Introduction to Git and GitHub

Kendra Oudyk (she/her)

2021-07-26

Many parts of this presentation are inspired / based on these great resources

- Chacon, S., & Straub, B. (2014). *Pro git.* Springer Nature. Available at https://git-scm.com/book/en/v2
- The Carpentries. (2021). *Version Control with Git.* https://swcarpentry.github.io/git-novice/.

Goals

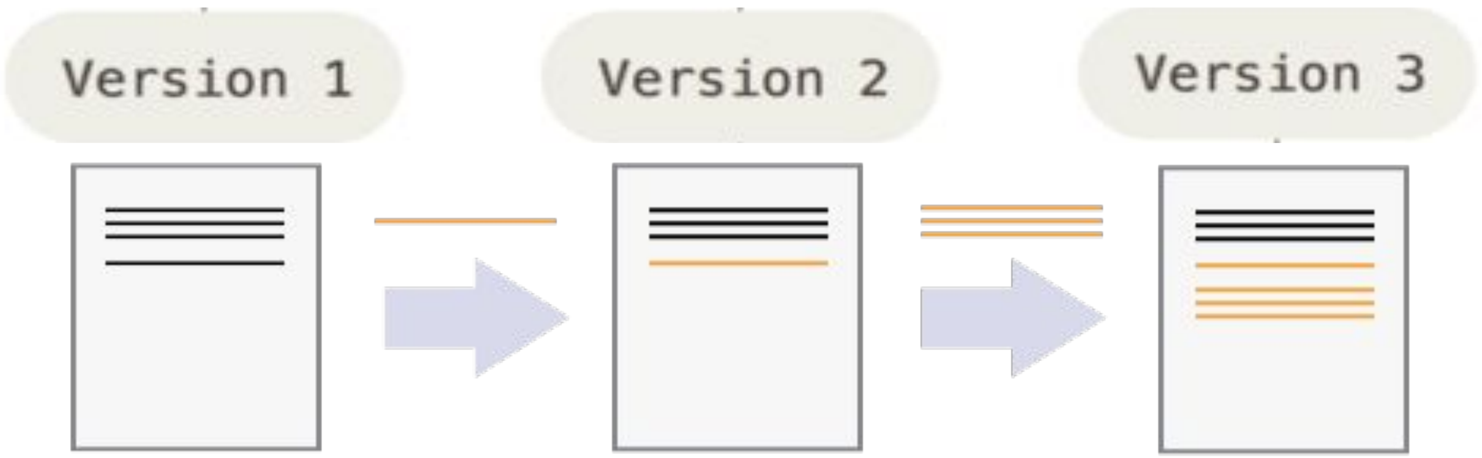
- What is distributed version control?
- Why is Git useful?
- Track your own work with Git; and
- Share your work and collaborate on GitHub.

Goals

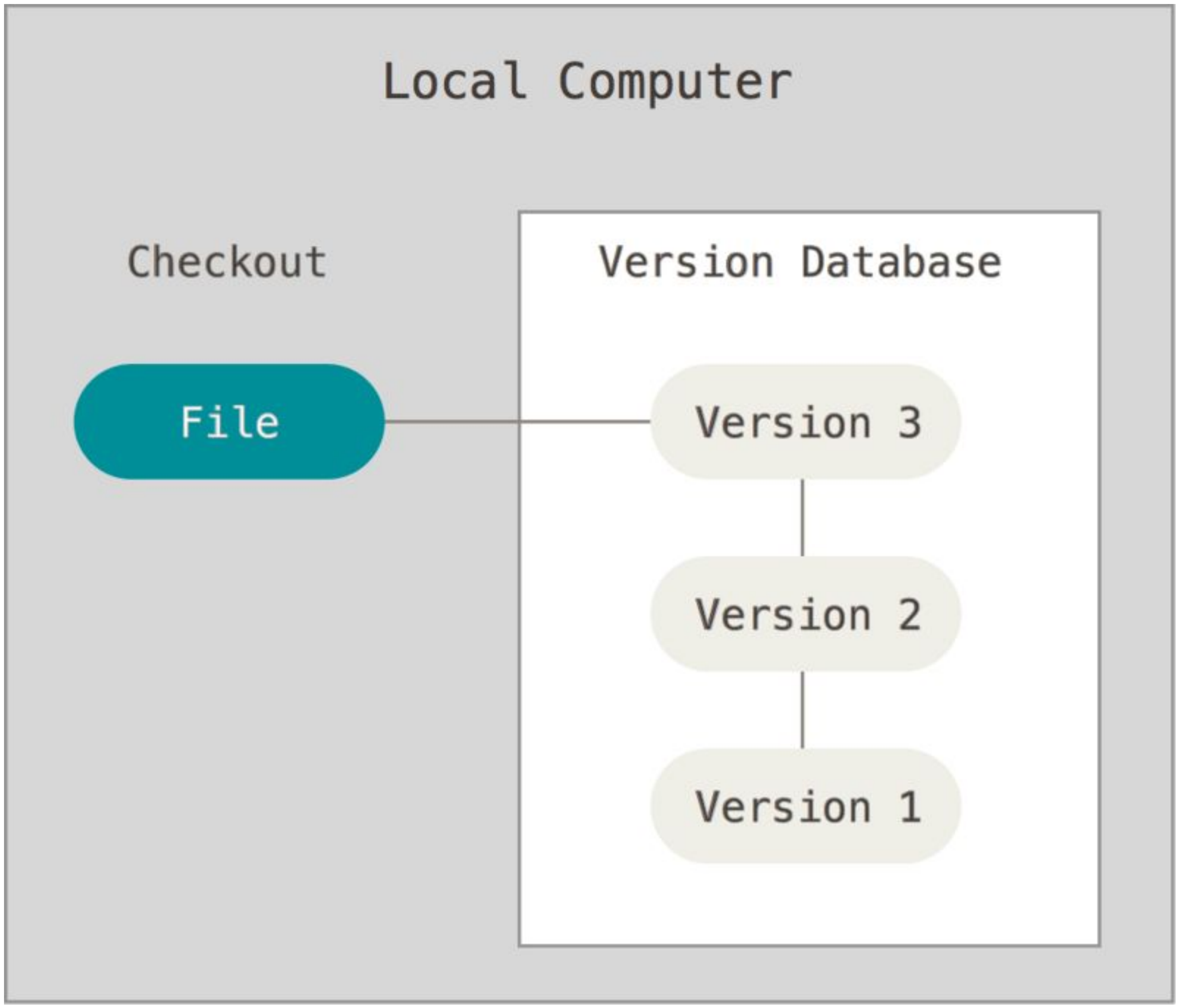
- **What is distributed version control?**
- Why is Git useful?
- Track your own work with Git; and
- Share your work and collaborate on GitHub.

Version control

- Tracks **changes** to files
- Lets you recall different **versions** of files
- Becomes more essential as **collaborative projects** grow
- Can track almost **any type** of file (works best on text-based files)
- There are different **types** of version control

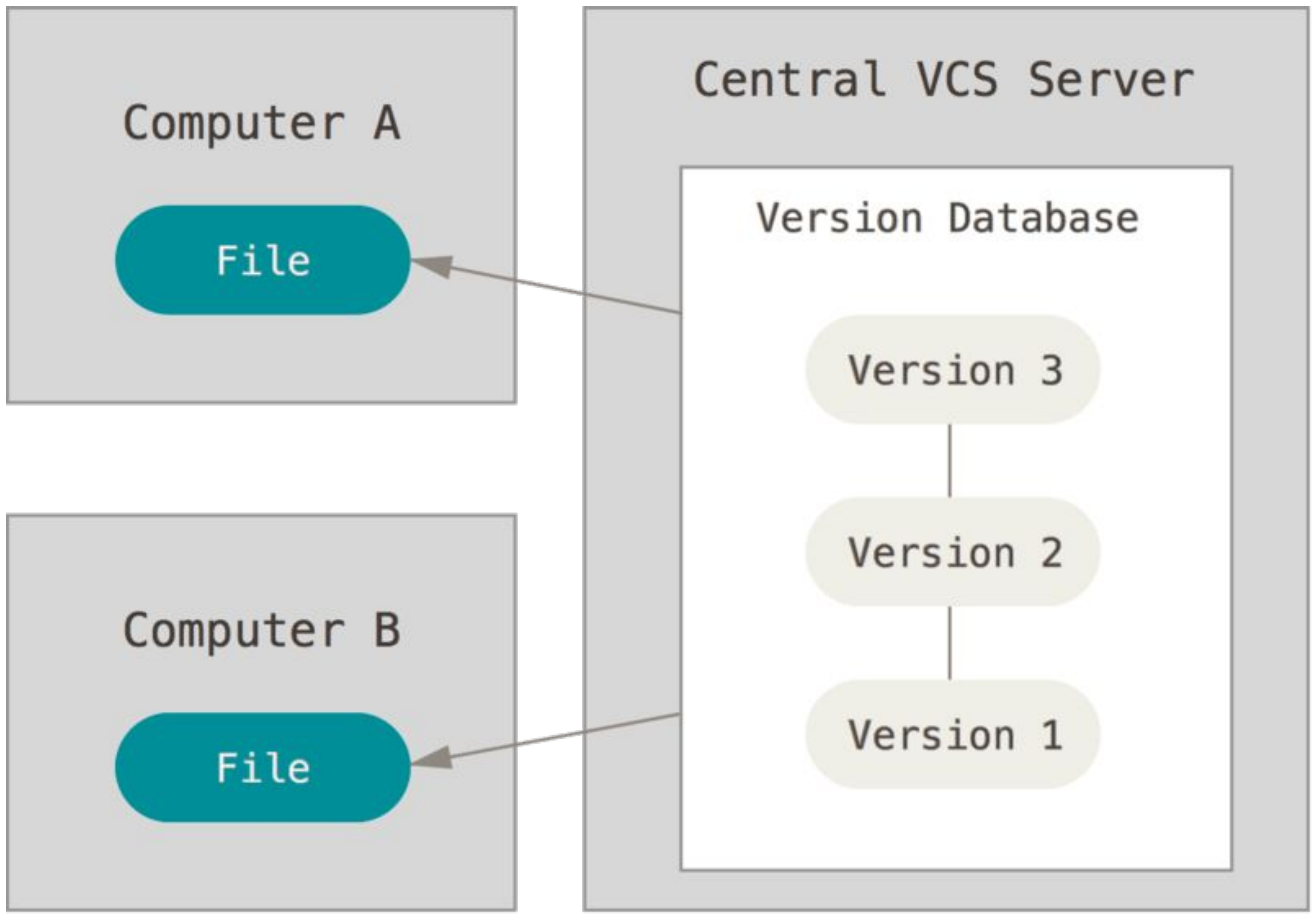


Local version control



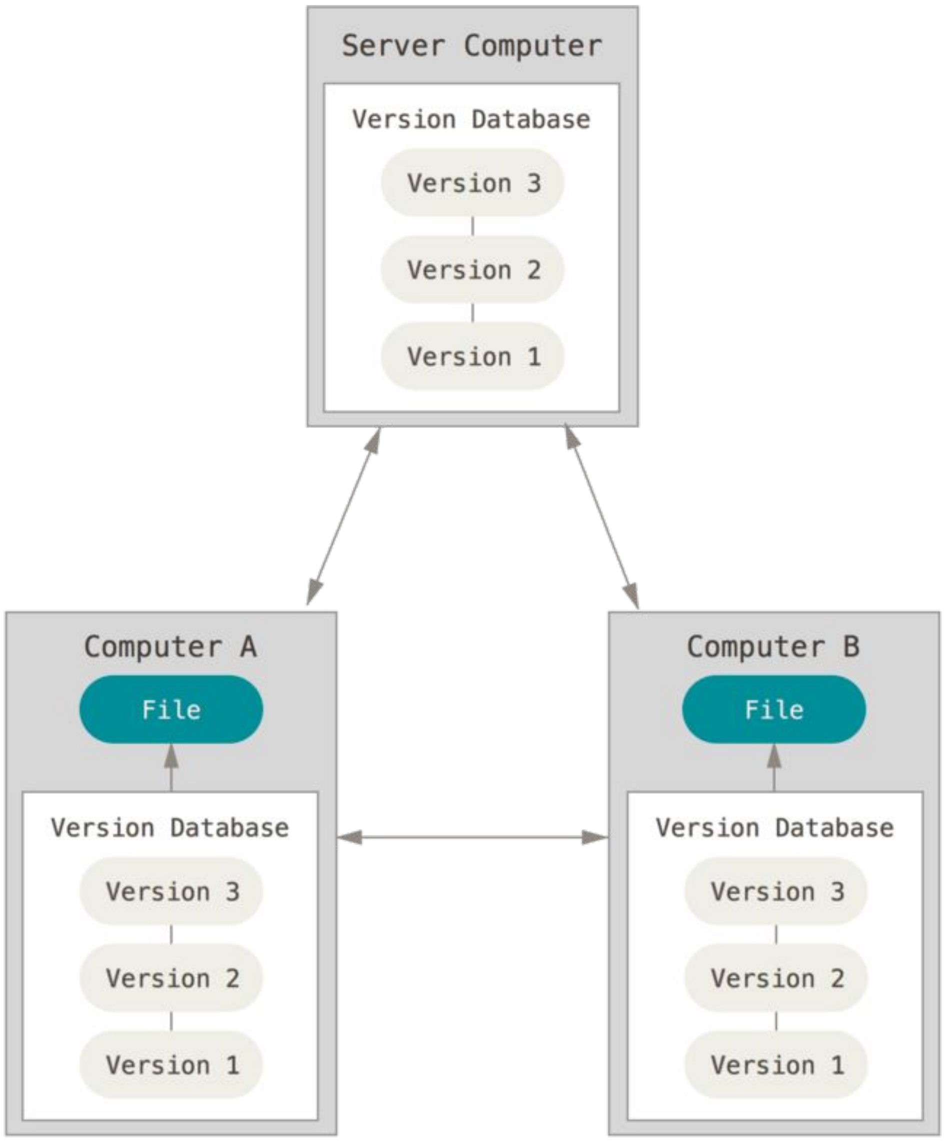
But how do we collaborate?

Centralized version control



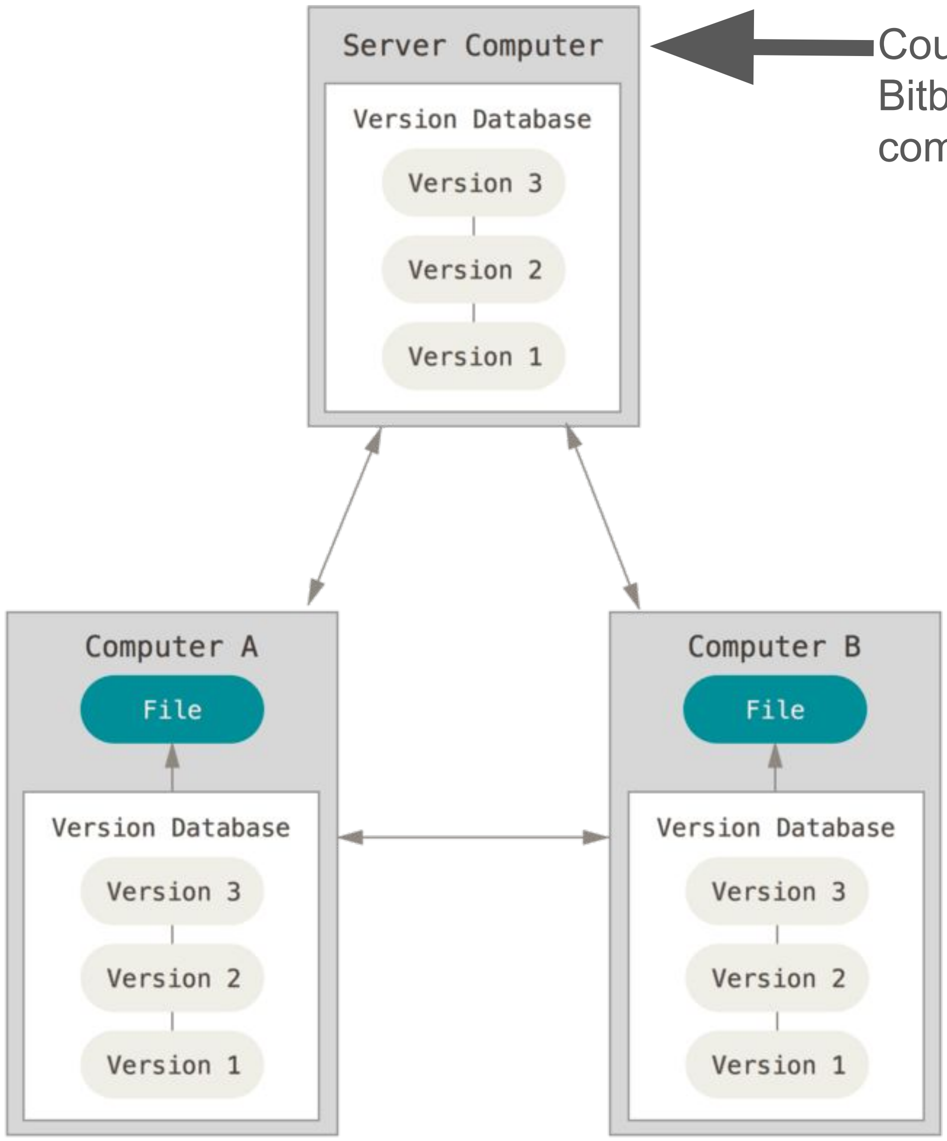
But what if the server crashes?

Distributed version control



Git vs. GitHub

- Git is the "**language**" we use to do version control.
- GitHub **hosts** git repositories online.



Could be GitHub, Bitbucket, Gitlab, a server computer...

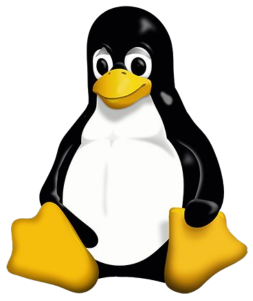
Goals

- **What is distributed version control?**
- Why is Git useful?
- Track your own work with Git; and
- Share your work and collaborate on GitHub.

Goals

- What is distributed version control?
- **Why is Git useful?**
- Track your own work with Git; and
- Share your work and collaborate on GitHub.

A quick history lesson



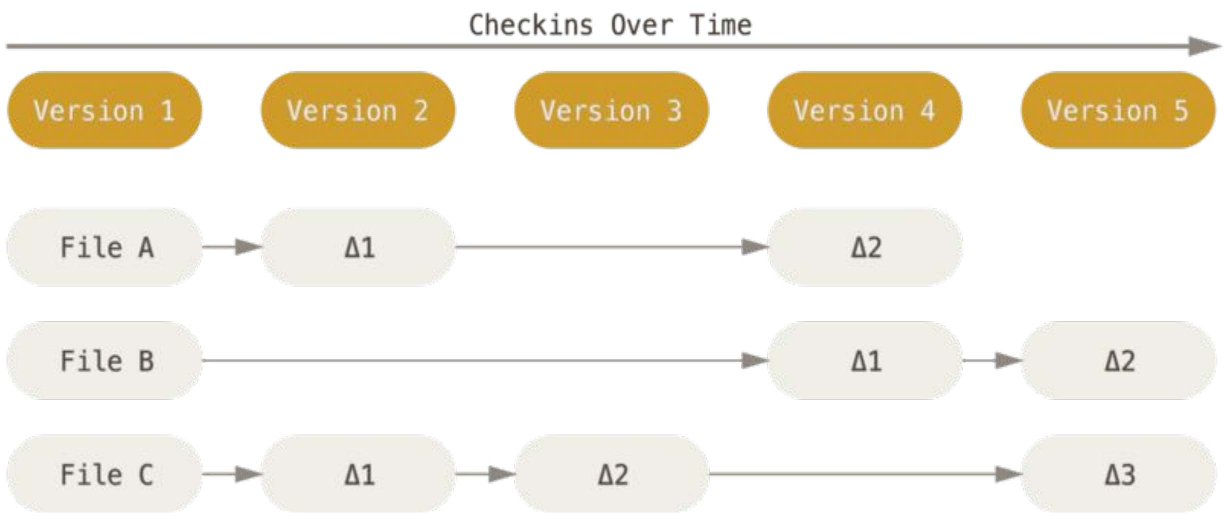
Linux kernel - a huge open source software project

- 1991-2002 - tracked changes with archive files and patches
- 2002-2005 - used BitKeeper, a proprietary DVCS
- 2005 - poor relationship → BitKeeper was going to start charging $$
- 2005 - Linus Torvalds created a new system.

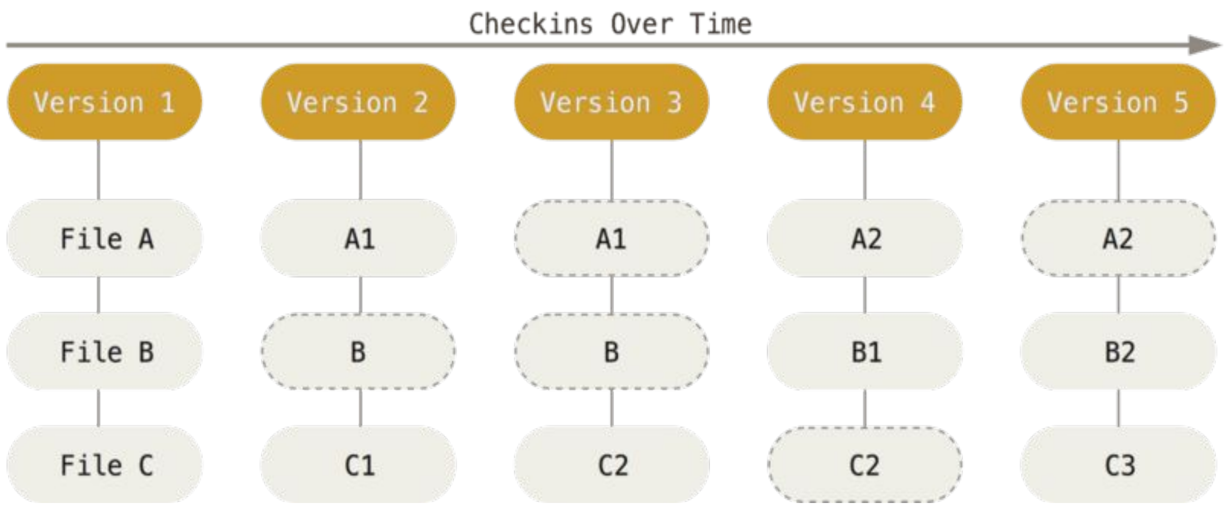
TL;DR git was created for a HUGE software project, so it's very efficient, but you won't need all its features

A different user experience

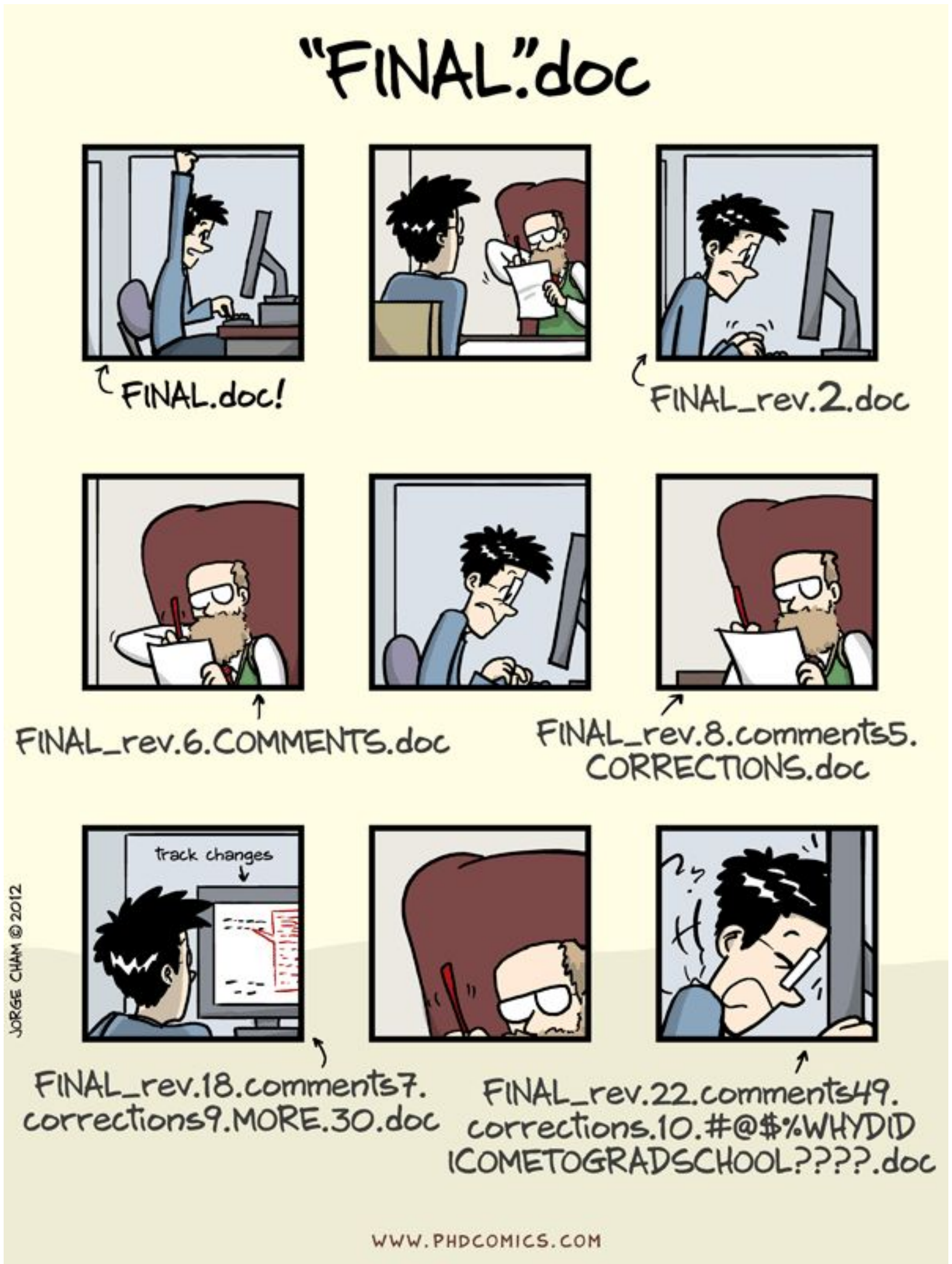
“Delta-based version control” (other version control systems)



Stream of snapshots (git)



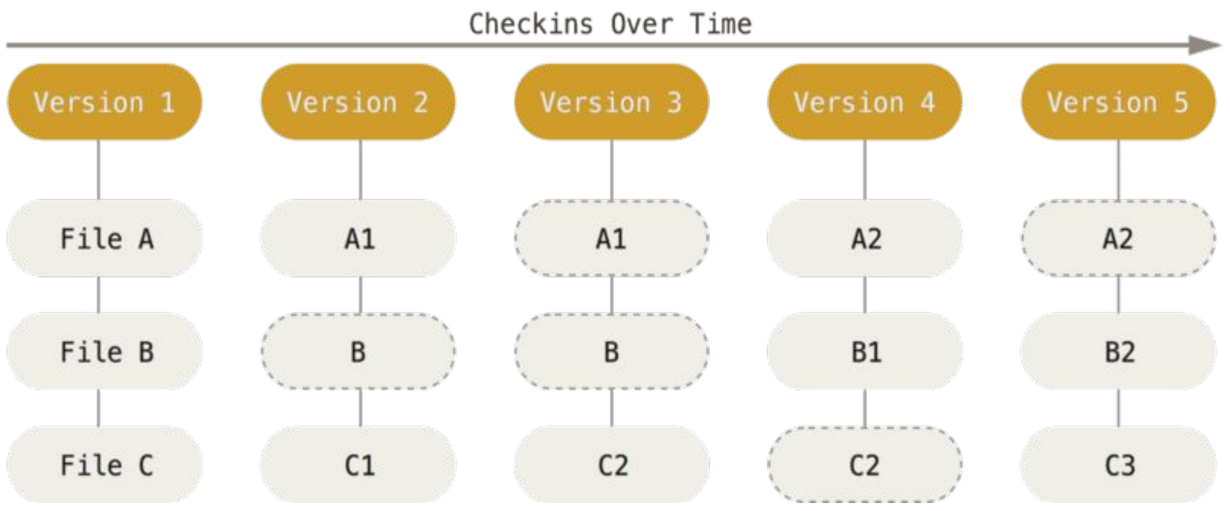
A more intuitive user experience



Git is reliable

- It doesn’t rely on filenames to keep track of files

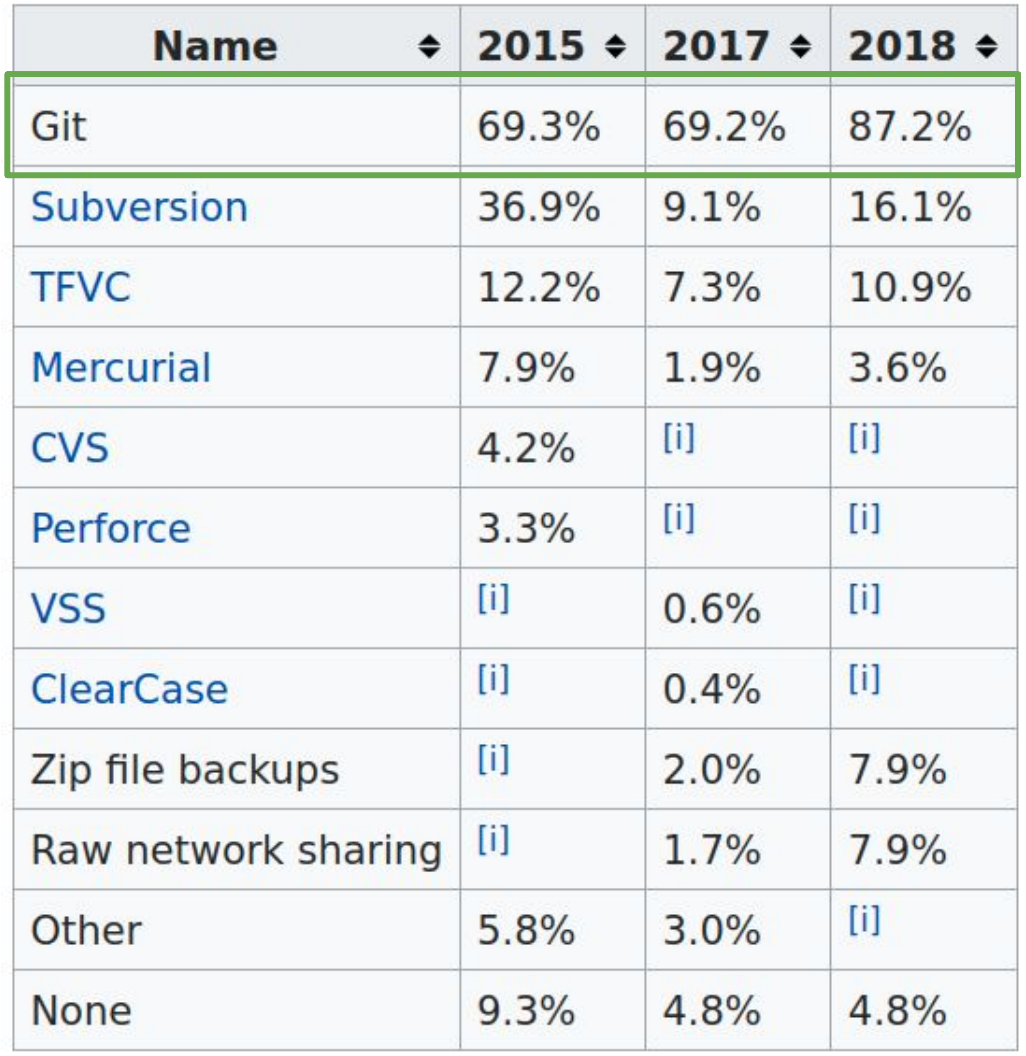
- It converts the contents of a file/directory → hash
    - E.g., `24b9da6552252987aa493b52f8696cd6d3b00373`
    - It is unique and deterministic (1-way)
    - Changed file content → changed hash



Most developers use git for version control

Stack Overflow developer survey

- 2015 (16,694 responses)
- 2017 (30,730 responses)
- 2018 (74,298 responses)

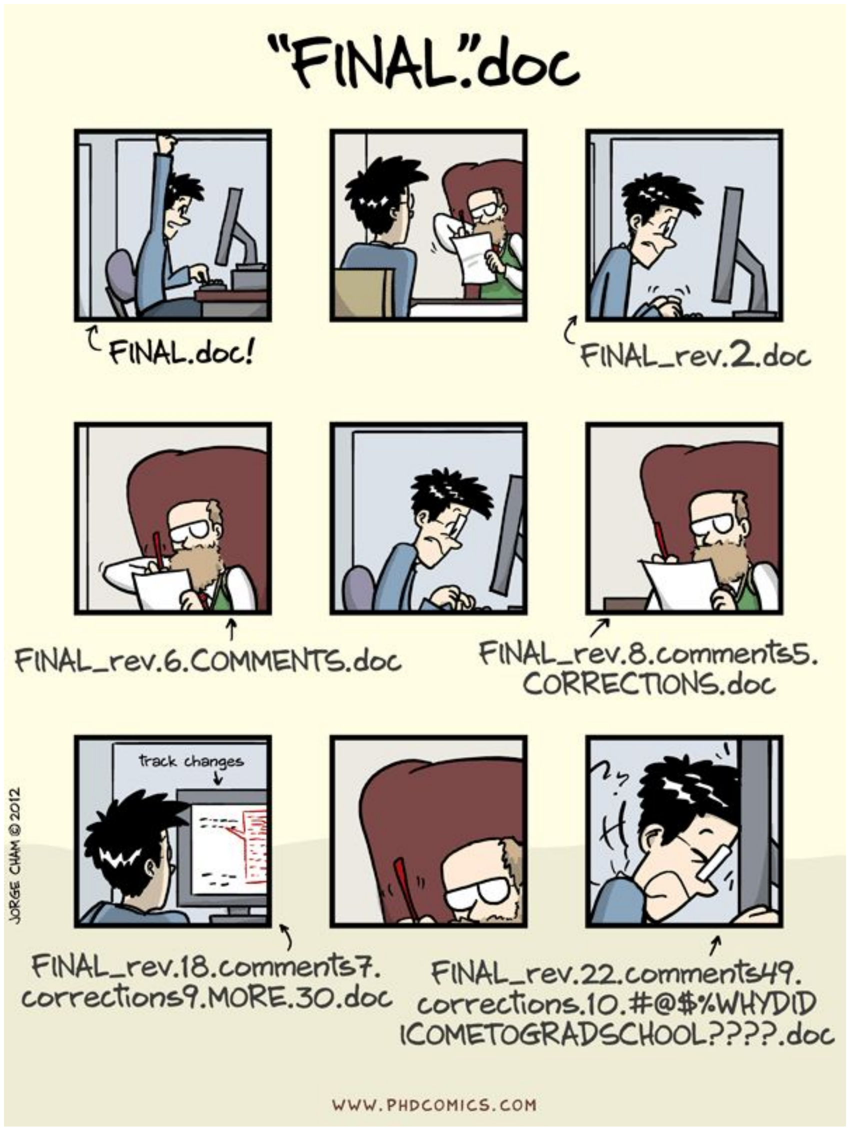


Name	2015	2017	2018
Git	69.3%	69.2%	87.2%
Subversion	36.9%	9.1%	16.1%
TFVC	12.2%	7.3%	10.9%
Mercurial	7.9%	1.9%	3.6%
CVS	4.2%	[i]	[i]
Perforce	3.3%	[i]	[i]
VSS	[i]	0.6%	[i]
ClearCase	[i]	0.4%	[i]
Zip file backups	[i]	2.0%	7.9%
Raw network sharing	[i]	1.7%	7.9%
Other	5.8%	3.0%	[i]
None	9.3%	4.8%	4.8%

But we’re not developers

But we’re not developers… right?

- We write code or use GUIs for research
- We (hopefully) want to do our research in a way that is open, reproducible, and collaborative
- Do you have your own system that does this?
  (I certainly don’t, not for lack of trying)


"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

“Science, after all, aspires to be distributed, open-source knowledge development.”

Science as Amateur Software Development

McElreath, R. (2020, September 26). *Science as amateur software development* [video]. YouTube. https://www.youtube.com/watch?v=zwRdO9_GGhY

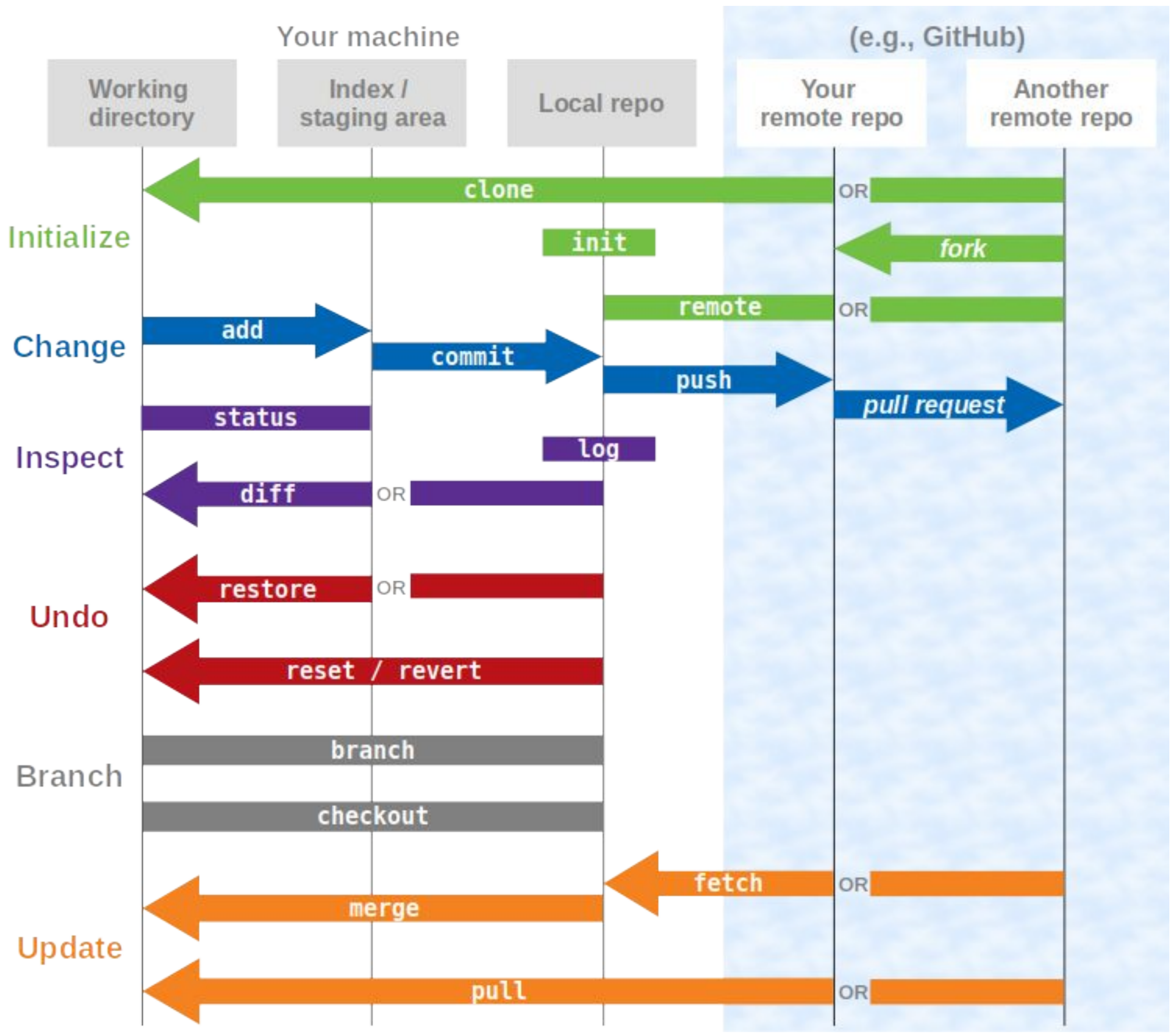
Goals

- What is distributed version control?
- **Why is Git useful?**
- Track your own work with Git; and
- Share your work and collaborate on GitHub.

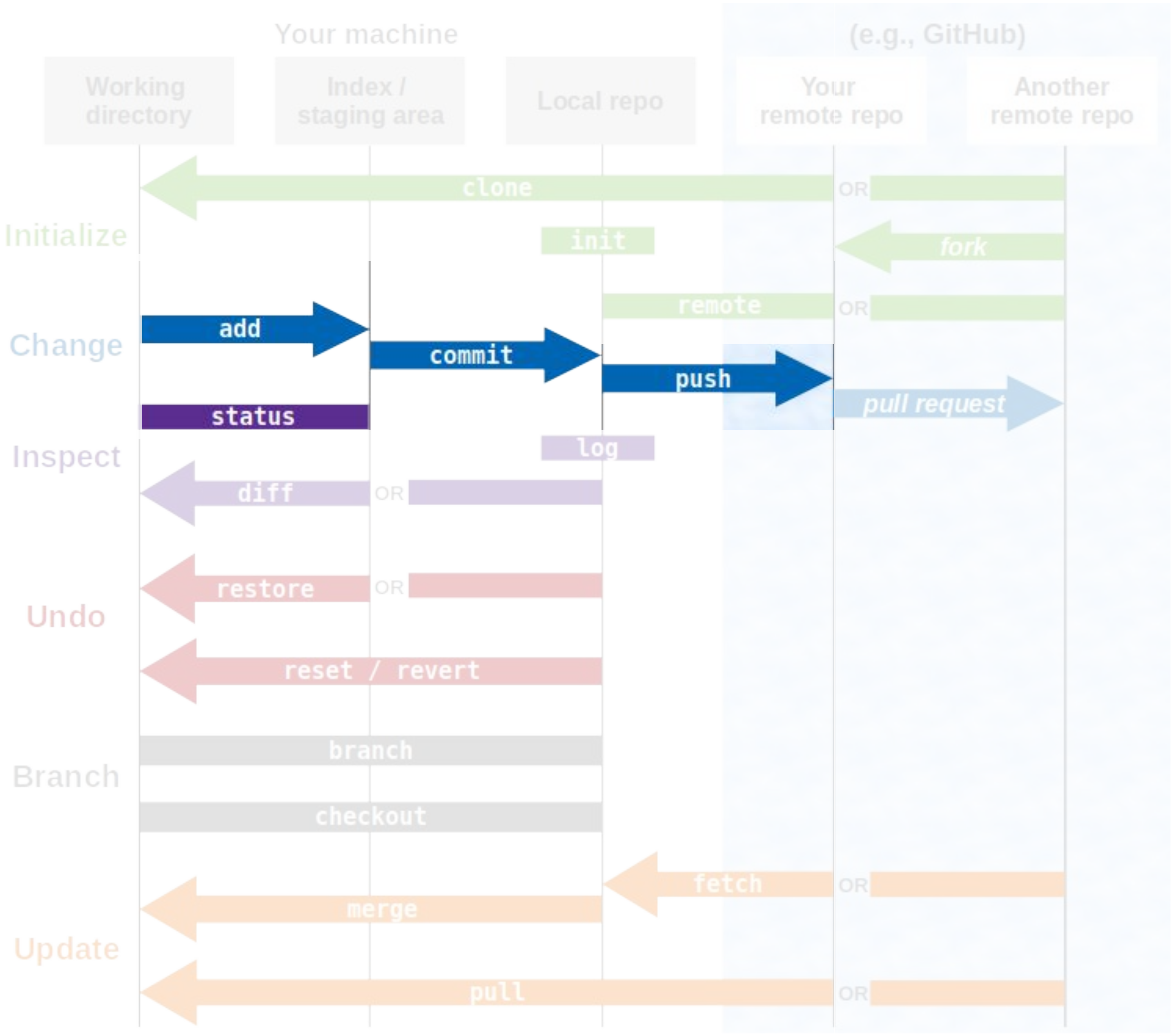
Goals

- What is distributed version control?
- Why is Git useful?
- **Track your own work with Git; and**
- Share your work and collaborate on GitHub.

The commands we'll cover



You only need a few to start

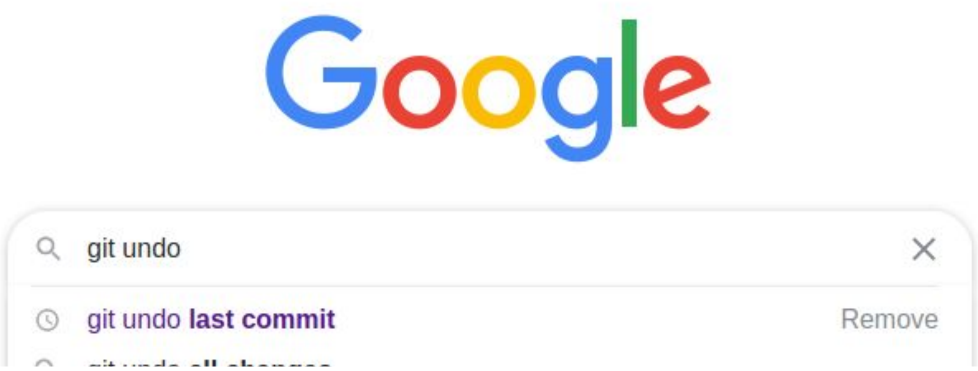


(Everything else you can look up when you need it)

First, how to get help

- In the terminal
  (if you know the verb and want to know what it does or what are its options)
  `git help <verb>`
  `git <verb> --help or git <verb> -h`
  `man git-<verb>`
  TIP: press ‘q’ to exit the manual in the terminal

- Look it up



3 file states

1. **Modified**
   You made a change to the file

2. **Staged**
   You indicated that you want the modified file in your next snapshot

3. **Committed**
   You took the snapshot

3 parts of a Git project

1. **Working directory**
   The version of the project that you're working on

2. **Staging area / Index**
   What will be in your next snapshot

3. **Local repository (i.e., `.git/` folder)**
   Metadata and objects that make up the snapshots

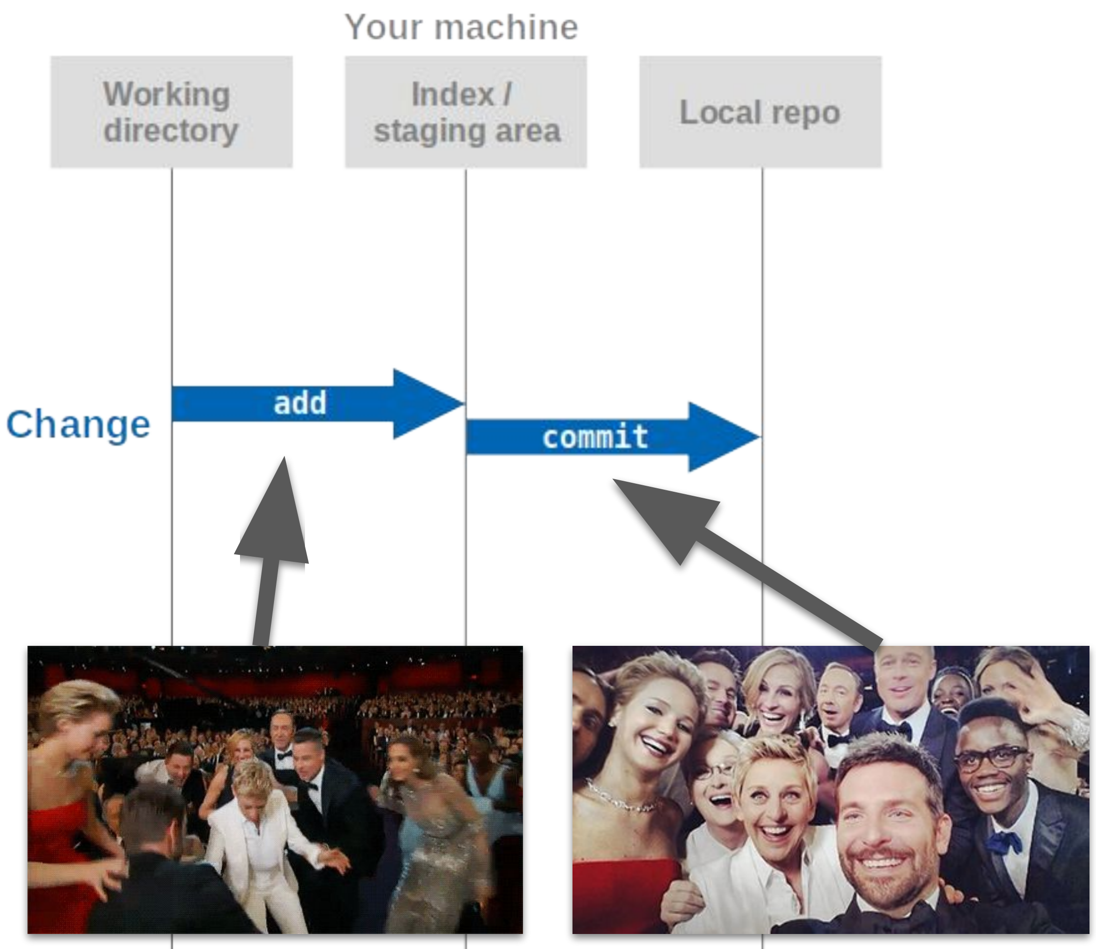
Your machine
Working directory
Index / staging area
Local repo

3-step basic workflow

1. **Modify**
   Change a file in your working tree

2. **Stage**
   ```
   git add <filename>
   ```

3. **Commit**
   ```
   git commit -m “<short, informative commit message>”
   ```



Starting a git repo

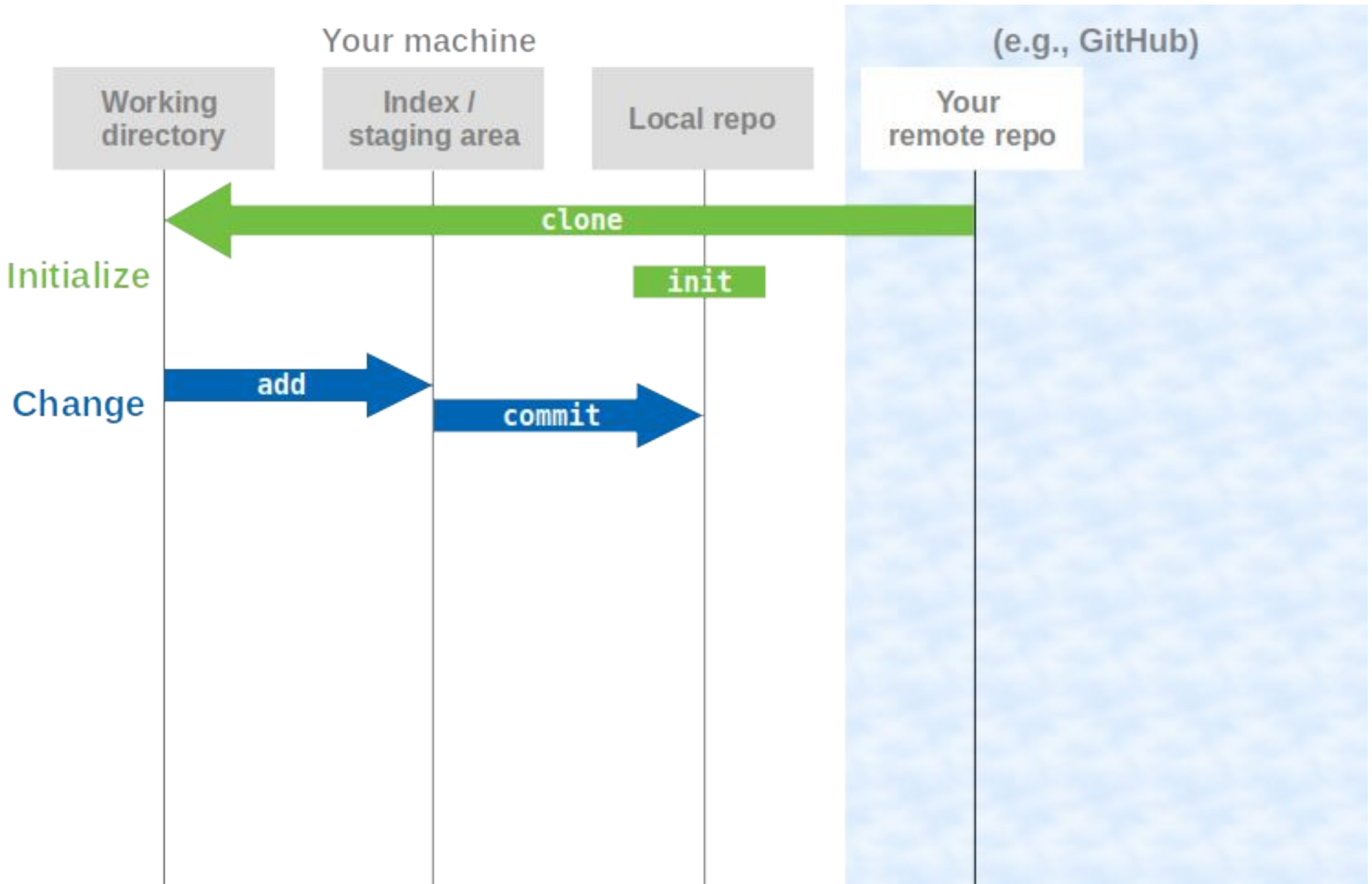
2 options

1. **Clone an existing repo (e.g., from GitHub)**

   ```
   git clone <repo URL>
   ```

2. **Make an existing folder into a git repo**

   ```
   cd <directory>
   git init
   ```

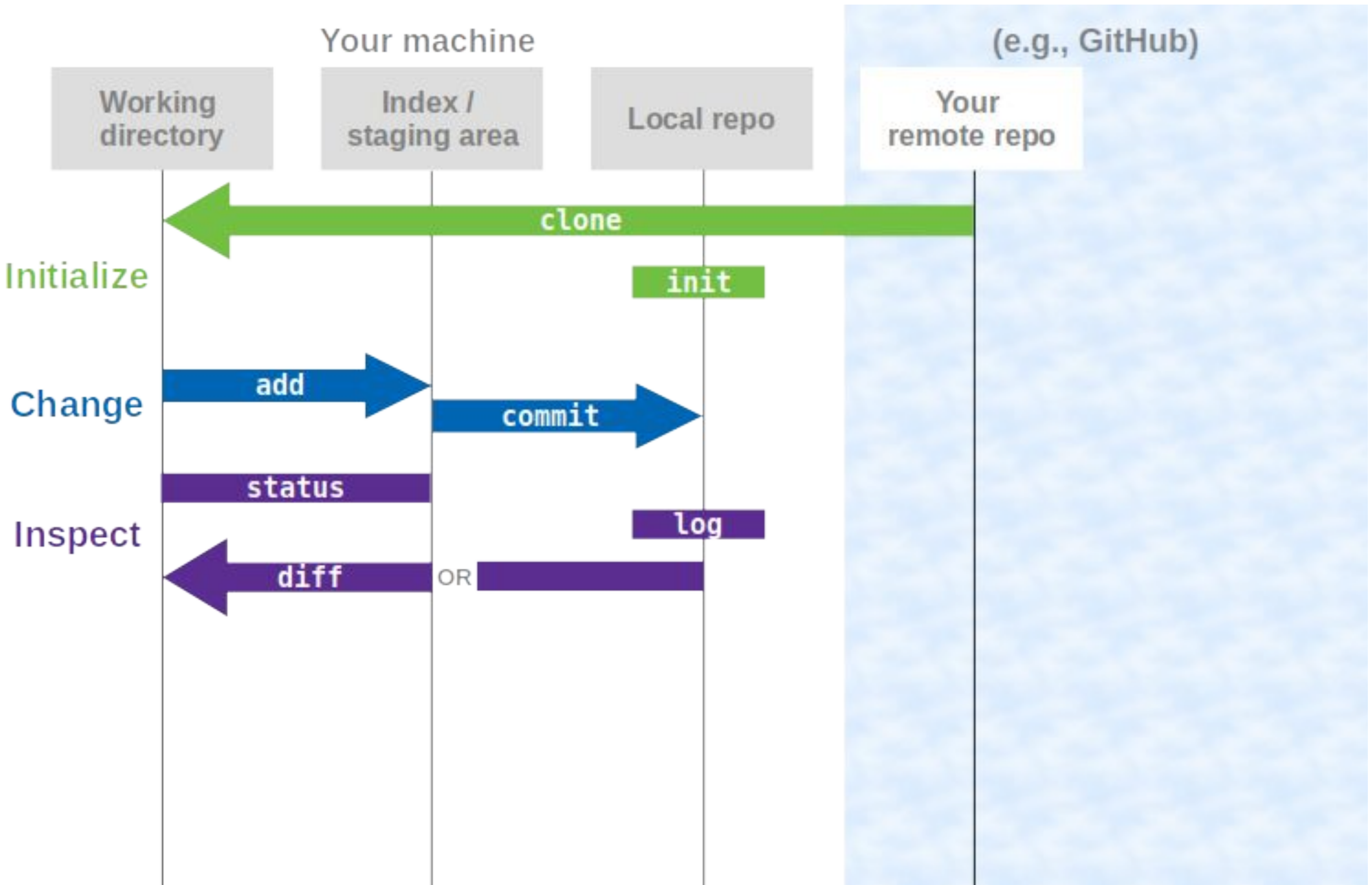


Inspecting
(useful commands that don't *do* anything)

- Check the status of the files in your repo
  `git status`

- See what changed
  `git diff`

- See the history of your repo
  `git log`
  Note: type 'q' to exit the log



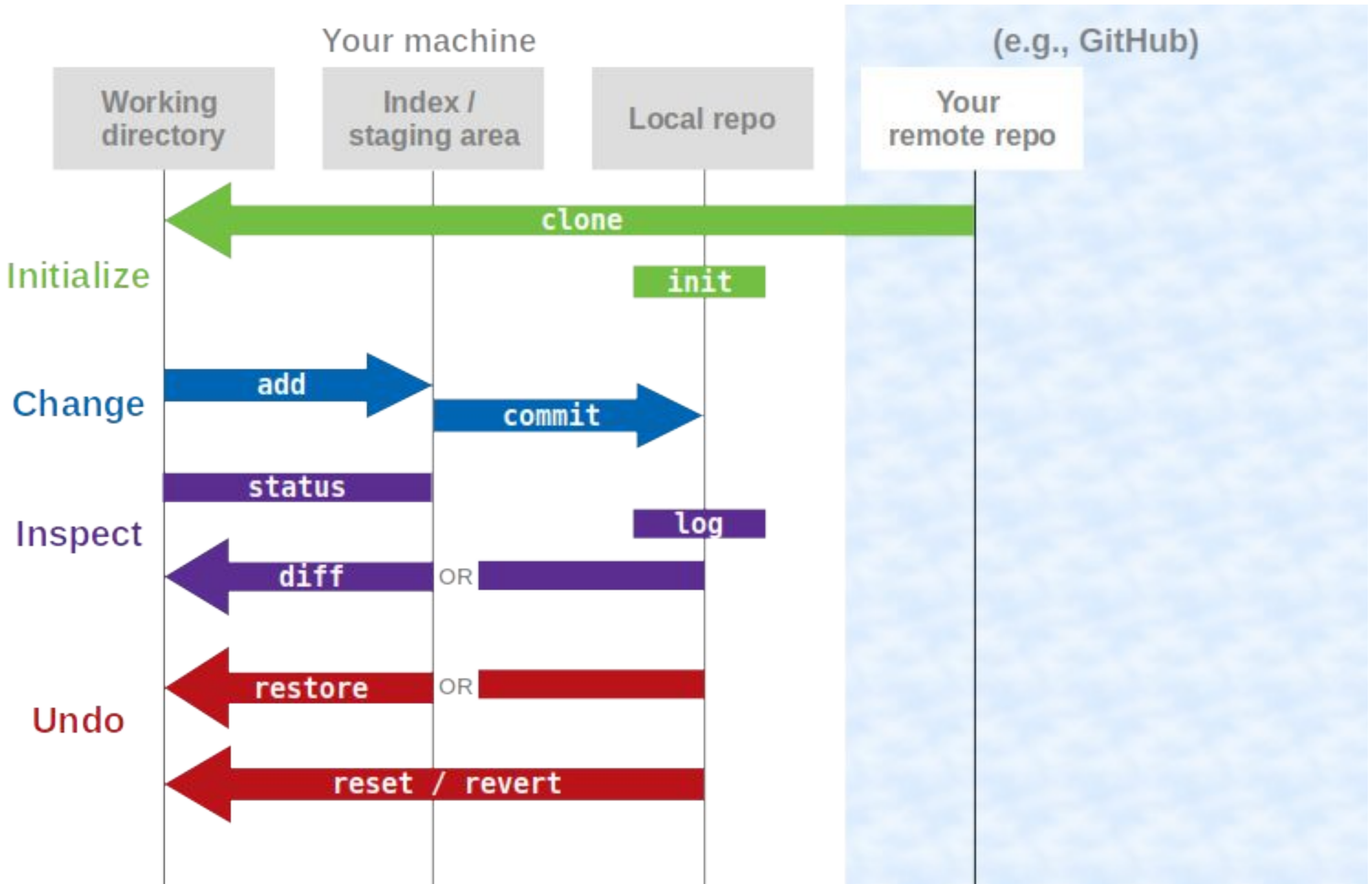
Common undoing goals

- **Unmodify** a file
  `git restore <file>` or `git checkout -- <file>`

- **Unstage** a file
  `git restore --staged <file>` or `git reset HEAD <file>`

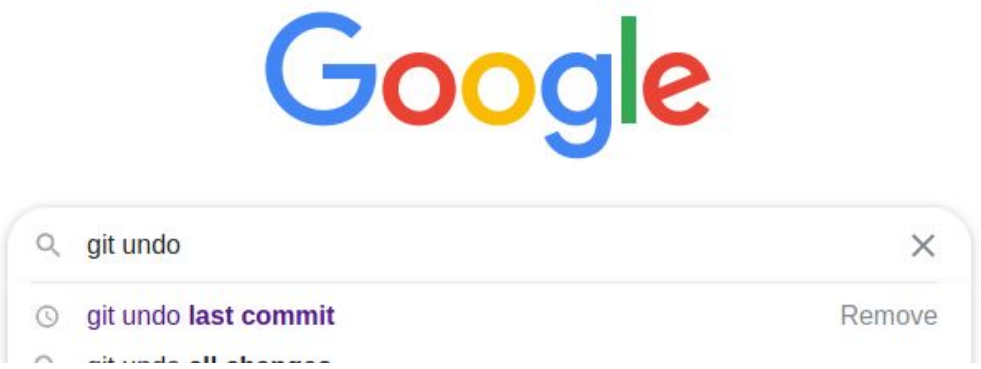
- Forgot a file in the last **commit**
  `git add <file>`
  `git commit --amend`

- Undo the last **commit**
  `git reset HEAD~`



Undoing feels intimidating at first

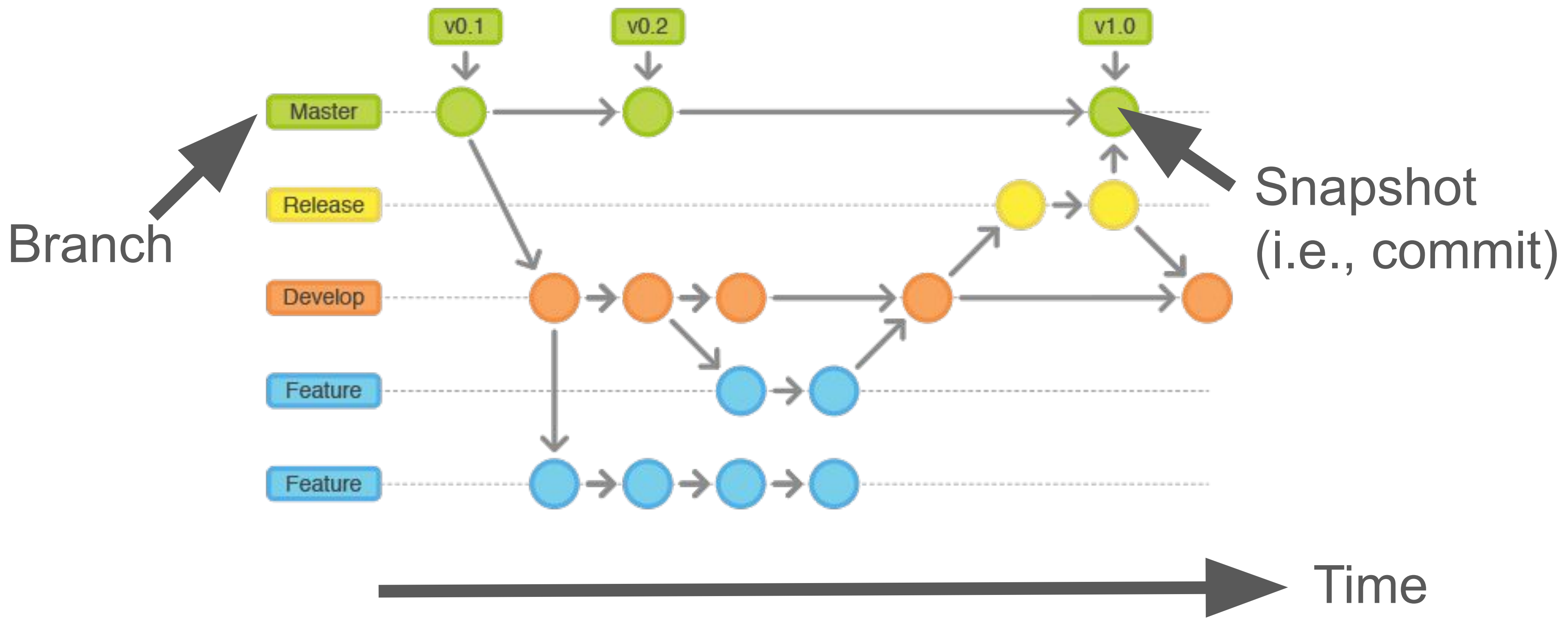
- Don’t try to memorize all the commands
- Try it out on a sandbox repo first
- Use `git checkout <commit hash> -b <branch name>` to start a new branch at the point you want to revert to in order to see what it would be like
- For minor recent fixes, try reading the output of `git status`
- Look it up; someone probably had your question before



Branches

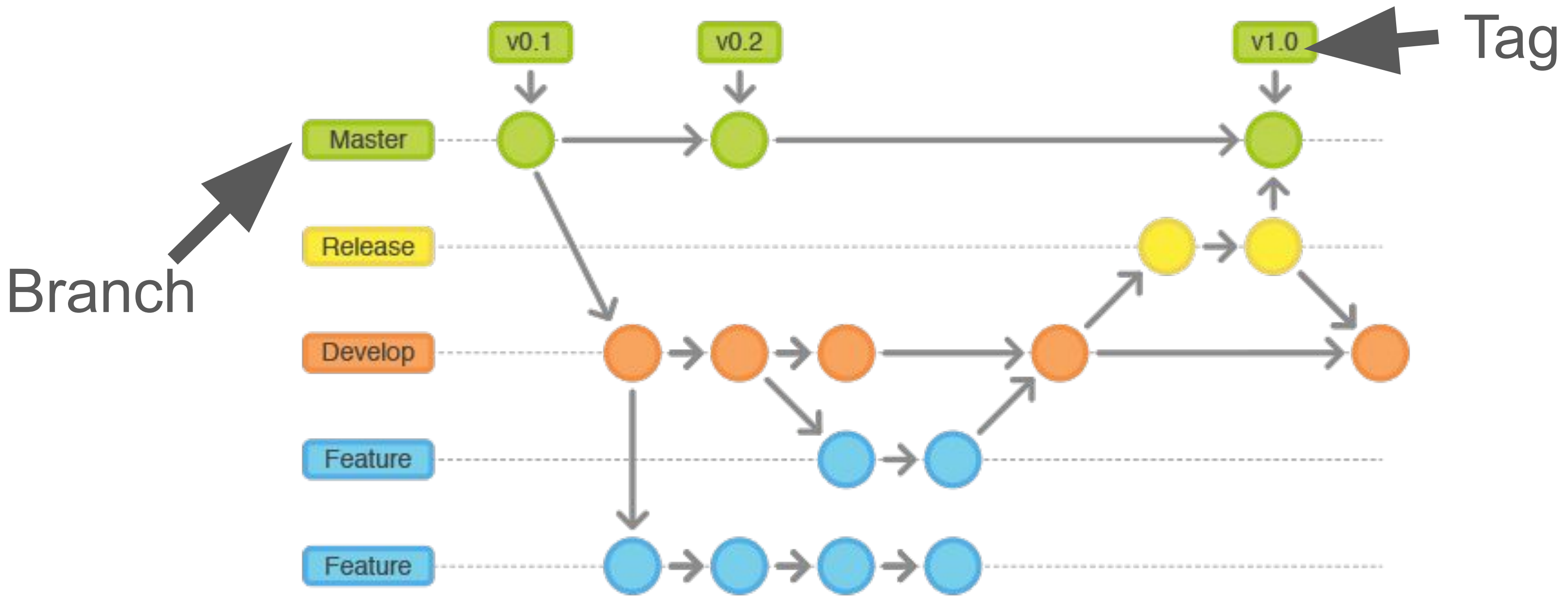
- For nonlinear development
  - Linux: In the last month, “1067 authors have pushed 5,329 commits…” [https://github.com/torvalds/linux/pulse/monthly, as of 2021-07-14]

Branches



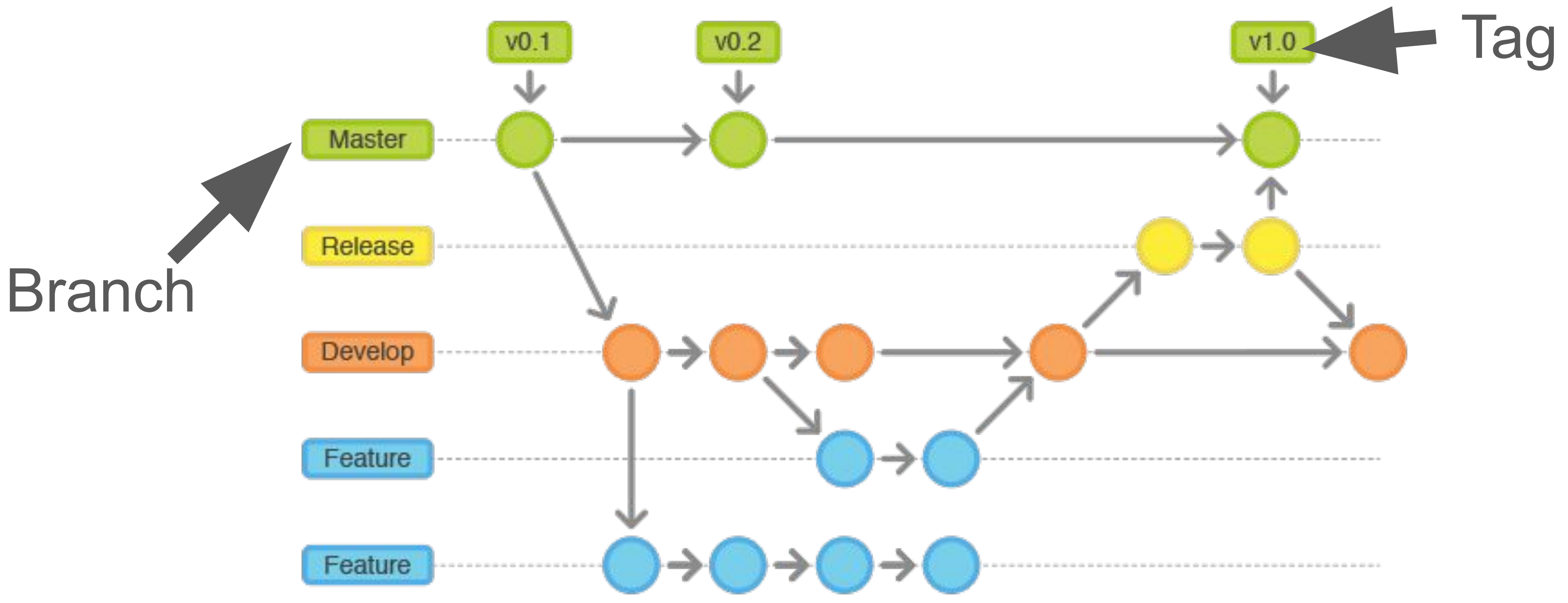
Tip: GitHub has a great way of viewing a project's "network"

Branches vs tags



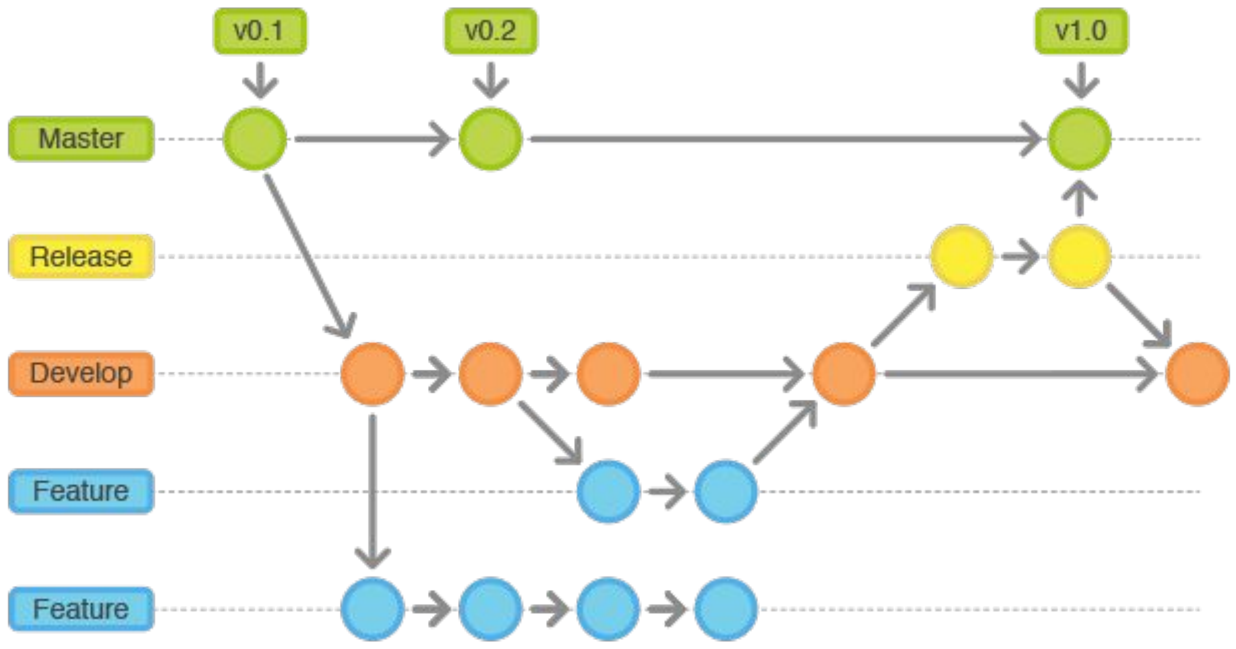
	What is it for?	
Branch	**Marks a line of development** E..g, a new feature, a collaborator’s contribution	
Tag	**Marks an important point in history** E..g, a version of a software package, a paper publication	

Branches vs tags



	What is it for?	What *exactly* is it?
Branch	**Marks a line of development** E..g, a new feature, a collaborator's contribution	**An text file** Filename: branch name Contents: commit hash for the latest commit in that branch
Tag	**Marks an important point in history** E..g, a version of a software package, a paper publication	**An text file** Filename: tag name Contents: commit hash for the commit when the tag was created

Branches



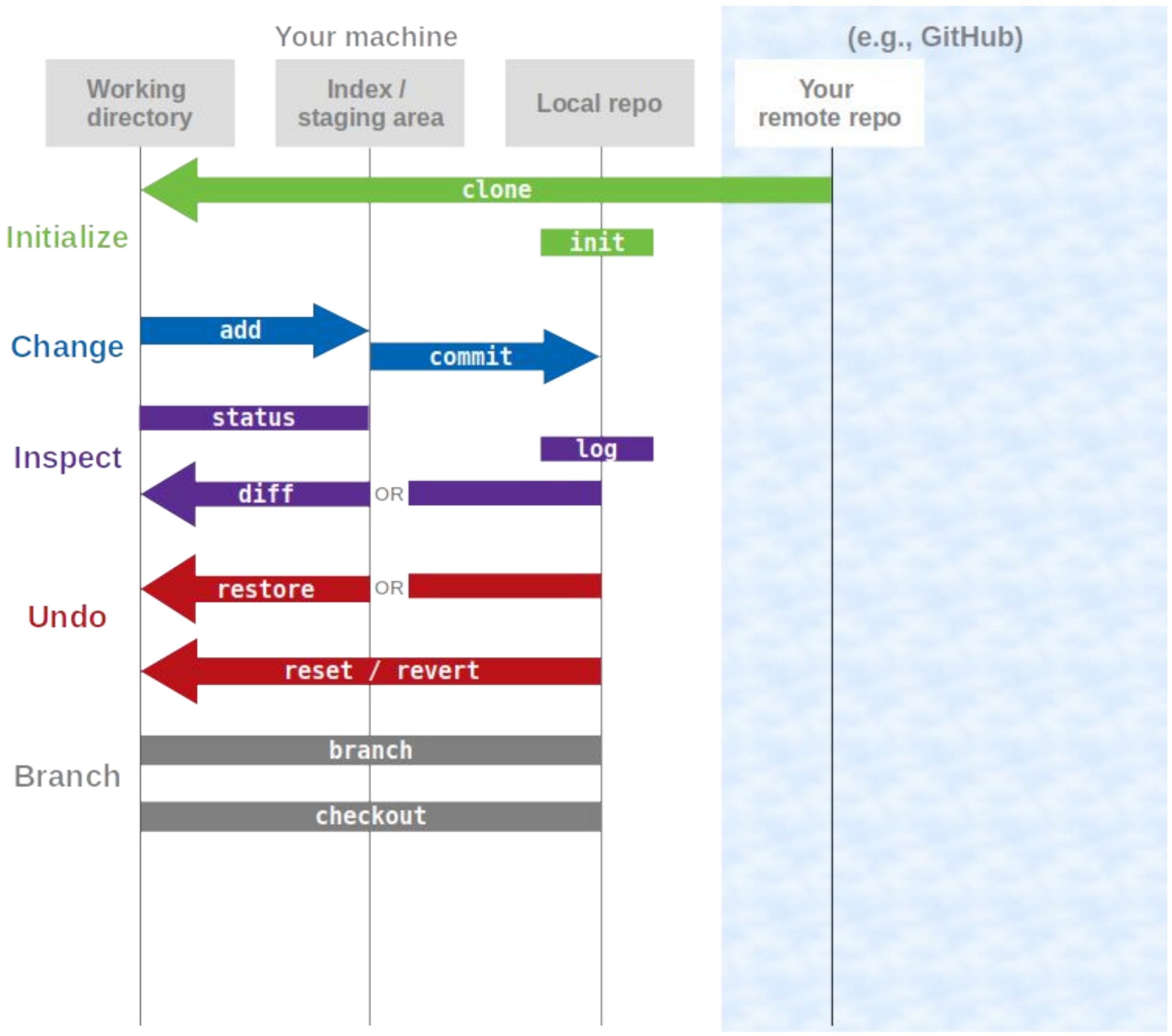
- See which branch you're on
  `git branch`

- Start a new branch
  `git branch <branch name>`

- Change branches
  `git checkout <branch name>`

- Merge a branch into your current branch
  `git merge <branch name>`

Branches


Your machine
(e.g., GitHub)
Working directory
Index / staging area
Local repo
Your remote repo
clone
Initialize
init
Change
add
commit
status
Inspect
log
diff
OR
restore
OR
Undo
reset / revert
branch
Branch
checkout

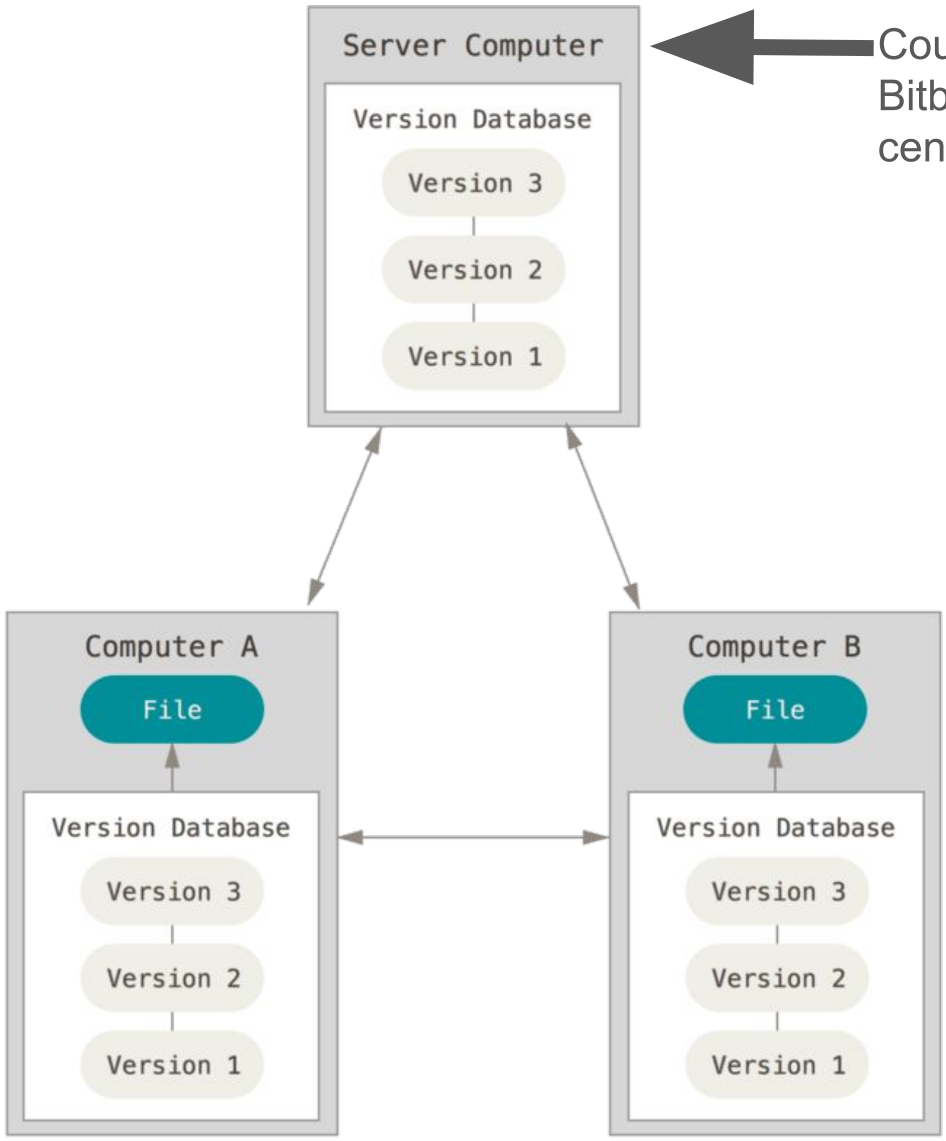
Goals

- What is distributed version control?
- Why is Git useful?
- **Track your own work with Git; and**
- Share your work and collaborate on GitHub.

Goals

- What is distributed version control?
- Why is Git useful?
- Track your own work with Git; and
- **Share your work and collaborate on GitHub.**

Remotes

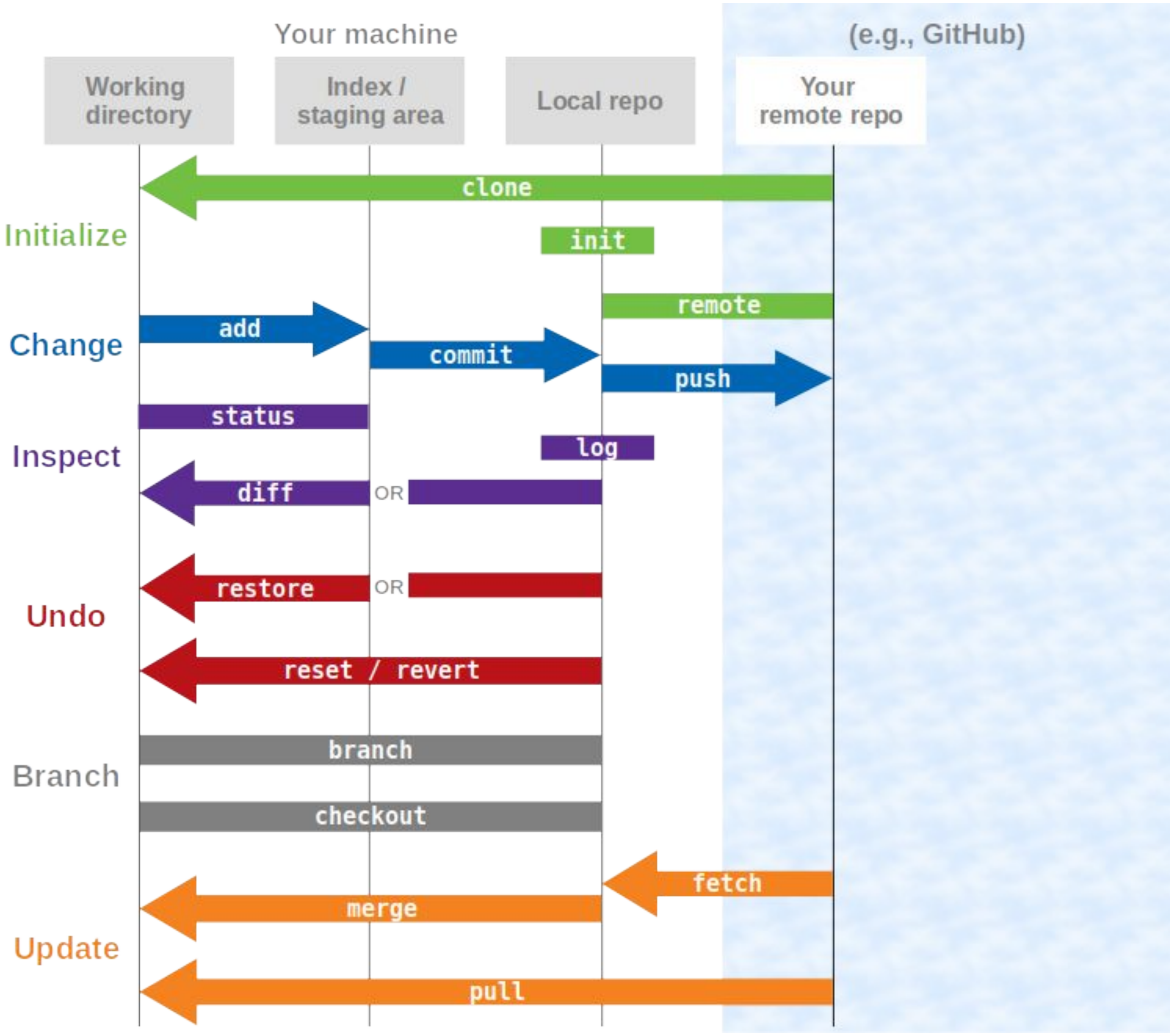


Could be GitHub, Bitbucket, Gitlab, a central server...

Remotes

- Show your remote repos
  ```
  git remote -v
  ```
- Add a remote repo
  ```
  git remote add <remote name> <remote address>
  ```
- Push commits to a remote repo
  ```
  git push <remote name> <branch>
  ```
- Fetch commits from a remote repo
  ```
  git fetch <remote name>
  ```
- Merge fetched commits from a remote repo
  ```
  git merge <remote name>/<branch>
  ```

Remotes - sharing your own work

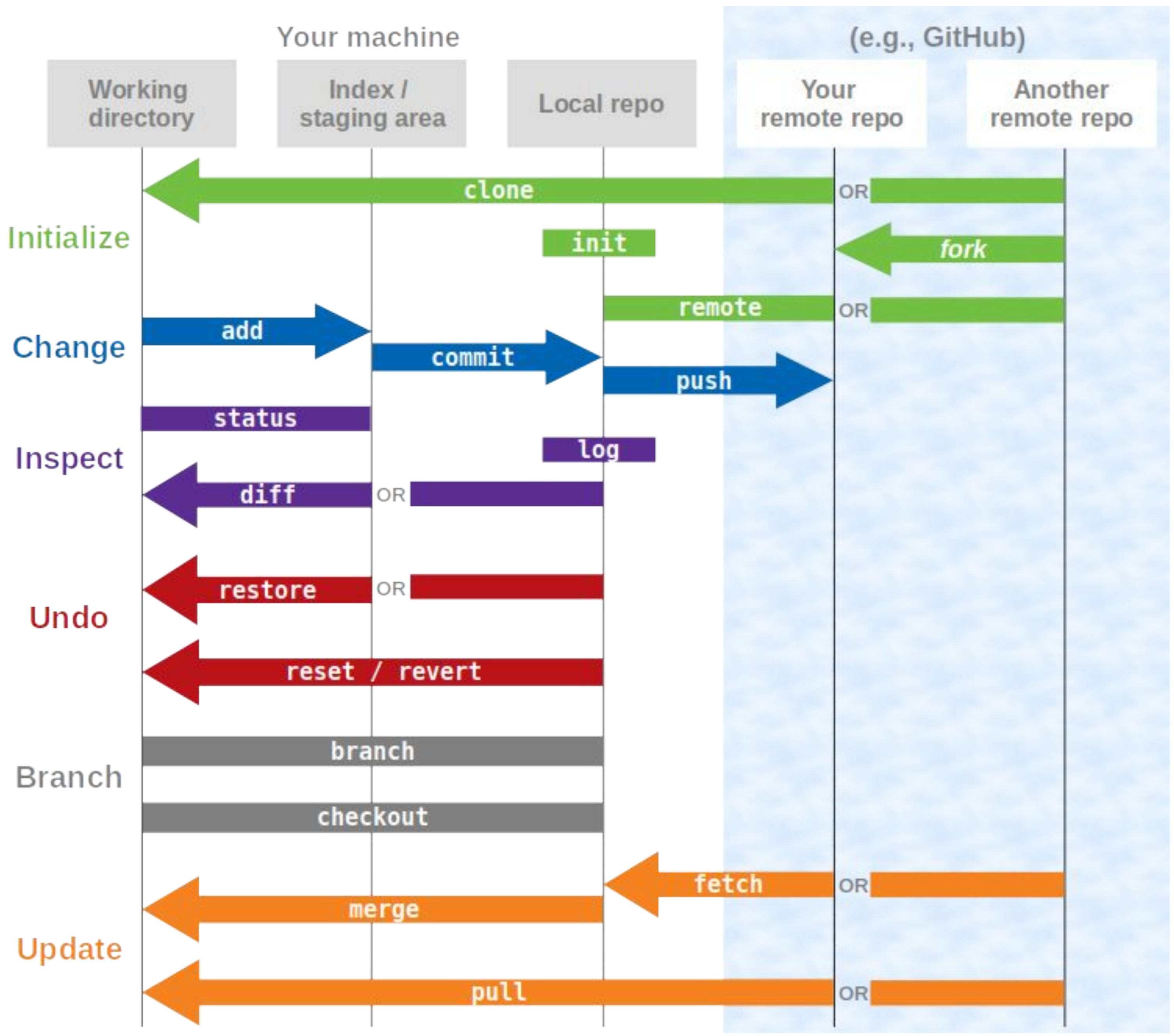


Remotes - collaborating/contributing

Forks

- Like a clone, but on GitHub
- Most collaborators will have their own fork where they work

Forks

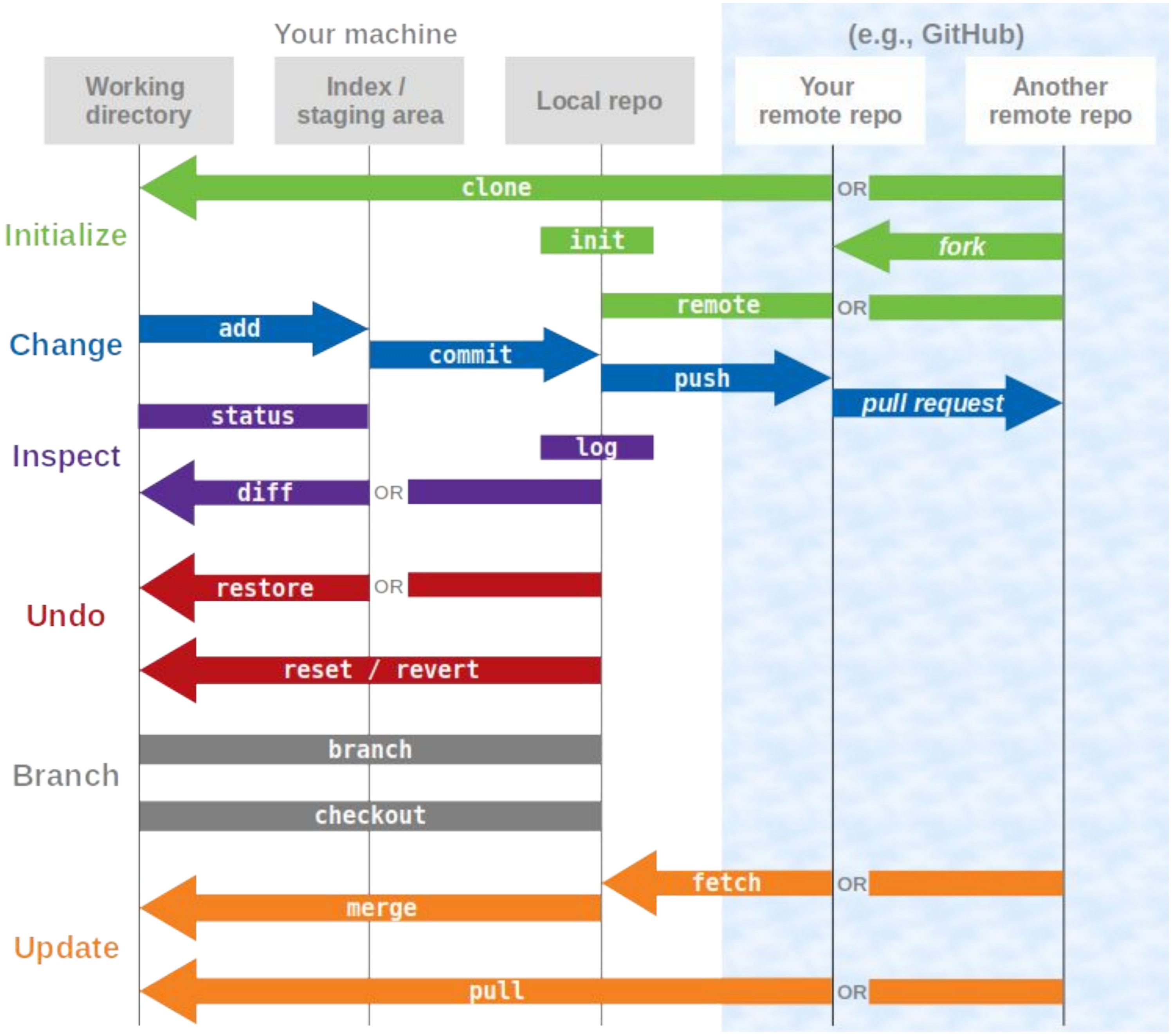

Your machine
(e.g., GitHub)
Working directory
Index / staging area
Local repo
Your remote repo
Another remote repo
Initialize
clone
OR
init
fork
remote
OR
Change
add
commit
push
Inspect
status
log
diff
OR
Undo
restore
OR
reset / revert
Branch
branch
checkout
Update
fetch
OR
merge
pull
OR

Remotes - collaborating/contributing

Pull requests

- Push your commits to your own fork, then open a pull request
- Then the project maintainers can
  - review your code
  - make suggestions / edits
  - decide whether to merge it into the original repository

Pull requests

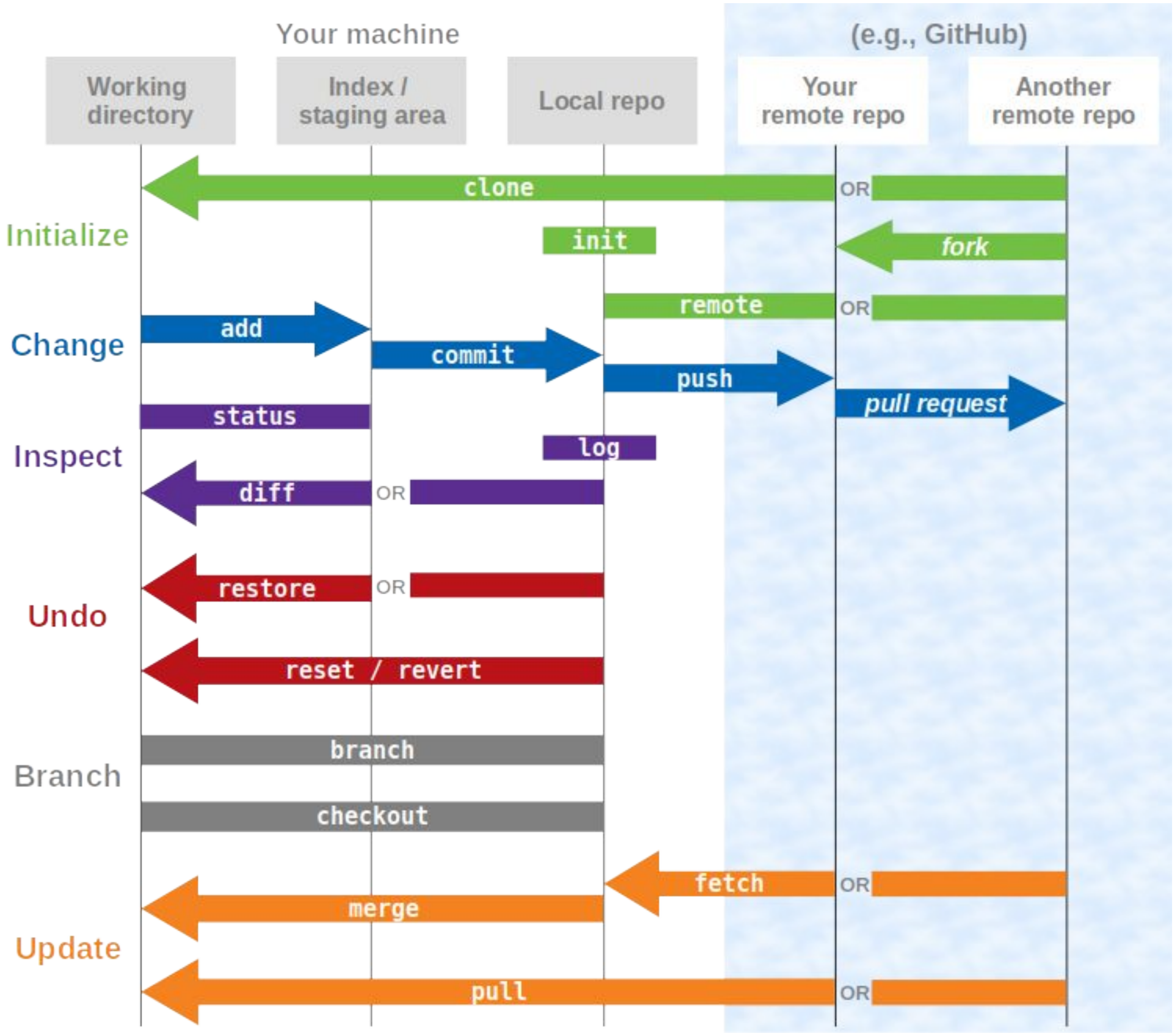


Remotes - collaborating/contributing

Issues

- Report a problem
- Propose something new
- Find something to work on

Remotes - collaborating/contributing

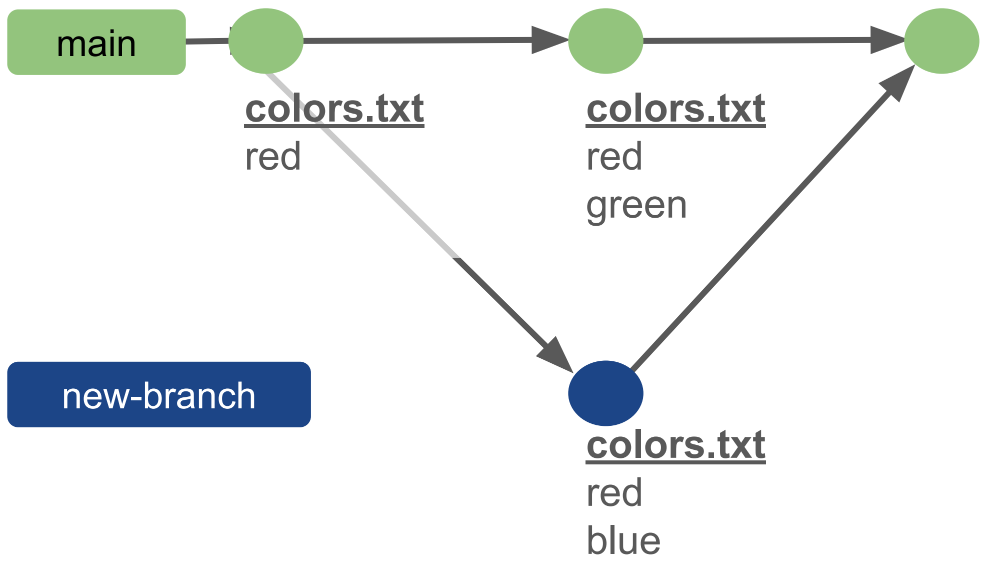


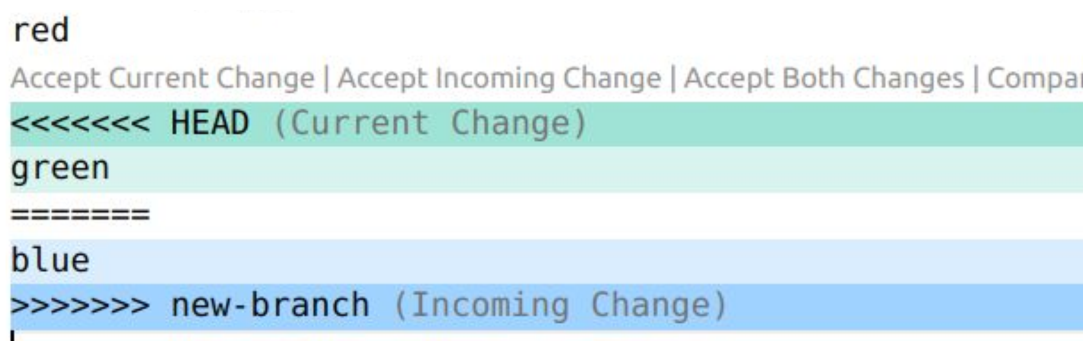
Merge conflicts

- What if several people edit the same line of code?

  → merge conflict

- Someone needs to manually resolve it




Accept Current Change | Accept Incoming Change | Accept Both Changes | Compa

```
red
<<<<<<< HEAD (Current Change)
green
=======
blue
>>>>>>> new-branch (Incoming Change)
```

Goals

- What is distributed version control?
- Why is Git useful?
- Track your own work with Git; and
- **Share your work and collaborate on GitHub.**

Some final tips

Options

- Commands can become powerful with options

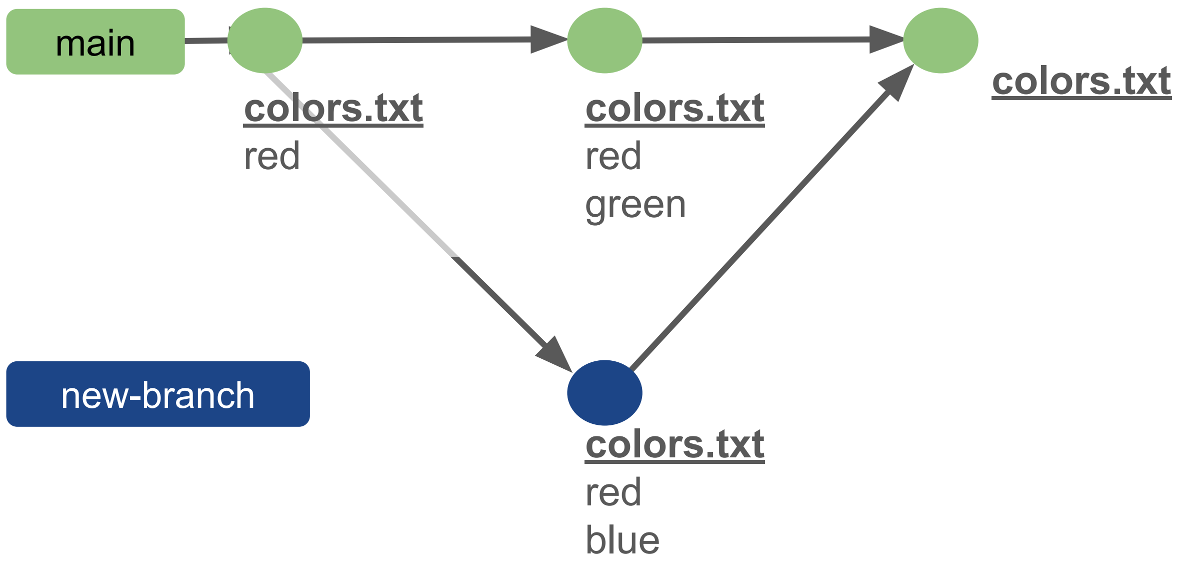
```
git log --pretty=format:"%h - %an, %ar : %s" --graph
```

- “How on earth will I remember that??”

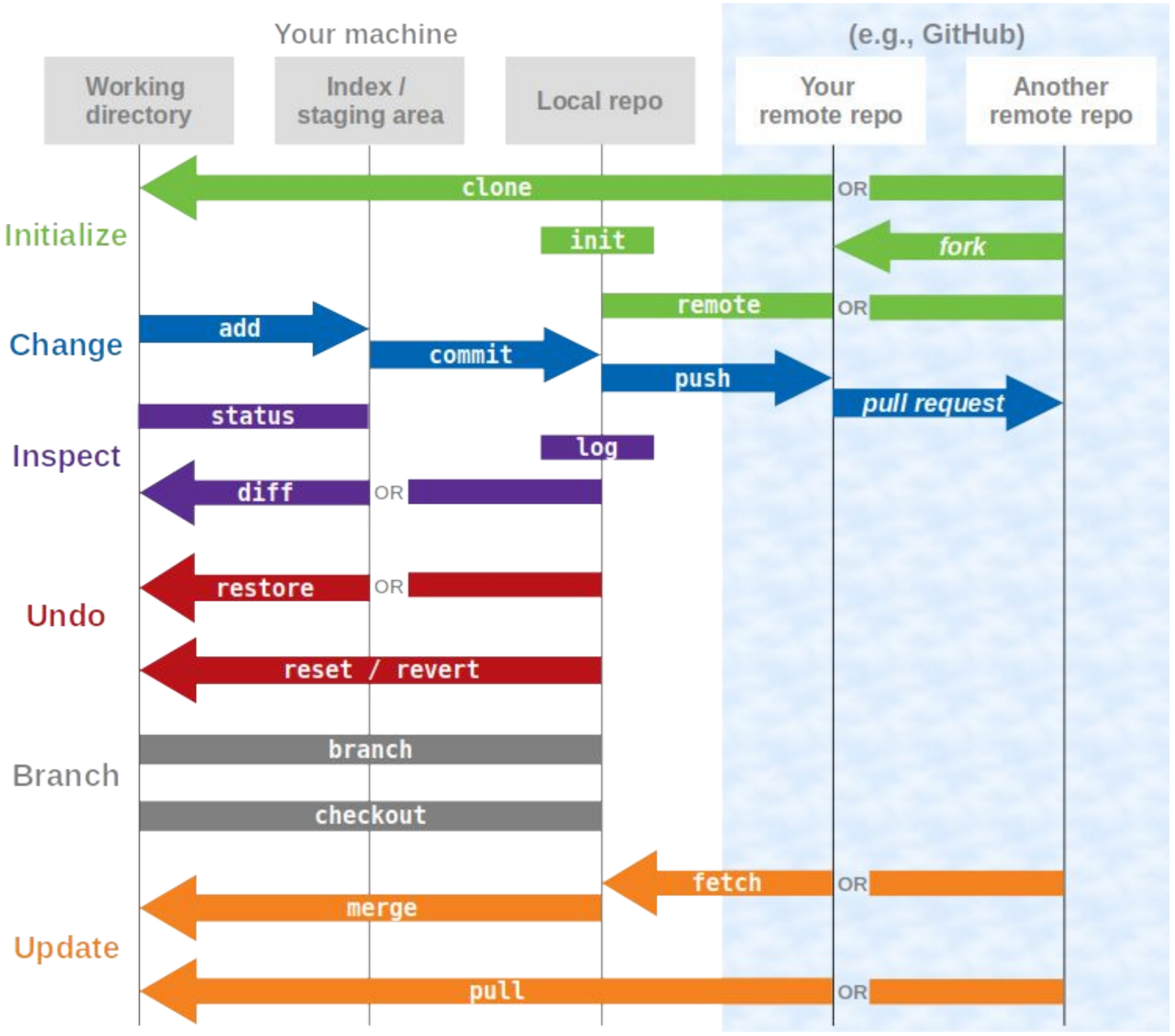
- You can set aliases for commands you use a lot

```
git config --global alias.fancylog 'log
--pretty=format:"%h - %an, %ar : %s" --graph'

git fancylog
```

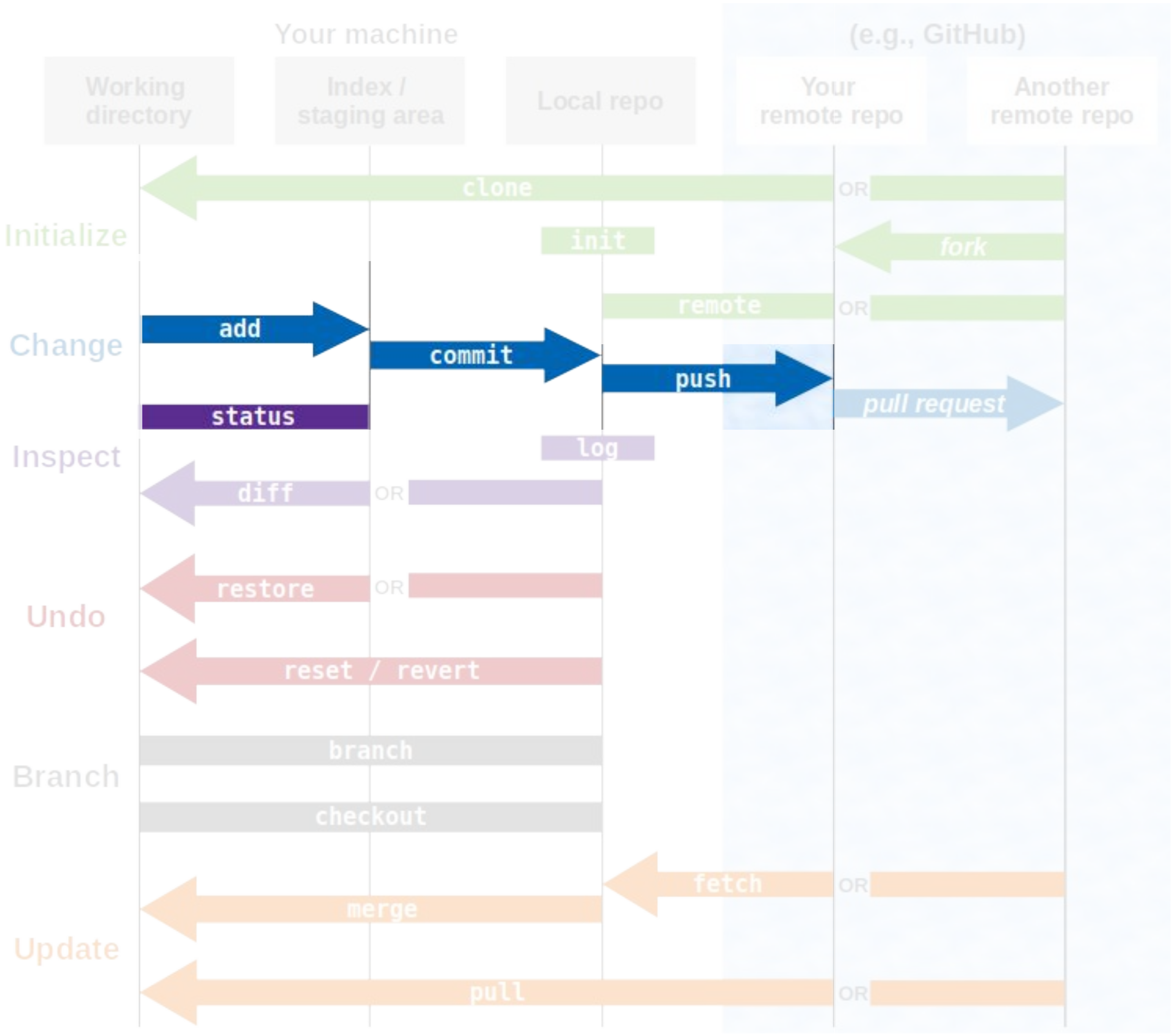


Pick a bite-size amount to start with



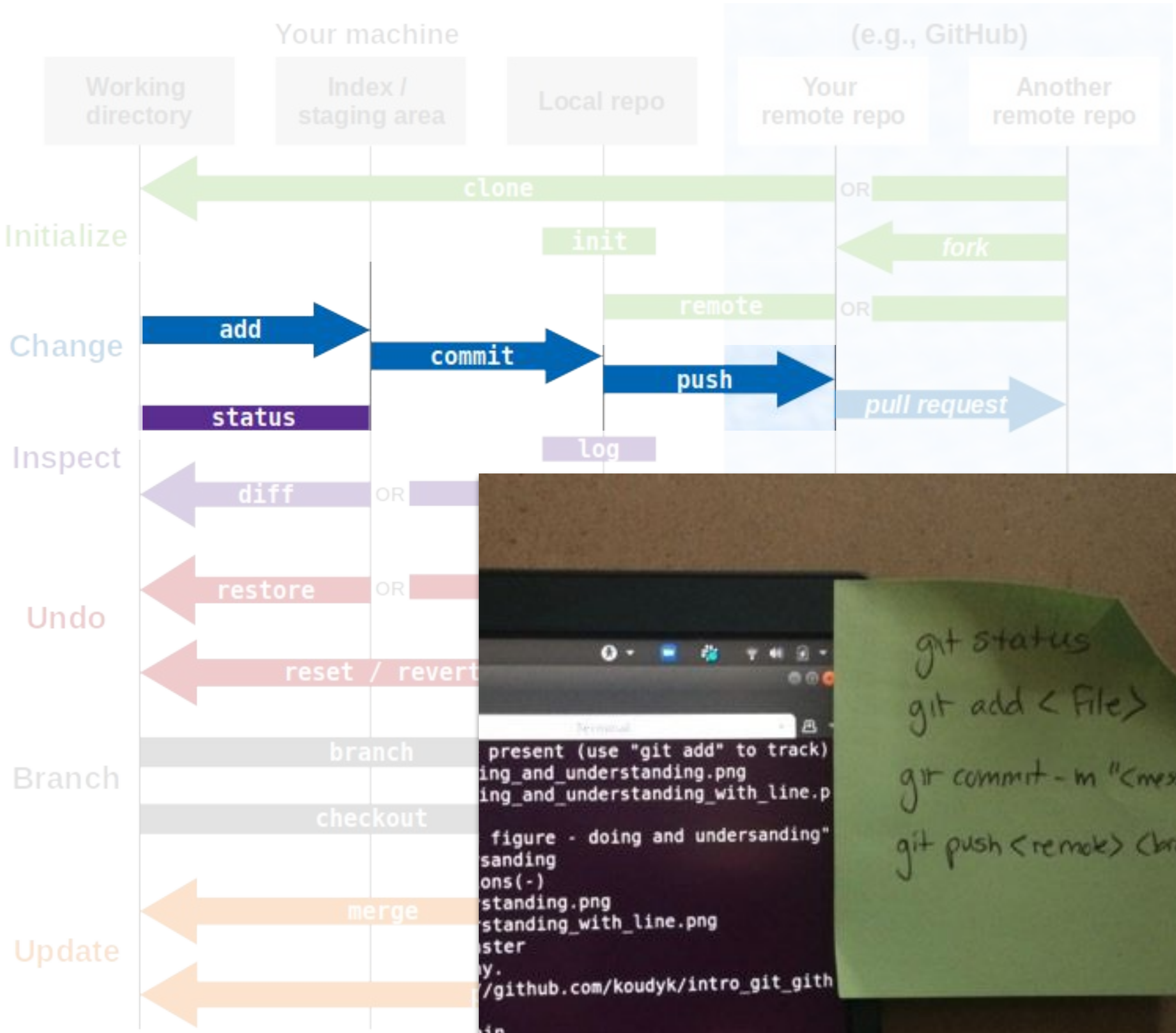
(Everything else you can look up when you need it)

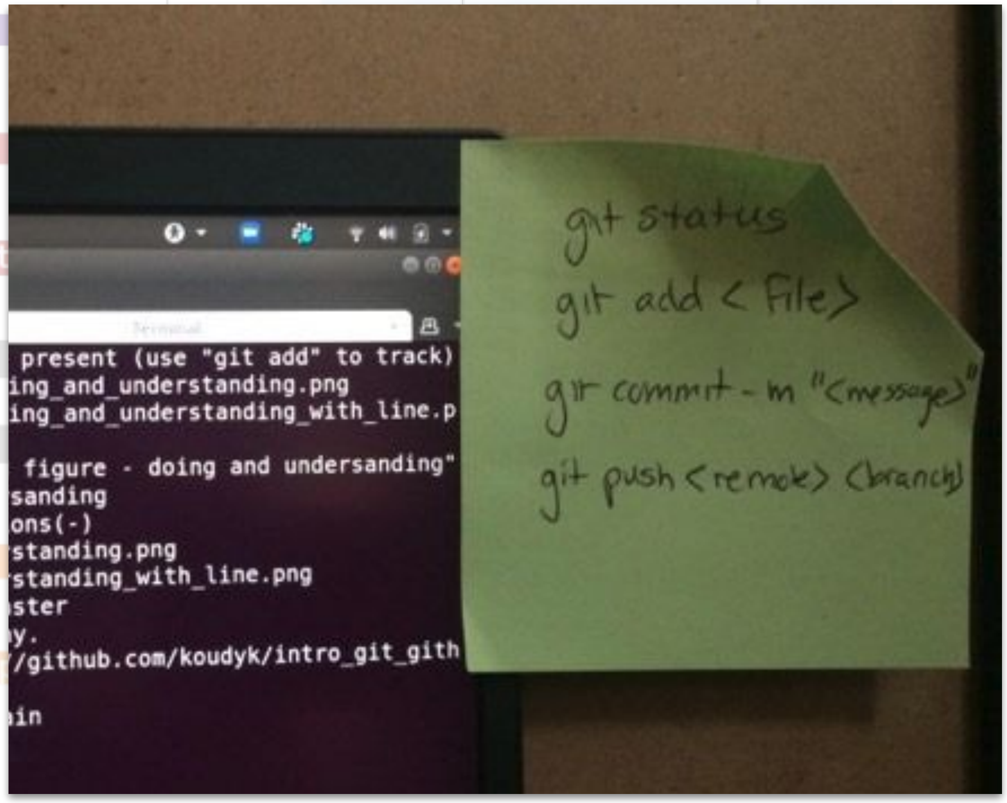
Pick a bite-size amount to start with



(Everything else you can look up when you need it)

Pick a bite-size amount to start with





```
present (use "git add" to track)
ing_and_understanding.png
ing_and_understanding_with_line.p

 figure - doing and undersanding"
sanding
ons(-)
standing.png
standing_with_line.png
ster
y.
/github.com/koudyk/intro_git_gith

in
```

```
git status
git add <file>
git commit -m "<message>"
git push <remote> <branch>
```

(Everything else you can look up when you need it)

Sharing your work on GitHub (a type of remote)

If you already have a local repo

- Go to github.com/<your github username>
- Click “Repositories”
- Click “New”
- Enter your repo name
- Choose if you want it to be public or private
- DON’T click any boxes to add README, .gitignore, or licence files
- Follow the instructions for the option “push an existing repository from the command line”

Note: If these instructions are outdated, check out the GitHub documentation

To follow on your machine, you'll need

1. Bash
2. Git
3. Text editor
4. GitHub account

Check if you're ready

☑️ **Can you open a bash shell?**

- Open a terminal, type `echo $SHELL` and press ENTER.
- The output should be `/bin/bash`

☑️ **Do you have git installed?**

- In the bash terminal, `git --version` and press ENTER.
- The output should be `git version X` (where the X is the version number)
- *Don't worry if you don't have the exact same version as I do*

☑️ **Do you have git configured?**

- In the bash terminal, type `git config --list` and press ENTER
- You should see your name and email (and other things that aren't essential to configure

☑️ **Can you open a text editor?** E.g.,

- Linux: gedit, nano
- macOS: textedit
- Windows: notepad

☑️ **Can you go your GitHub account?**

Initial setup

1. Tell git who you are

   ```
   git config --global user.name "John Doe"
   git config --global user.email johndoe@example.com
   ```

2. Tell git your default branch name

   ```
   git config --global init.defaultBranch main
   ```

3. D
4.

Are you ready?

You should have some familiarity with bash to understand my demos

- `ls`
- `cd`
- Accessing help manuals
- Options

All together

Repeat over and over as you work

- **Modify**
  Change a file in your working tree
- **Stage**
  `git add <filename>`
- **Commit**
  `git commit -m “<short, informative commit message>”`
- **Push**
  `git push <remote name> <branch>`

Inspect what's happening at any time

- **See file states**
  `git status`
- **See differences**
  `git diff`
- **See the repo history**
  `git log`

At the beginning

- **Initialize local repo**
  `cd <folder>` then `git init`
- **Make remote repo**
  on GitHub
- **Link local and remote repos**
  `git remote add <url of GitHub repo>`

Undoing and rewriting history

- **Unmodify a file**
  `git  restore <file>`
- **Unstage a file**
  `git restore --staged <file>`
- **Forgot a file in the last commit**
  `git add <file>`
  `git commit --amend`
- **Undo the last commit**
  `git reset HEAD~`

Choose a bite-size amount to remember

(Everything else you can look up when you need it)

Repeat over and over as you work

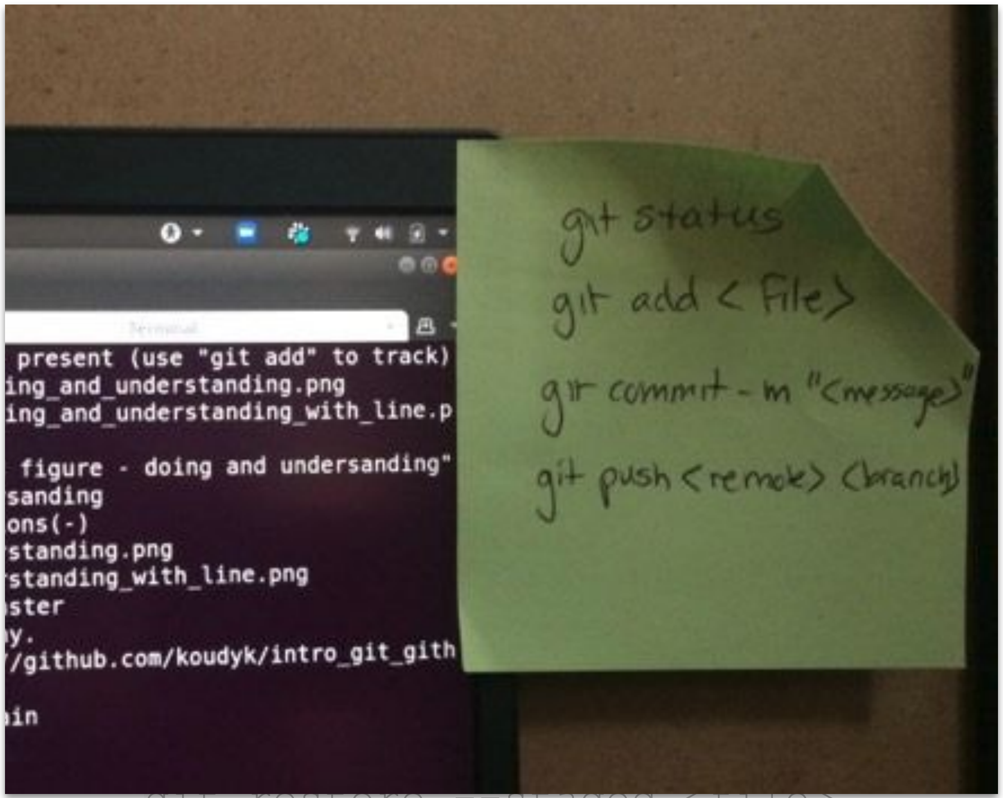
- **Modify**
  Change a file in your working tree
- **Stage**
  `git add <filename>`
- **Commit**
  `git commit -m "<short, informative commit message>"`
- **Push**
  `git push <remote name> <branch>`

Inspect what's happening at any time

- **See file states**
  `git status`
- **See differences**
  `git diff`
- **See the repo history**
  `git log`

At the beginning

- **Initialize local repo**
  `cd <folder>` then `git init`



```
present (use "git add" to track)
ing_and_understanding.png
ing_and_understanding_with_line.p

 figure - doing and undersanding"
sanding
ons(-)
standing.png
standing_with_line.png
ster
y.
/github.com/koudyk/intro_git_gith

in
```

git status
git add < File>
git commit -m "<message>"
git push <remote> <branch>

`git restore --staged <file>`

- **Forgot a file in the last commit**
  `git add <file>`
  `git commit --amend`
- **Undo the last commit**
  `git reset HEAD~`

Basic undoing

- Reset current HEAD to the specified state
  `git reset`

- Restore working tree files
  `git restore`

- Revert some existing commits
  `git revert`

- ?
  `git checkout`

!! Use with caution !!

Once you track something, it’s *very* hard to delete it

- You can try new things without worrying about messing up something that works

- If your repo is or will be open, don’t track passwords, API keys, sensitive info, etc.

- **Git makes it hard to permanently delete something, but sometimes it’s hard to recover it**