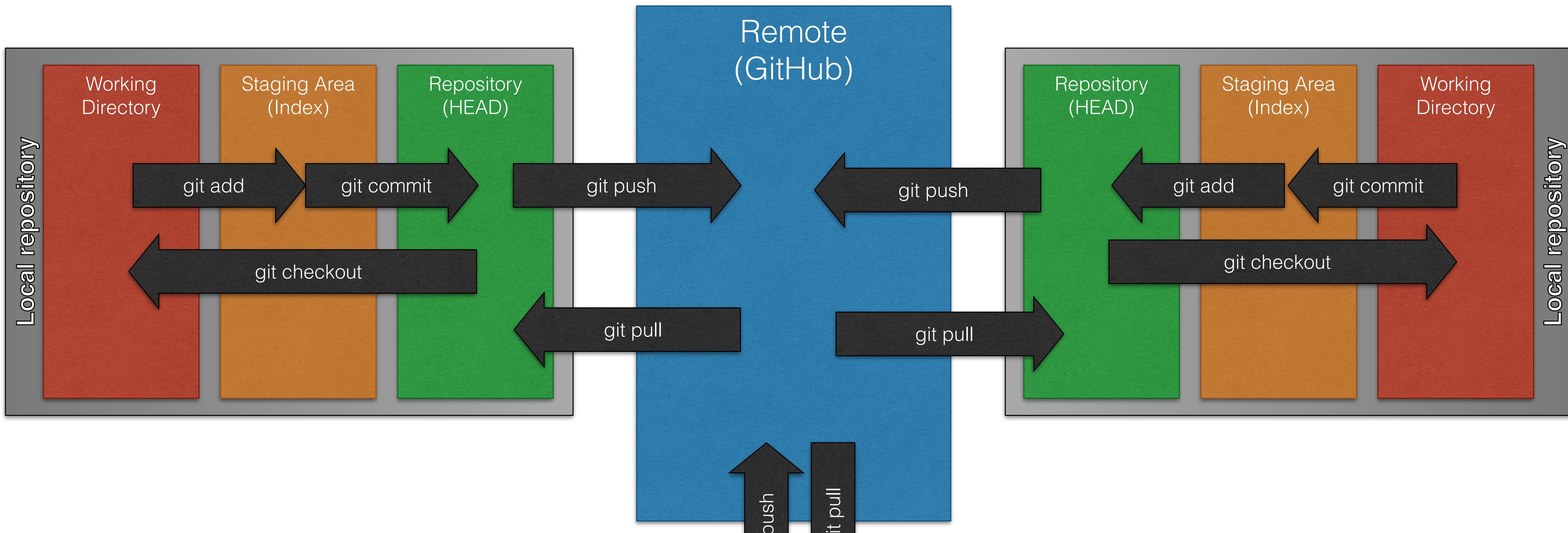


# How does git work? (on a superficial level)



# How does git work? (on a superficial level)

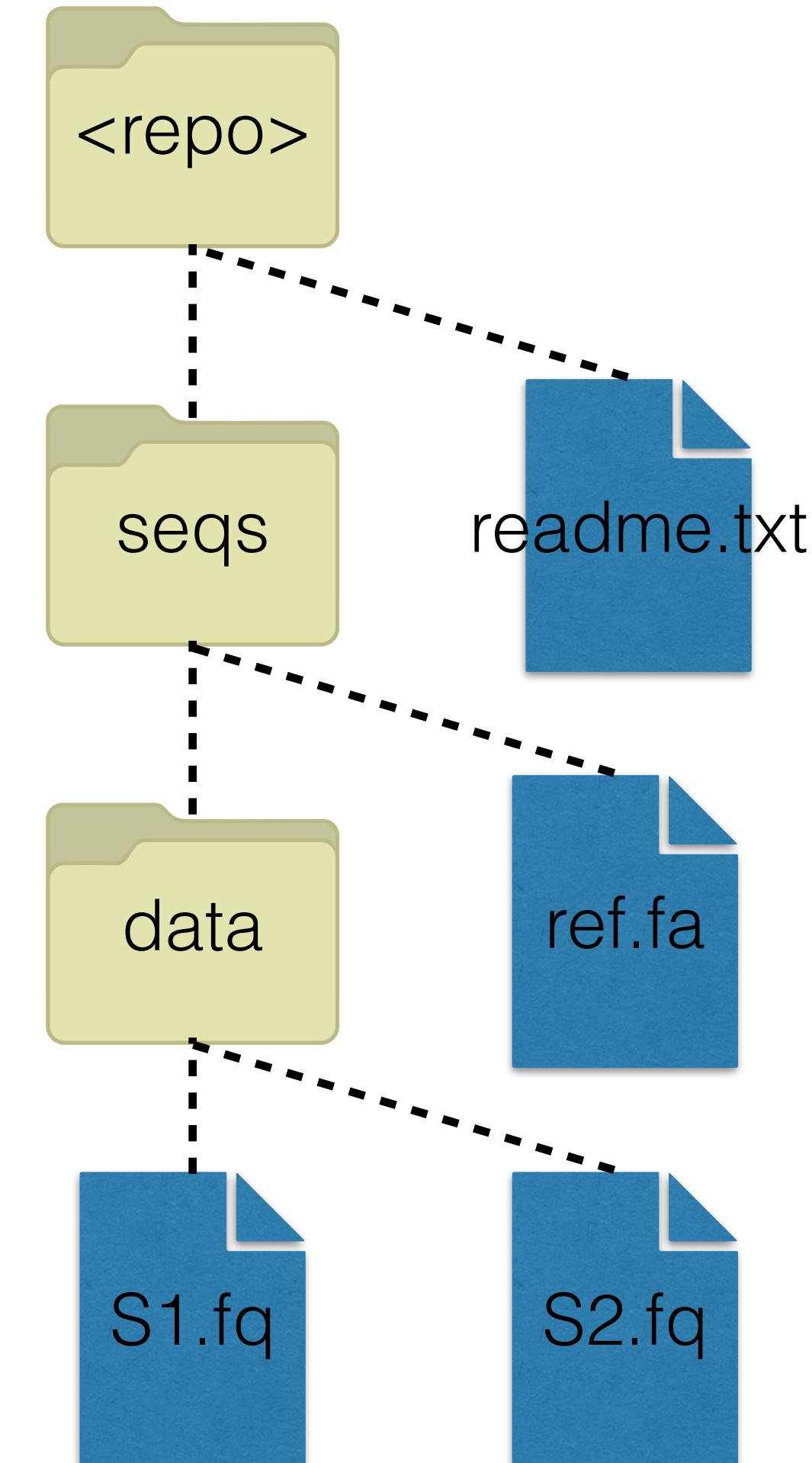




Okay, that's cool and  
all...but how does git  
**REALLY** work?

# 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

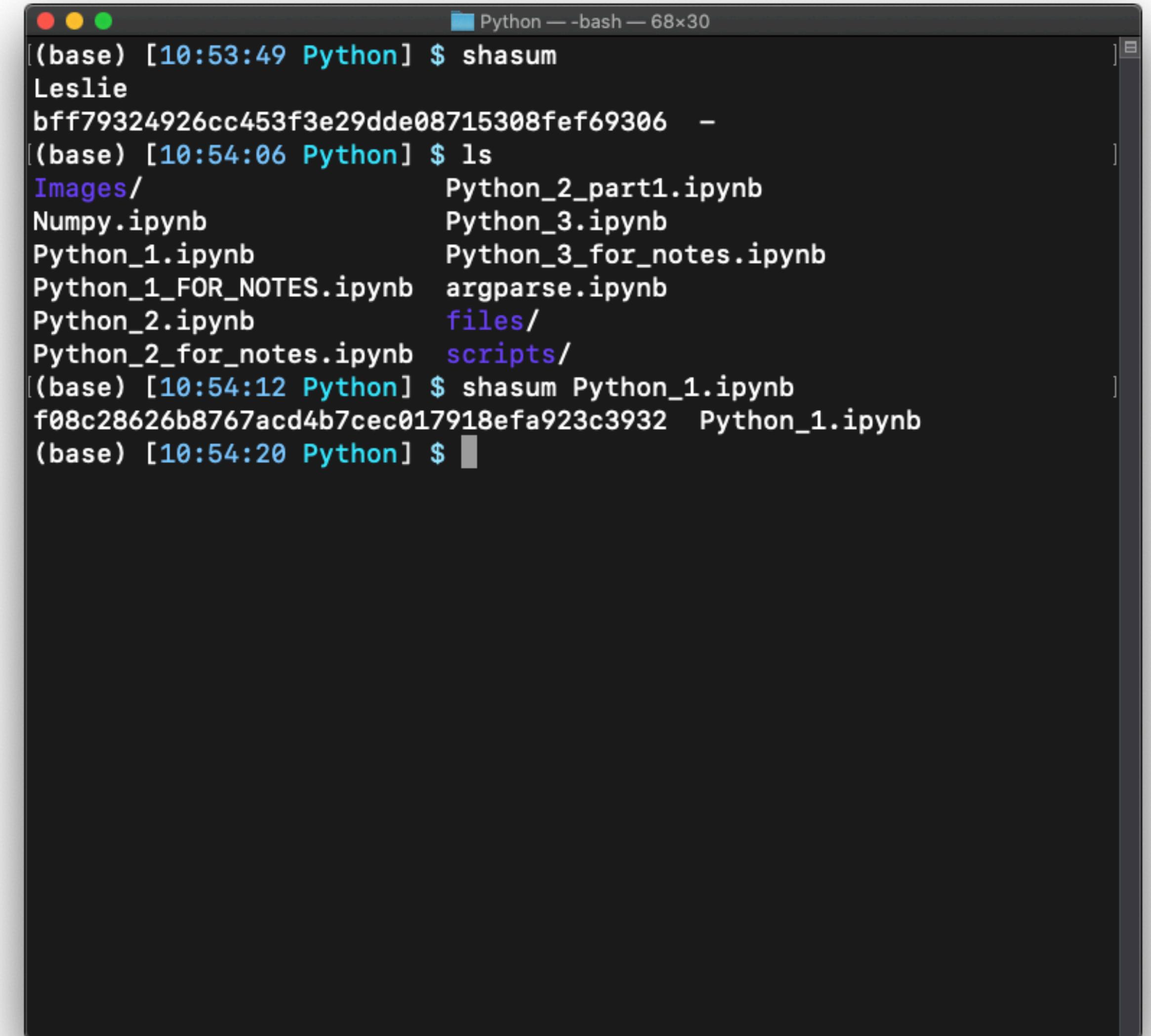
OBJECTS	blob	array<bytes>
	tree	map{string[filename], location[tree or blob]}
	commit	custom_structure{ parents: array<commit> author: string message: string snapshot: tree }
objects stored by map{string[hash(o)], object}		

# 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
- **SHATTERED**



A screenshot of a terminal window titled "Python — bash — 68x30". The window shows a file listing and two SHA-1 checksum commands. The file listing includes "Leslie", "bff79324926cc453f3e29dde08715308fef69306", "Images/", "Python\_2\_part1.ipynb", "Numpy.ipynb", "Python\_3.ipynb", "Python\_1.ipynb", "Python\_3\_for\_notes.ipynb", "Python\_1\_FOR\_NOTES.ipynb", "argparse.ipynb", "Python\_2.ipynb", "files/", "Python\_2\_for\_notes.ipynb", "scripts/", and "Python\_1.ipynb". The first SHA-1 command is "shasum" followed by the file name "Leslie". The second SHA-1 command is "shasum Python\_1.ipynb". The output of the second command shows the original hash "bff79324926cc453f3e29dde08715308fef69306" followed by a space and the new hash "f08c28626b8767acd4b7cec017918efa923c3932" separated by a tab, with the file name "Python\_1.ipynb" at the end.

```
(base) [10:53:49 Python] $ shasum
Leslie
bff79324926cc453f3e29dde08715308fef69306 -
(base) [10:54:06 Python] $ ls
Images/                  Python_2_part1.ipynb
Numpy.ipynb               Python_3.ipynb
Python_1.ipynb             Python_3_for_notes.ipynb
Python_1_FOR_NOTES.ipynb  argparse.ipynb
Python_2.ipynb              files/
Python_2_for_notes.ipynb   scripts/
(base) [10:54:12 Python] $ shasum Python_1.ipynb
f08c28626b8767acd4b7cec017918efa923c3932  Python_1.ipynb
(base) [10:54:20 Python] $
```

# Aside of the aside: SHA-1 is broken



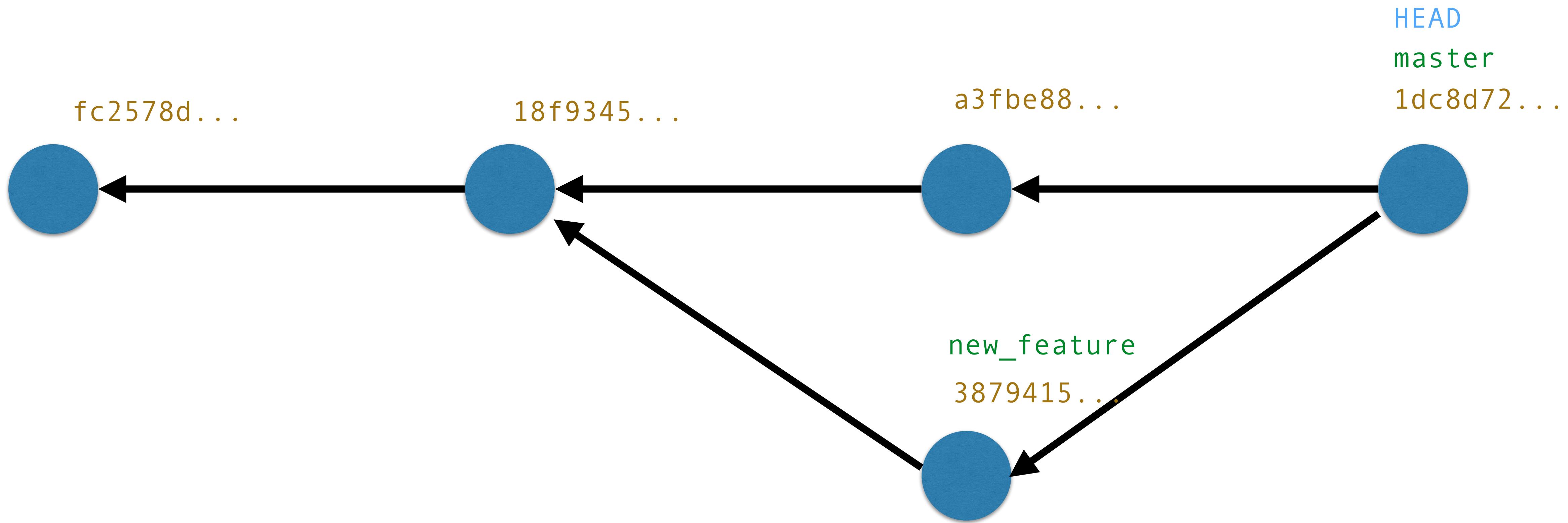
Attack complexity

9,223,372,036,854,775,808

SHA-1 compressions performed

[redacted]  
d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff 2.pdf

# Commit graph



# Let's do it!

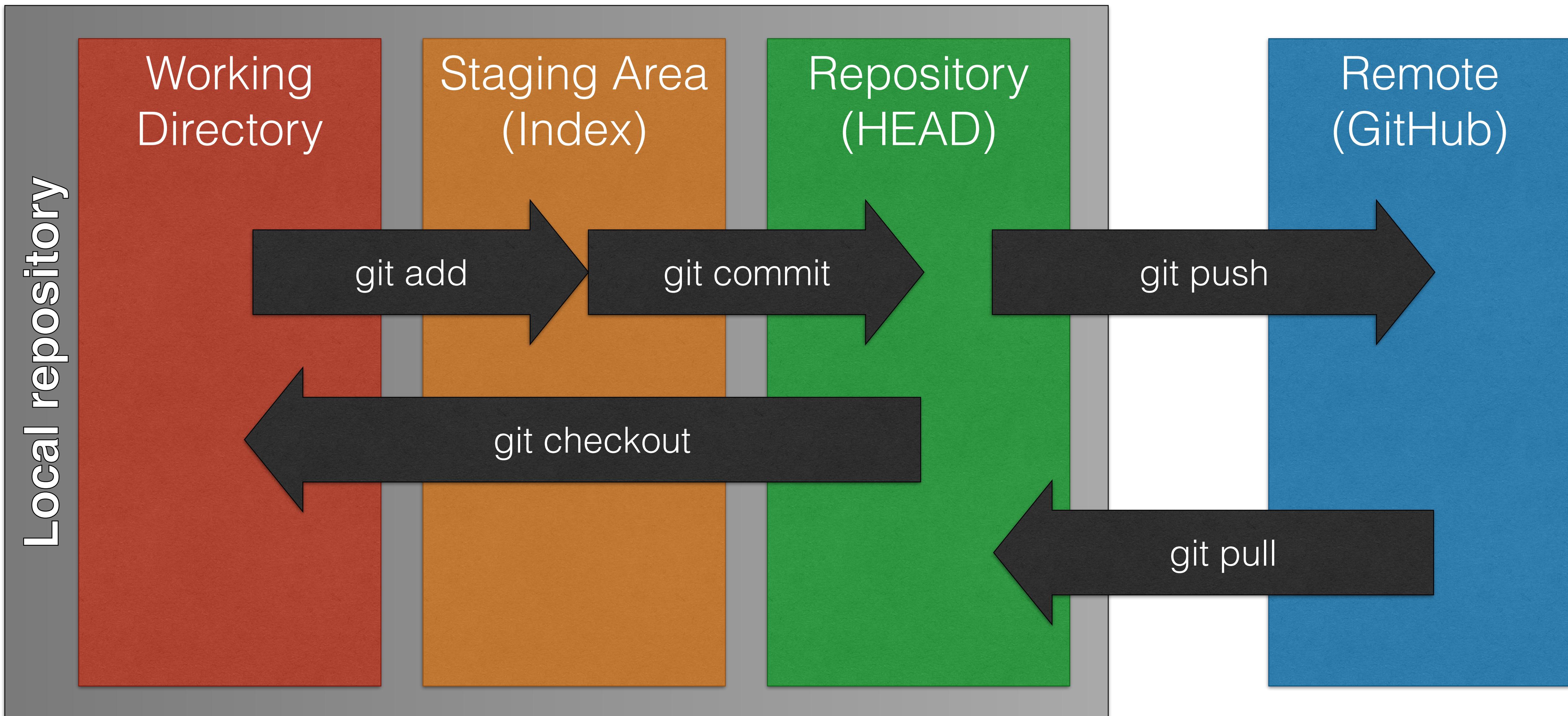
```
#create a repo
$ mkdir git-demo
$ cd git-demo
$ git init
Initialized empty Git repository in ~/git-demo/.git/

#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
$ git 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`