Open-Source Software Practice

2. Git Basics

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

Version Control System (VCS)

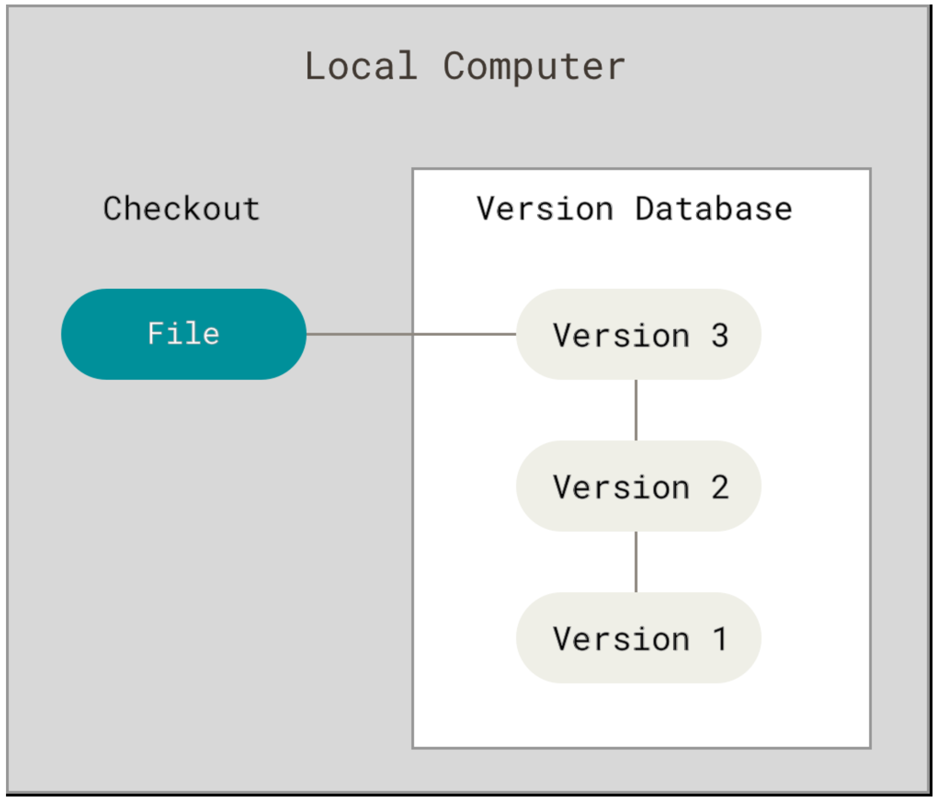
- **A version control system** is a system that records changes to a file or set of files over time.
  - i.e., code versioning

  - revert selected files back to a previous state
  - revert the entire project
  - compare changes over time
  - see who last modified something that might be causing a problem
  - who introduced an issue and when

Local Version Control

- Local version control has limitations.

- No back-ups
- No collaboration
- Hard to maintain multiple files.

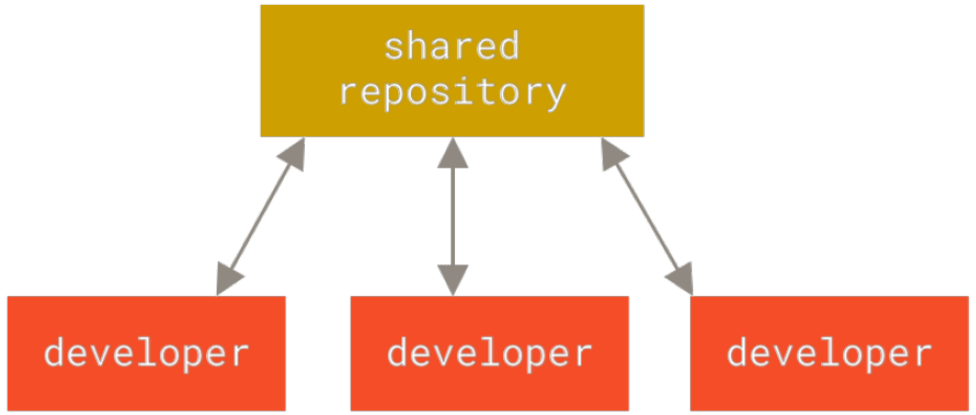


Centralized Version Control

- Centralized version control systems are better but have drawbacks.

- Single point of failure
- If the shared server is down, you cannot save versioned changes to anything.
- Require connection to the server

- Examples: CVS, Subversion, Perforce

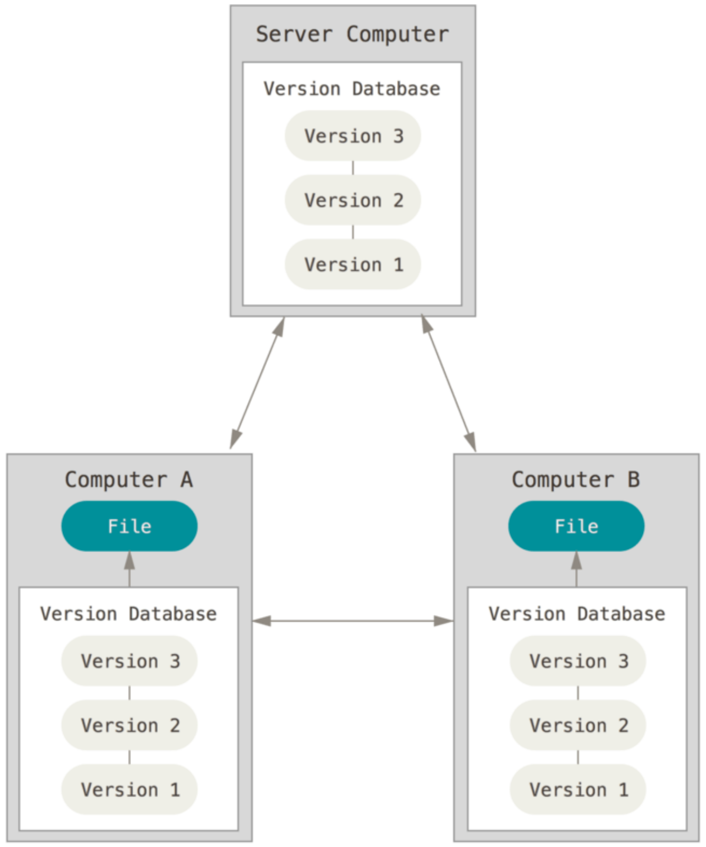


Distributed Version Control

- Distributed version control systems fully mirror the remote repository to a local machine, including its full history.

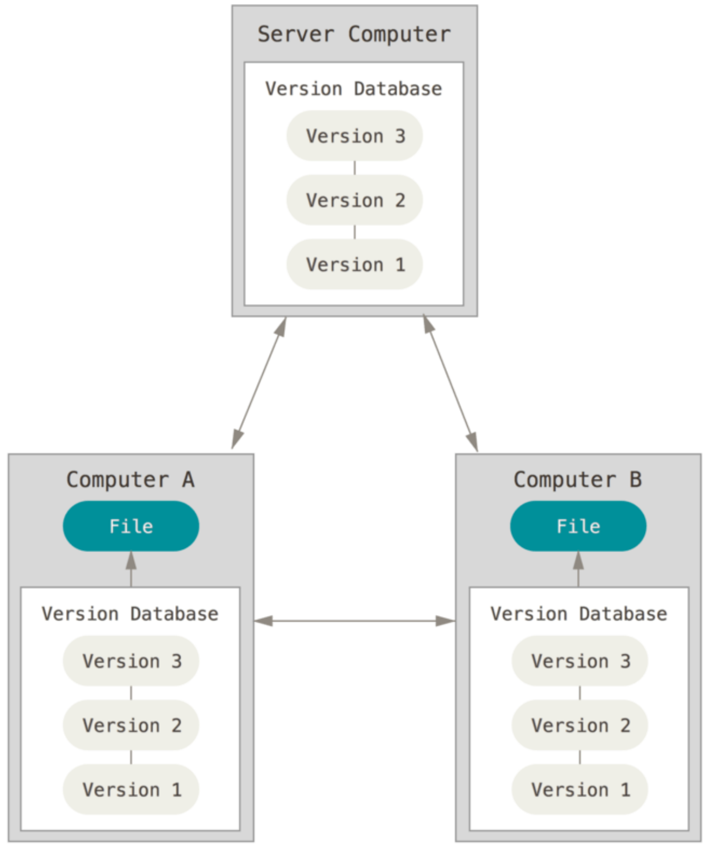
- Even the server dies, any of the client repos can be copied back up to the server.

- Examples: ***Git***, Mercurial, Bazzar, Darcs



Distributed Version Control

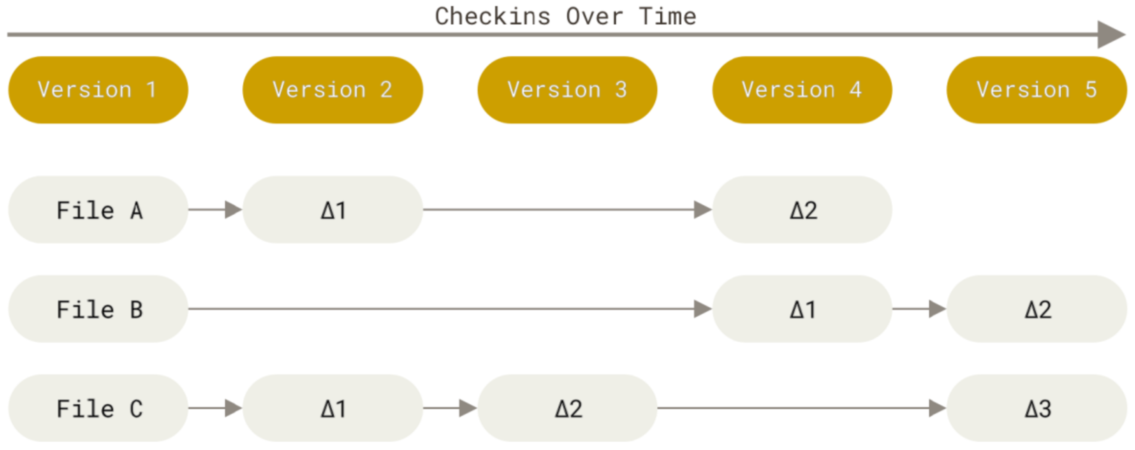
- For most of time with Git, you will save code changes to your local repos.
  - `git commit`
- Sometimes, you will upload the changes in the local repo to the remote.
  - `git push`
- Or, download the changes from the remote to reflect changes made by others.
  - `git fetch`



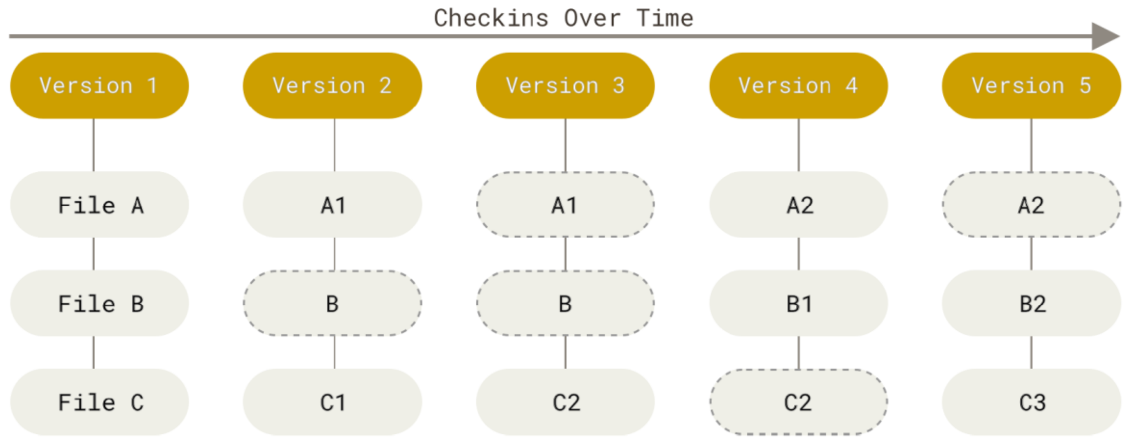
Snapshots, Not Differences

- Git uses snapshot-based version control
  - c.f. delta-based version control which is used in many other VCSs

- Git stores the snapshots of every file modified in each version.
  - Faster lookups, but uses more disk space



Delta-based



Snapshot-based

Nearly Every Operation is Local

- Most operations in Git need only local files and resources to operate.
- This is very convenient especially when the network is unavailable.
- Local operations are done faster than remote ones.

Git Generally Only Adds Data

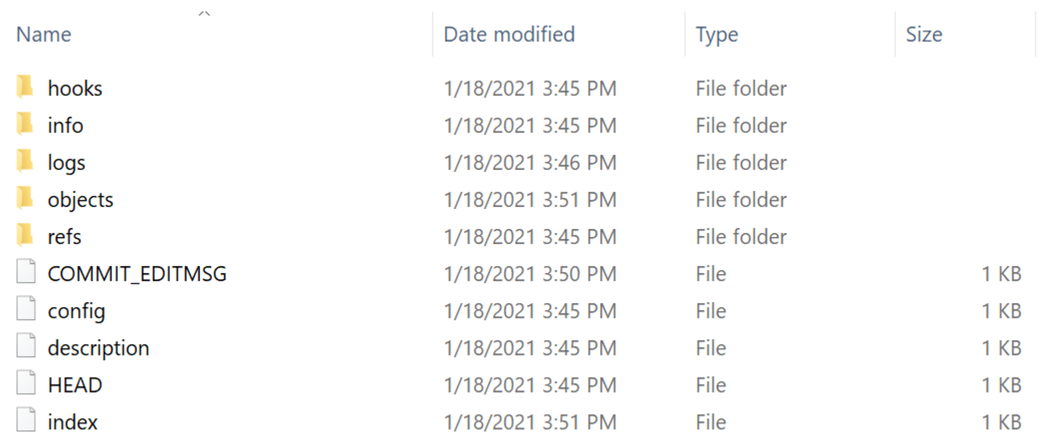
- When you do actions in Git, nearly all of them only add data to the Git database.

- It is hard to make the system to erase data in any way if you regularly push your local repo to the remote.

.git Directory

- If you initialize a Git repo (or cloning a Git repo), a hidden directory named ".git" is created.

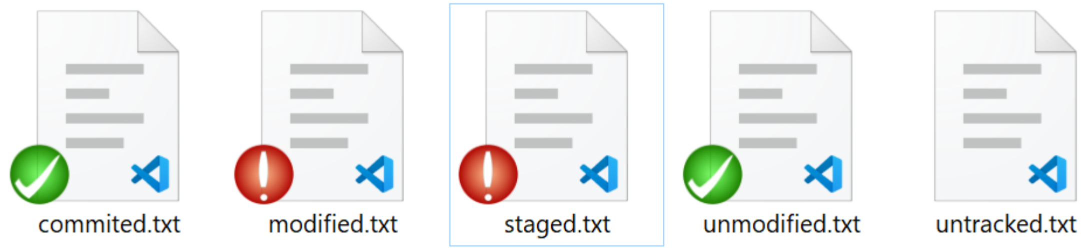
- This is the local database of the Git repo, and you must not delete or modify this directory.



Name	Date modified	Type	Size
hooks	1/18/2021 3:45 PM	File folder	
info	1/18/2021 3:45 PM	File folder	
logs	1/18/2021 3:46 PM	File folder	
objects	1/18/2021 3:51 PM	File folder	
refs	1/18/2021 3:45 PM	File folder	
COMMIT_EDITMSG	1/18/2021 3:50 PM	File	1 KB
config	1/18/2021 3:45 PM	File	1 KB
description	1/18/2021 3:45 PM	File	1 KB
HEAD	1/18/2021 3:45 PM	File	1 KB
index	1/18/2021 3:51 PM	File	1 KB

Four States of Files

- Your files in a Git repo will be in any of the following 4 states:
- **Untracked**: The file is not under version control. Ignored by Git.
- **Unmodified**: The file is under VC, but not modified. You don't have to record changes for this file.
- **Modified:** The file is under VC and modified. You should record the changes.
- **Staged**: The file is under VC, modified, and the changes to the file are ready to commit.
- **Committed**: The changes were committed. This is the same as the "unmodified" state.



Why Modified and Staged?

- Why is an extra state "staged"?

- Suppose you modified 10 files.
- Sometimes, you want to record the changes only made to specific files.
  - e.g., commit the changes made to "interface.h" not "server.h"

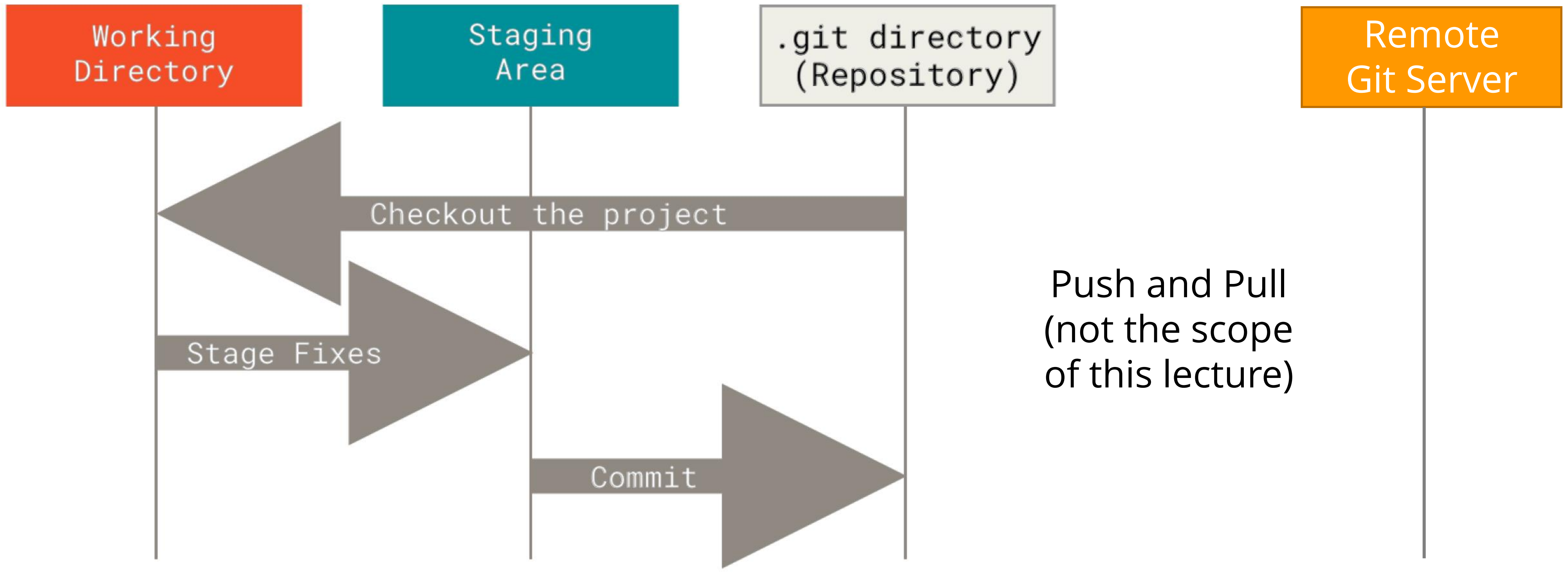
- You can commit files selectively by staging them only.
  - *git add interface.h* vs. *git add .*
  - 10 modified, 5 staged for commit

Three Main Sections

- **Working directory** (or working tree) is a single checkout of one version of the project.
  - Initially, all files are in an unmodified state.
  - Modify some files (using a text editor).
  - Selectively stage files that you want to be part of your next commit (`git add`).
- **Staging area** (or index) has files that are staged for the next commit.
  - Check out the staged files (`git status`), and do a commit (`git commit`)
- **Local Git repo** (.git) stores commits.

- Note that we don't consider a remote repo for now.

Three Main Sections



On your computer

Basic Git Workflow (Addition)

- Suppose you want to add a new file “new.txt” to an existing Git repo.

1. Create “new.txt” using your favorite editor (*untracked* for now).
2. Add “new.txt” to the staging area by “`git add new.txt`” (*staged*).
3. Commit the file by “`git commit`” (*committed and unmodified*).

- untracked → staged → committed (= unmodified)

Basic Git Workflow (Modification)

- Suppose you want to modify the file “new.txt”.

1. new.txt is in an *unmodified* state.
2. Modify new.txt (*modified*).
3. Stage new.txt by “`git add new.txt`” (*staged*).
4. Commit the file by “`git commit`” (*committed and unmodified*).

- unmodified → modified → staged → committed (= unmodified)

Basic Git Workflow (Deletion)

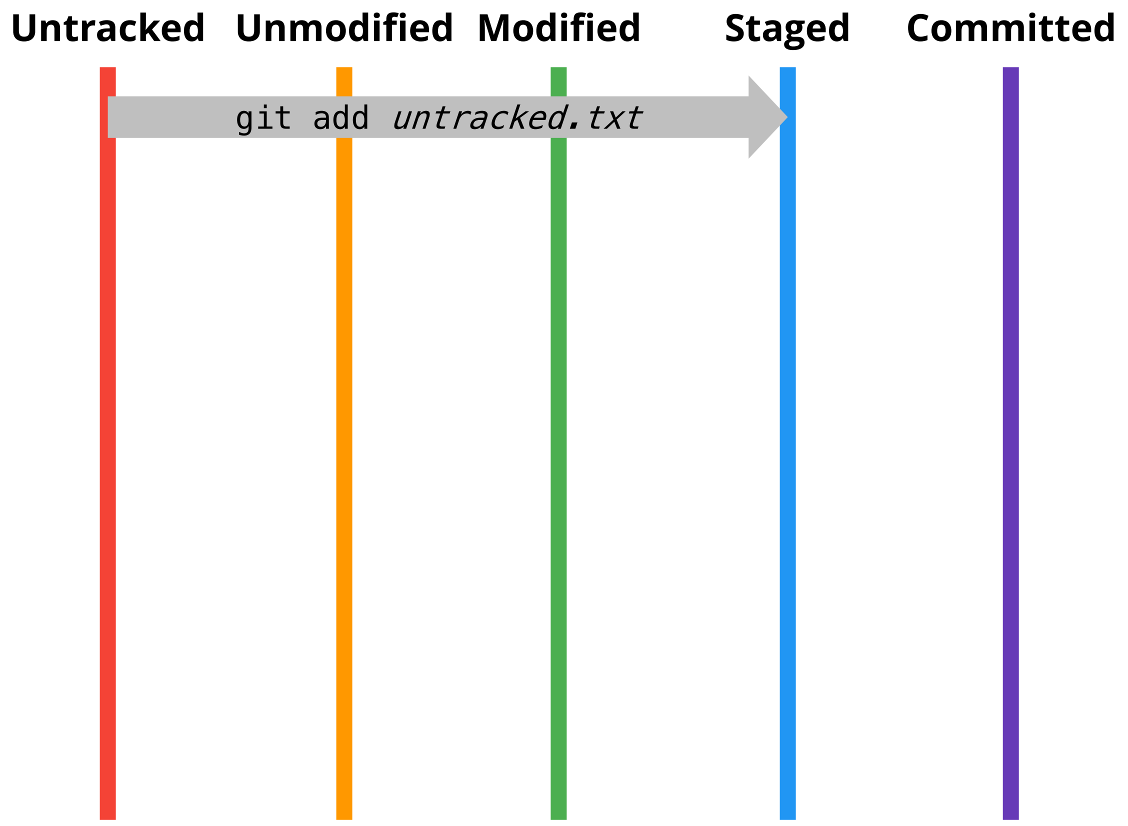
- Suppose you want to delete the file “new.txt”.

1. new.txt is in an *unmodified* state.
2. Delete new.txt (*modified*).
3. Stage new.txt by “`git add new.txt`” (*staged*).
4. Commit the file by “`git commit`” (*untracked*).

- unmodified → modified → untracked
- The file is removed from both the working directory and the local repository.

Git Commands

- git init
- git clone
- git add
  - .gitignore
- git commit
- git reset
- git rm
- git status



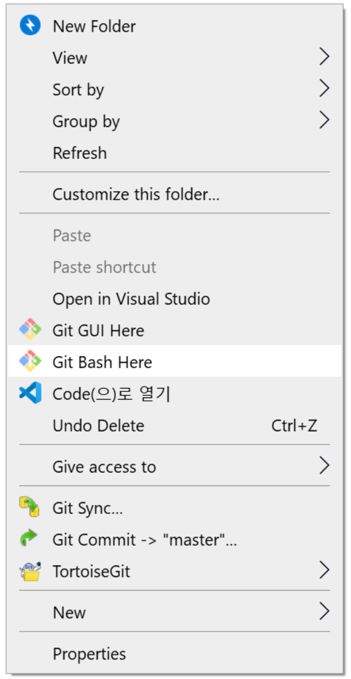
Prerequisites

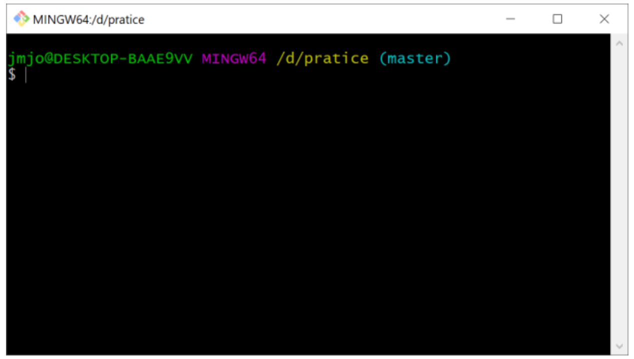
- Install Visual Studio Code
- Install Git
  - Windows: https://git-scm.com/
    - If you already have Git, update it to the latest version.
    - `git update-git-for-windows`
  - Mac
    - Type `git --version` on your terminal. It will prompt you to install it.

- Check out the version by `git --version`
- My Git version is `git version 2.37.2.windows`

Prerequisites

- Open Git command line
  - Windows
    - Go to your Git repo (or an empty directory that you want to initialize as Git repo).
    - Right-click in the Explorer.
    - Click on “Git Bash Here” in the context menu.
  - Mac
    - Open the Terminal app
    - Go to your Git repo (or an empty directory that you want to initialize as Git repo) using cd.
    - Type commands on the terminal.





```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

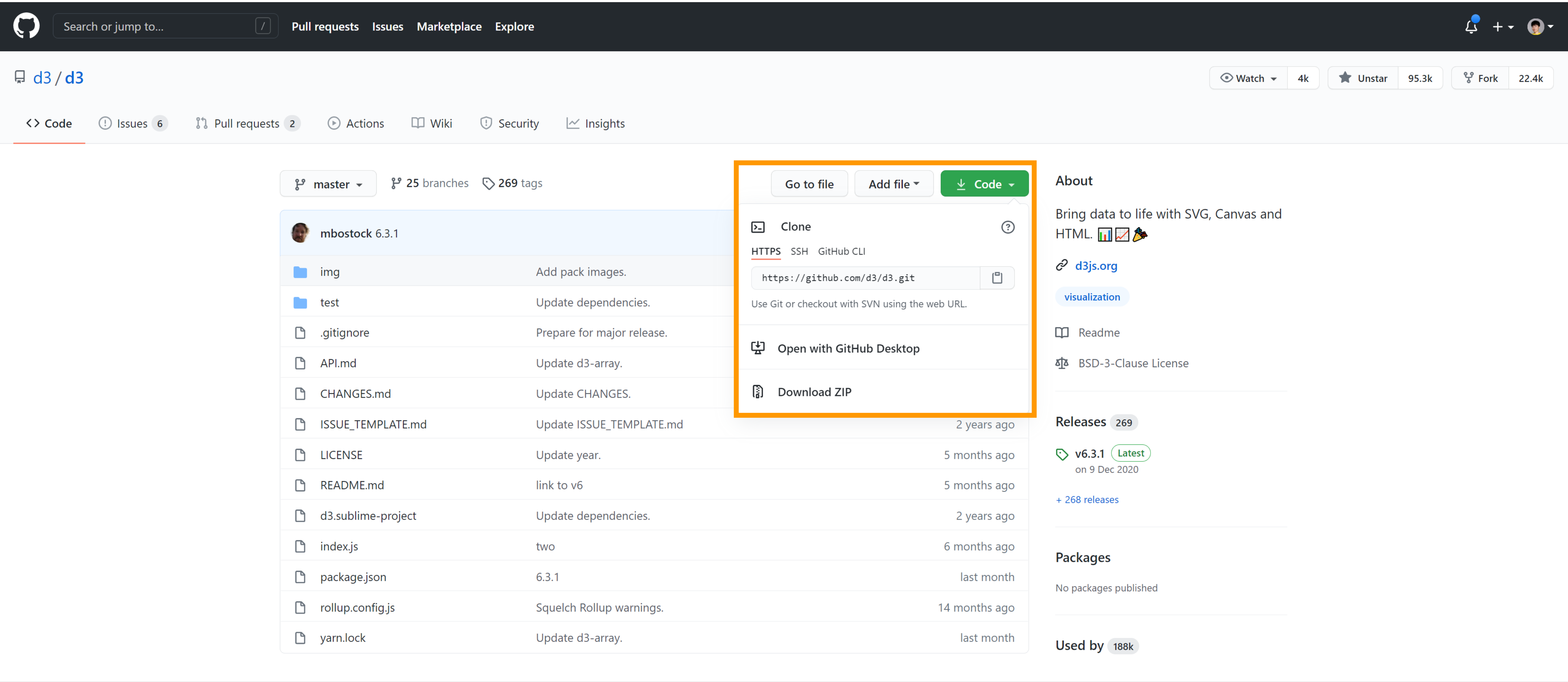
Getting a Git Repository

- There are two ways to get a Git Repository:

- Initialize one from scratch:
  - Go to an empty directory on your terminal (perhaps, you are already in it).
  - `cd path-to-directory`
  - `git init`

- You can clone an existing repository.
  - `git clone <url to an existing git repo>`

Cloning a Git Repository



```
https://github.com/d3/d3.git
```

Use Git or checkout with SVN using the web URL.

mbostock 6.3.1

img	Add pack images.	
test	Update dependencies.	
.gitignore	Prepare for major release.	
API.md	Update d3-array.	
CHANGES.md	Update CHANGES.	
ISSUE_TEMPLATE.md	Update ISSUE_TEMPLATE.md	2 years ago
LICENSE	Update year.	5 months ago
README.md	link to v6	5 months ago
d3.sublime-project	Update dependencies.	2 years ago
index.js	two	6 months ago
package.json	6.3.1	last month
rollup.config.js	Squelch Rollup warnings.	14 months ago
yarn.lock	Update d3-array.	last month

About

Bring data to life with SVG, Canvas and HTML. 📊📈🎉

d3js.org

visualization

Readme

BSD-3-Clause License

Releases 269

v6.3.1 Latest on 9 Dec 2020

+ 268 releases

Packages

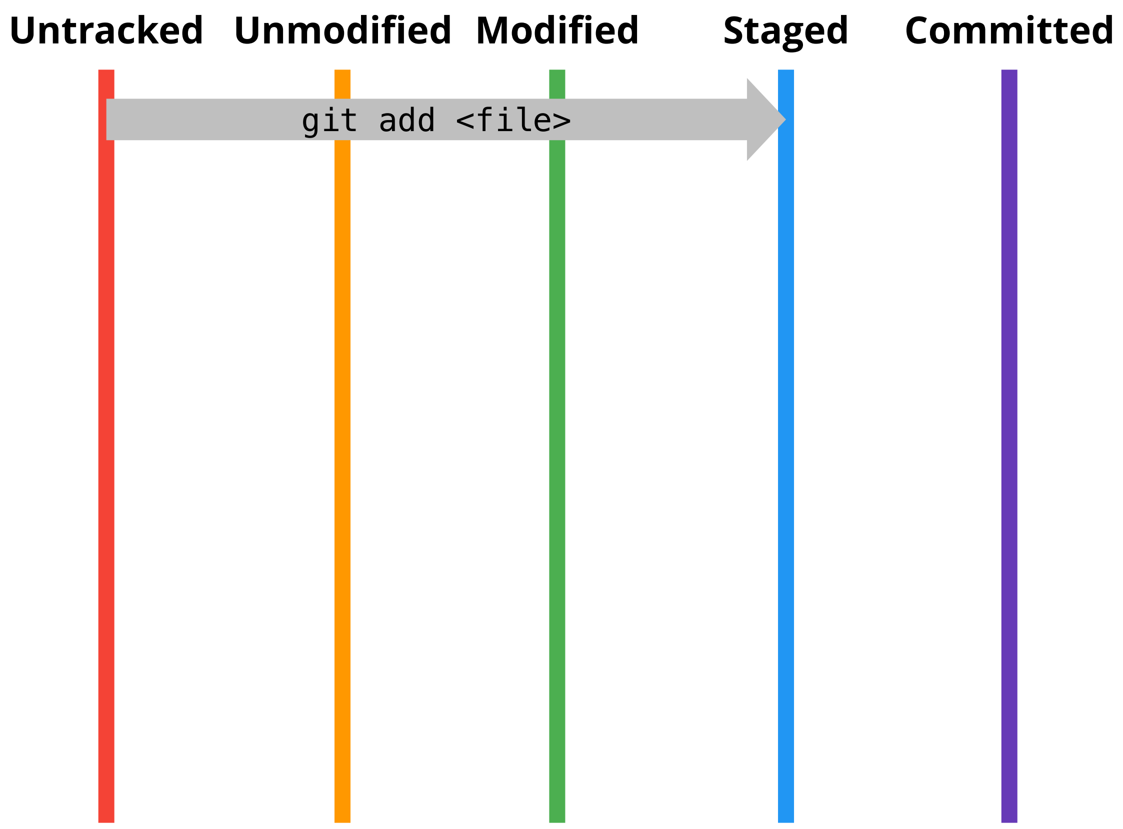
No packages published

Used by 188k

Add a New File

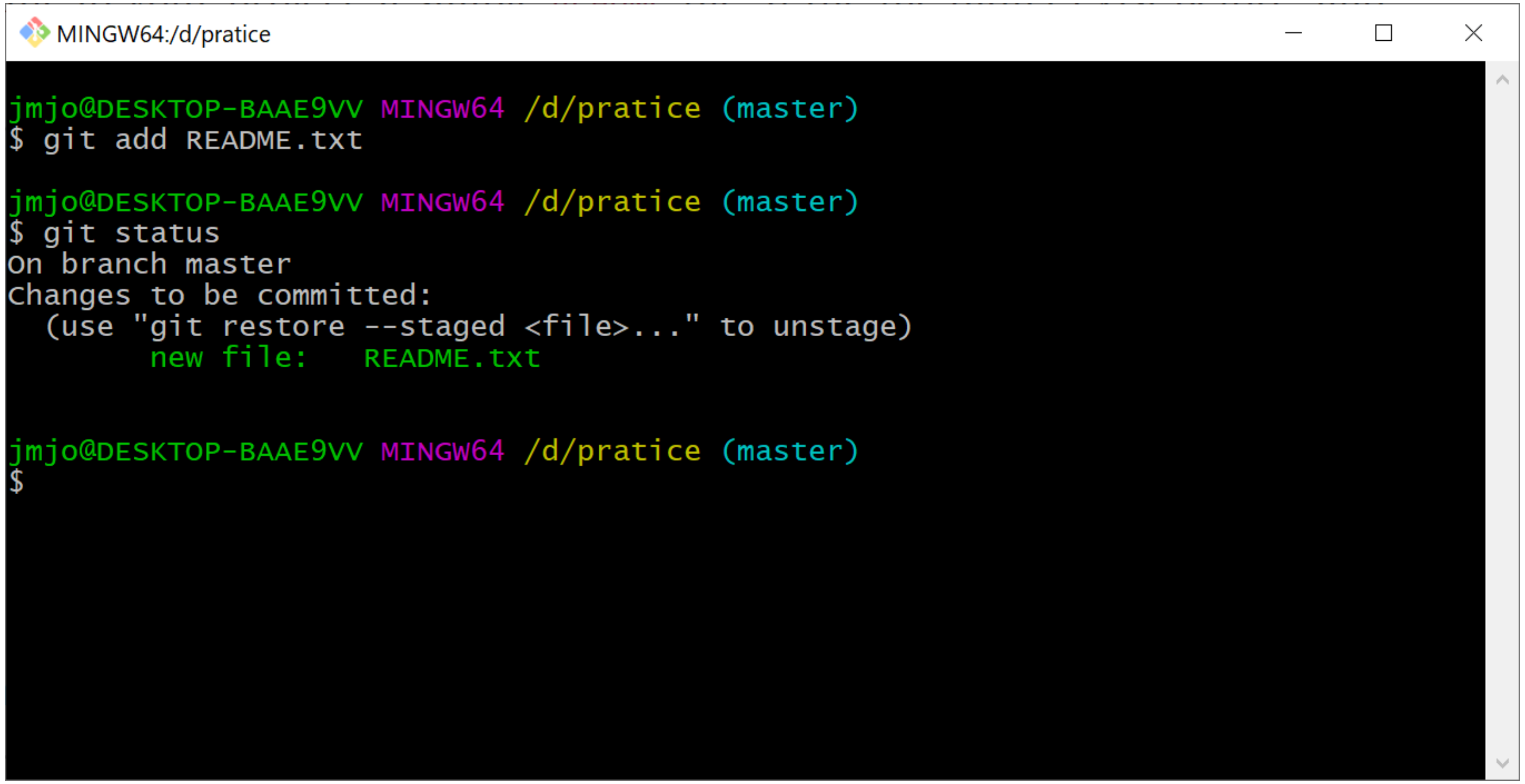
- Let's add README.txt to the repo.

1. Create README.txt with your favorite text editor.
2. Write "Hello, World" in README.txt
3. `git add README.txt`



Check the States

- Check the state of README.txt by typing `git status`.

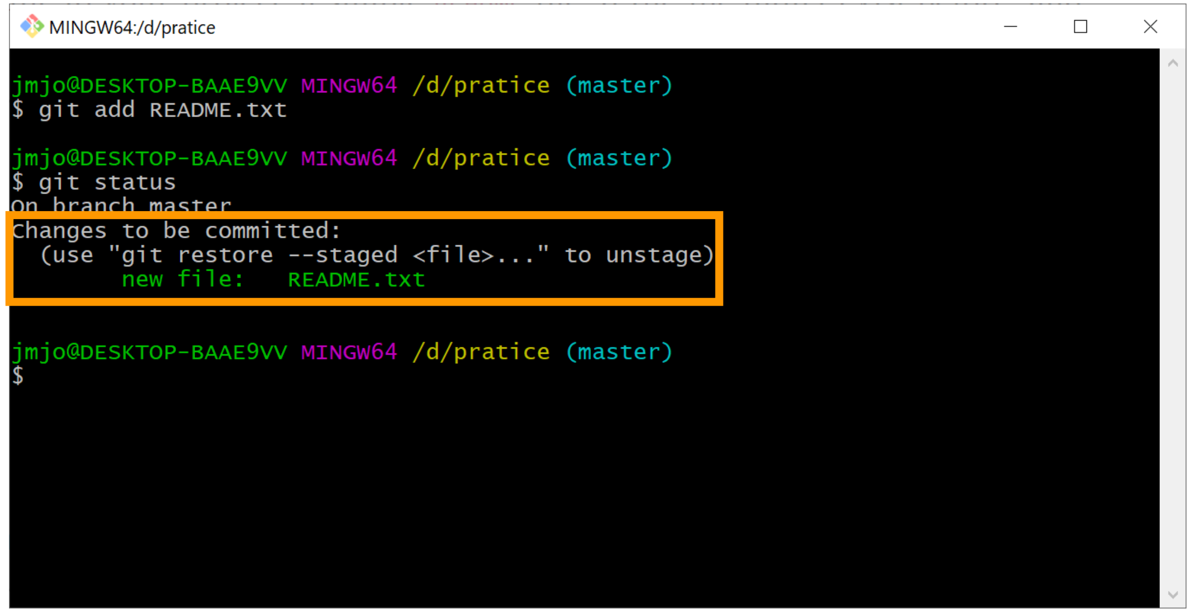


MINGW64:/d/pratice

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git add README.txt

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.txt


jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

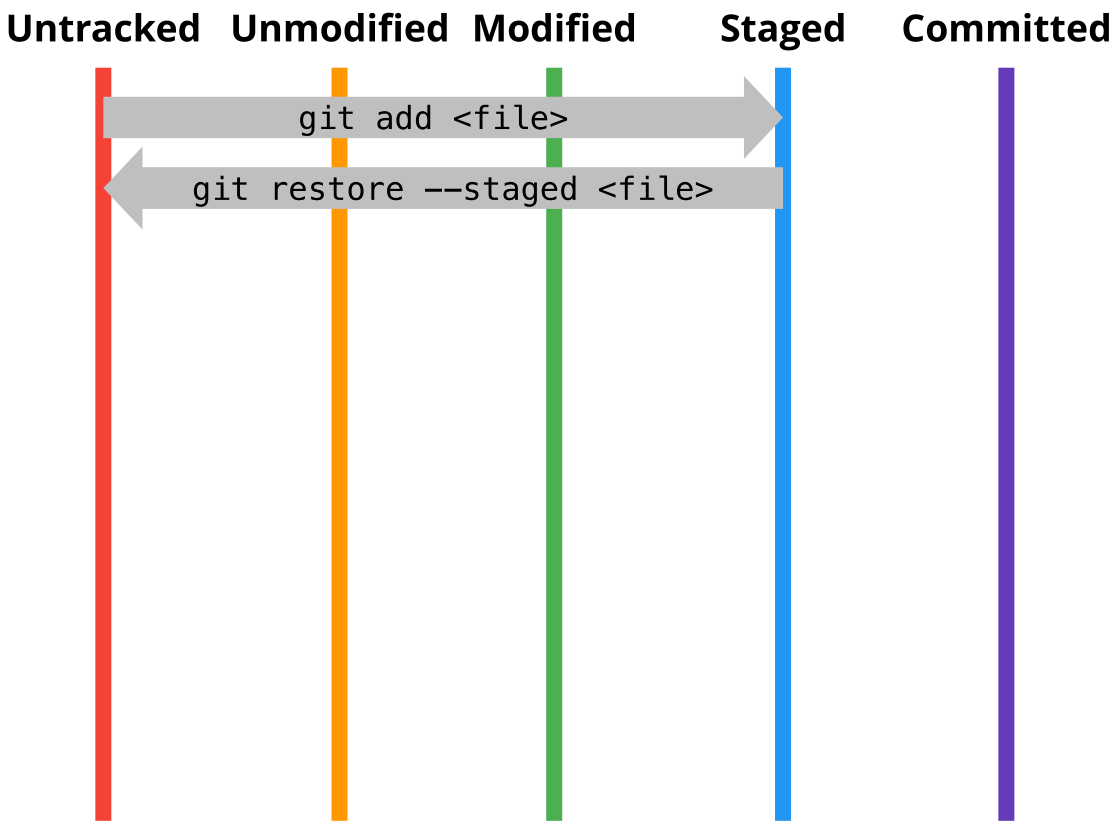
Unstage a File

- `git restore --staged README.txt` will unstage the file.
- You don't have to remember this command. There is a hint.



MINGW64:/d/pratice

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git add README.txt

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.txt

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```



Add the File Again

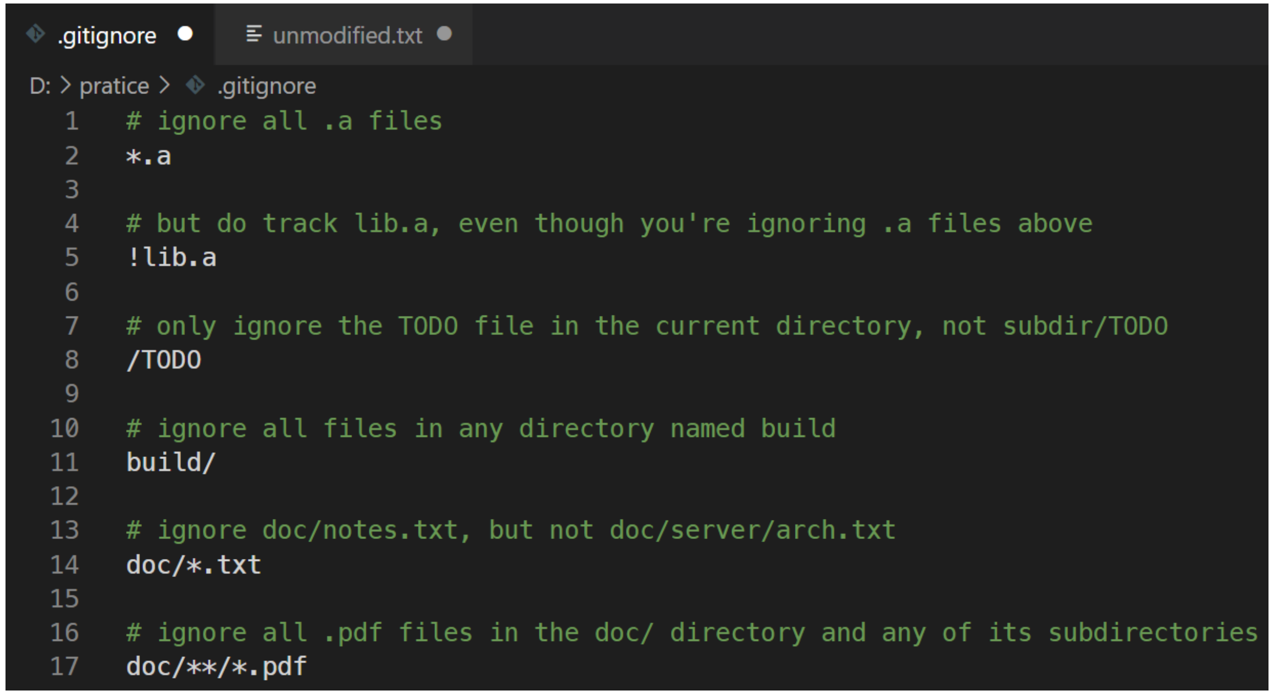
- Let’s add README.txt again, but this time type `git add .`

- This command adds **all untracked or modified** files to the staging area.
  - Useful when there are too many files to stage.

- However, with this command, files that should be excluded from VC can be staged mistakenly.
  - Very big files or datasets
  - Images
  - Executable or binary files created by compilers/interpreters
  - Private files (passwords/certificate)

.gitignore

- You'll have a class of files that you don't want Git to automatically add or even show you as being untracked.
- Add the names of those files to a special text file ".gitignore".



```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

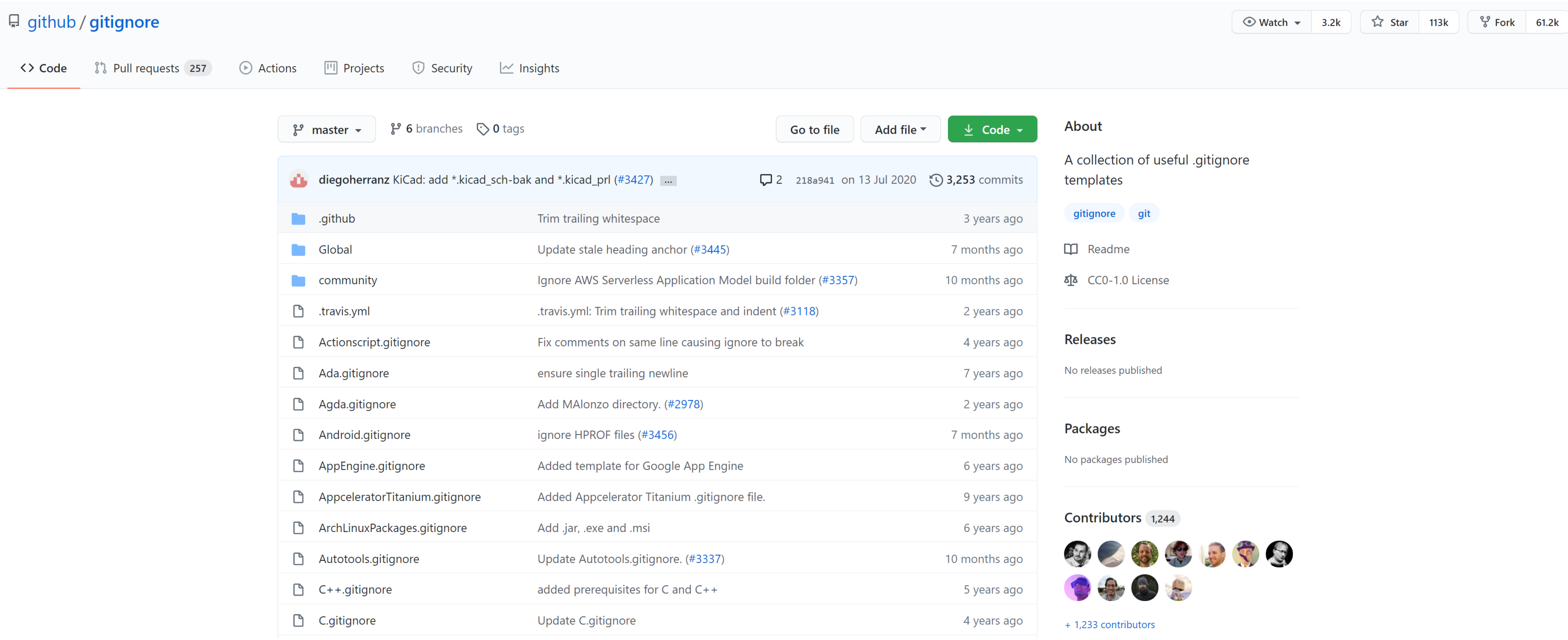
.gitignore

- Different development environments generate different temporary files that should be ignored.

- The gitignore repository has a lot of template .gitignore files for different environments.
  - https://github.com/github/gitignore

- It is always a good practice to copy and paste an appropriate .gitignore file to your repo just after you initialize it.

.gitignore



github / gitignore

Watch 3.2k | Star 113k | Fork 61.2k

Code | Pull requests 257 | Actions | Projects | Security | Insights

master | 6 branches | 0 tags | Go to file | Add file | Code

diegoherranz KiCad: add *.kicad_sch-bak and *.kicad_prl (#3427) ... 2 218a941 on 13 Jul 2020 3,253 commits

.github	Trim trailing whitespace	3 years ago
Global	Update stale heading anchor (#3445)	7 months ago
community	Ignore AWS Serverless Application Model build folder (#3357)	10 months ago
.travis.yml	.travis.yml: Trim trailing whitespace and indent (#3118)	2 years ago
Actionscript.gitignore	Fix comments on same line causing ignore to break	4 years ago
Ada.gitignore	ensure single trailing newline	7 years ago
Agda.gitignore	Add MAlonzo directory. (#2978)	2 years ago
Android.gitignore	ignore HPROF files (#3456)	7 months ago
AppEngine.gitignore	Added template for Google App Engine	6 years ago
AppceleratorTitanium.gitignore	Added Appcelerator Titanium .gitignore file.	9 years ago
ArchLinuxPackages.gitignore	Add .jar, .exe and .msi	6 years ago
Autotools.gitignore	Update Autotools.gitignore. (#3337)	10 months ago
C++.gitignore	added prerequisites for C and C++	5 years ago
C.gitignore	Update C.gitignore	4 years ago

About

A collection of useful .gitignore templates

gitignore git

Readme

CC0-1.0 License

Releases

No releases published

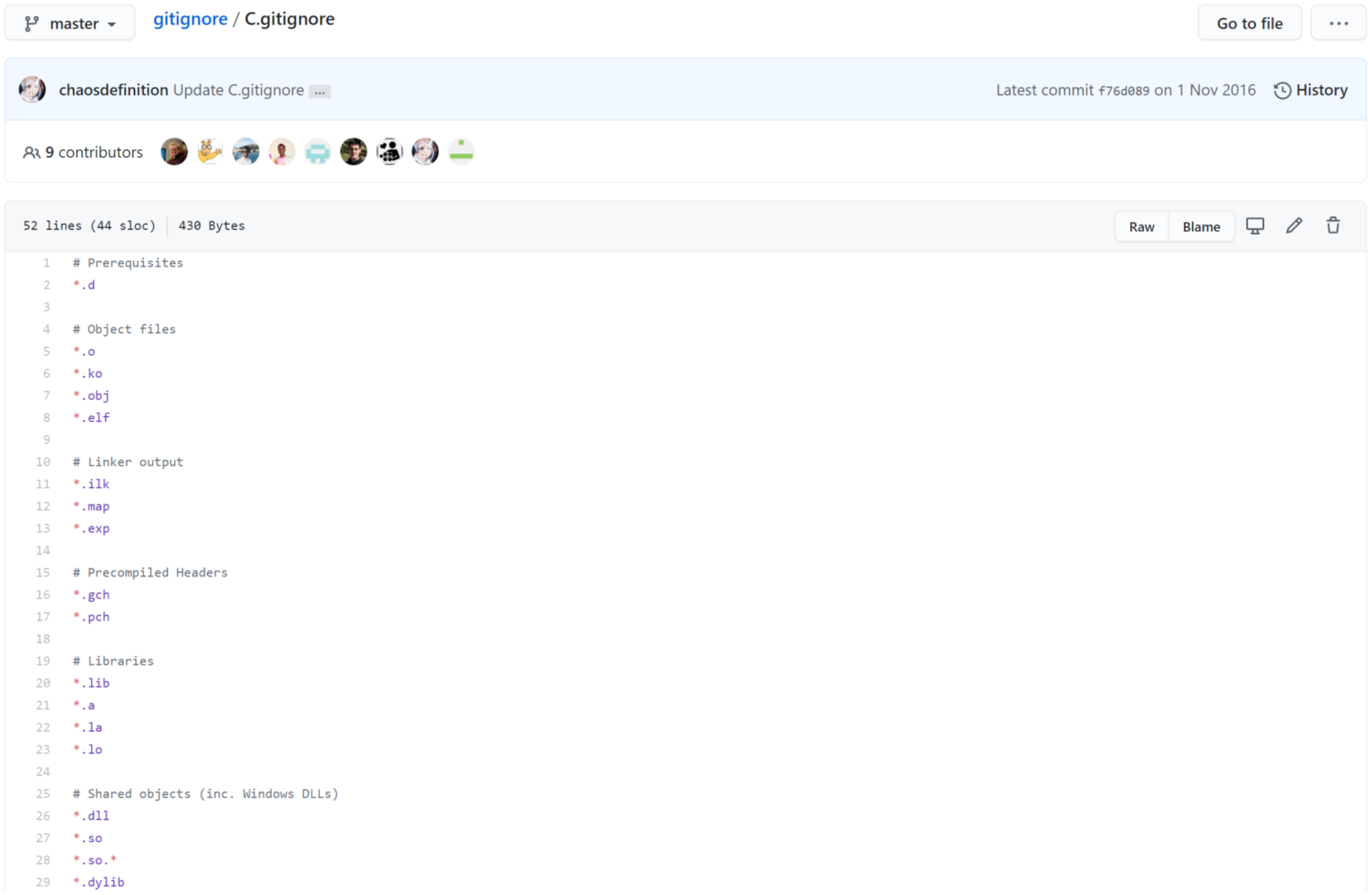
Packages

No packages published

Contributors 1,244

+ 1,233 contributors

.gitignore



gitignore / C.gitignore

chaosdefinition Update C.gitignore ... Latest commit f76d089 on 1 Nov 2016 History

9 contributors

52 lines (44 sloc) | 430 Bytes

```
# Prerequisites
*.d

# Object files
*.o
*.ko
*.obj
*.elf

# Linker output
*.ilk
*.map
*.exp

# Precompiled Headers
*.gch
*.pch

# Libraries
*.lib
*.a
*.la
*.lo

# Shared objects (inc. Windows DLLs)
*.dll
*.so
*.so.*
*.dylib
```

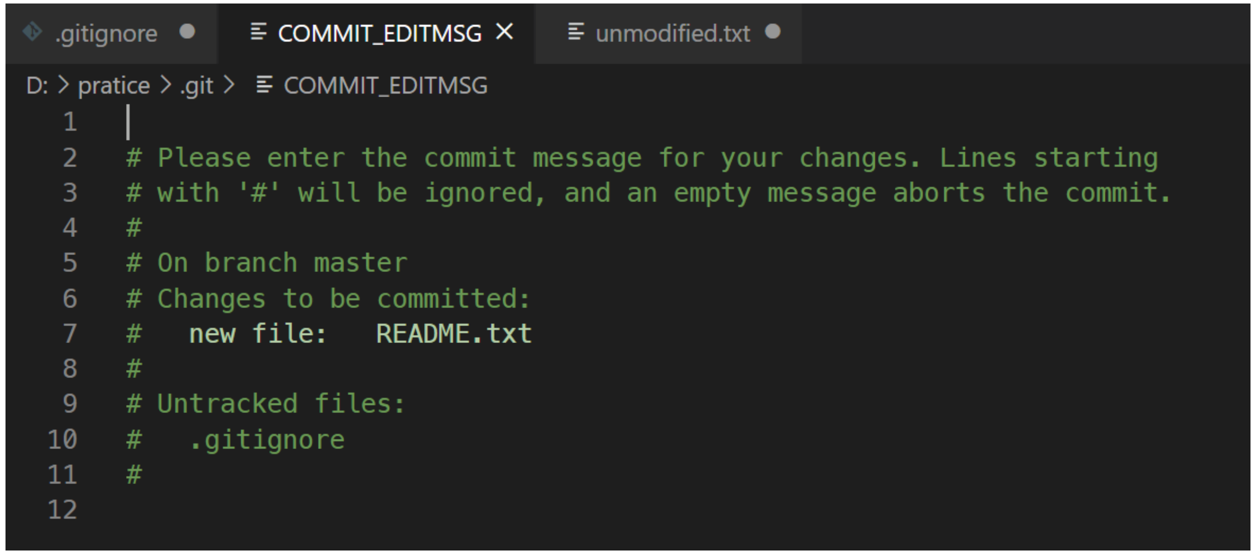
Usage of .gitignore and `git restore`

- You create a repo.
- You download one hundred CSV files as datasets.
- You write some source codes.
- `git add .`
- Oops, all CSV files are going to be uploaded with the source files!
- Add *.csv to the end of .gitignore (still CSV files are *staged*).
- `git restore ––staged .` (both CSV files and source codes become *untracked*).
- `git add .` (only source codes are *staged*)

Commit Changes

- Let’s commit the staged files by typing `git commit`

- If you have installed VS Code, VS Code will be launched for a commit message.



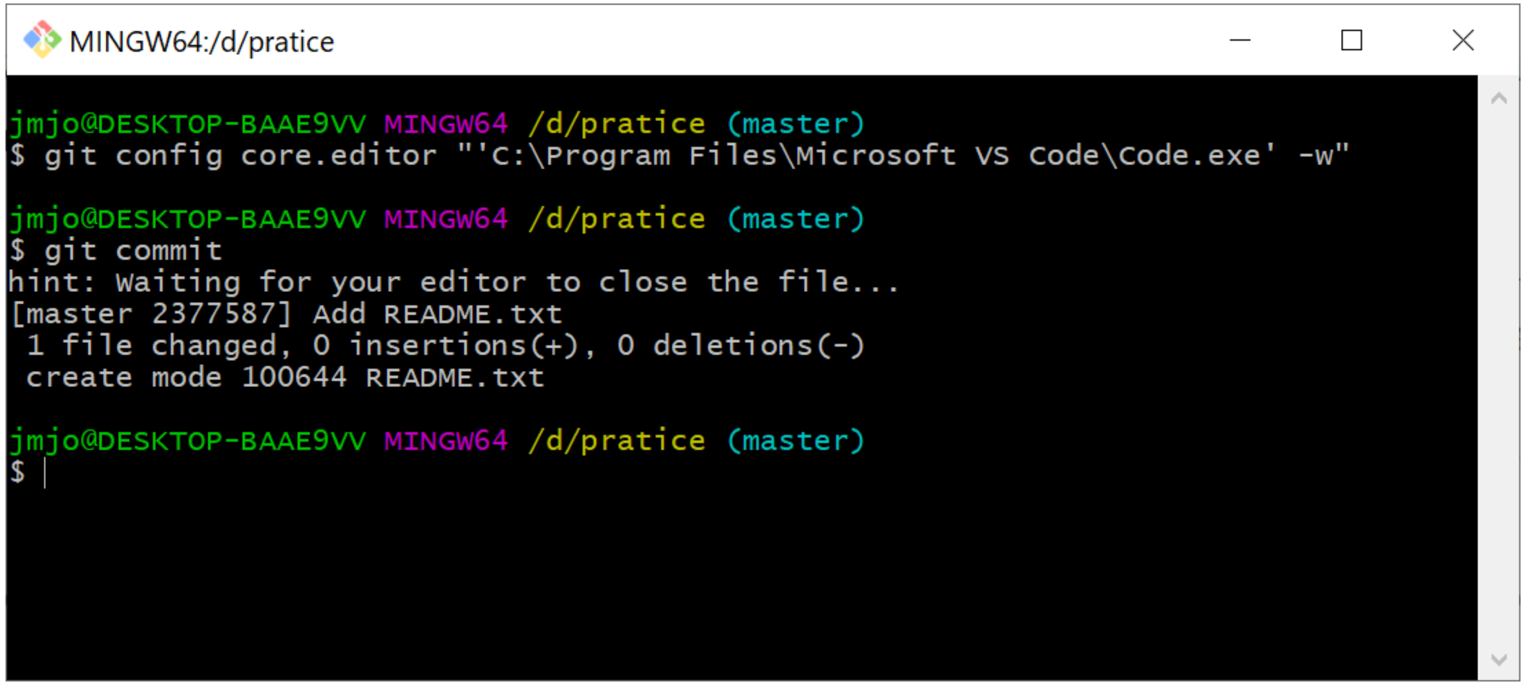
.gitignore ● COMMIT_EDITMSG × unmodified.txt ●

D: > pratice > .git > COMMIT_EDITMSG

```

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#	new file:   README.txt
#
# Untracked files:
#	.gitignore
#

```

Commit Changes

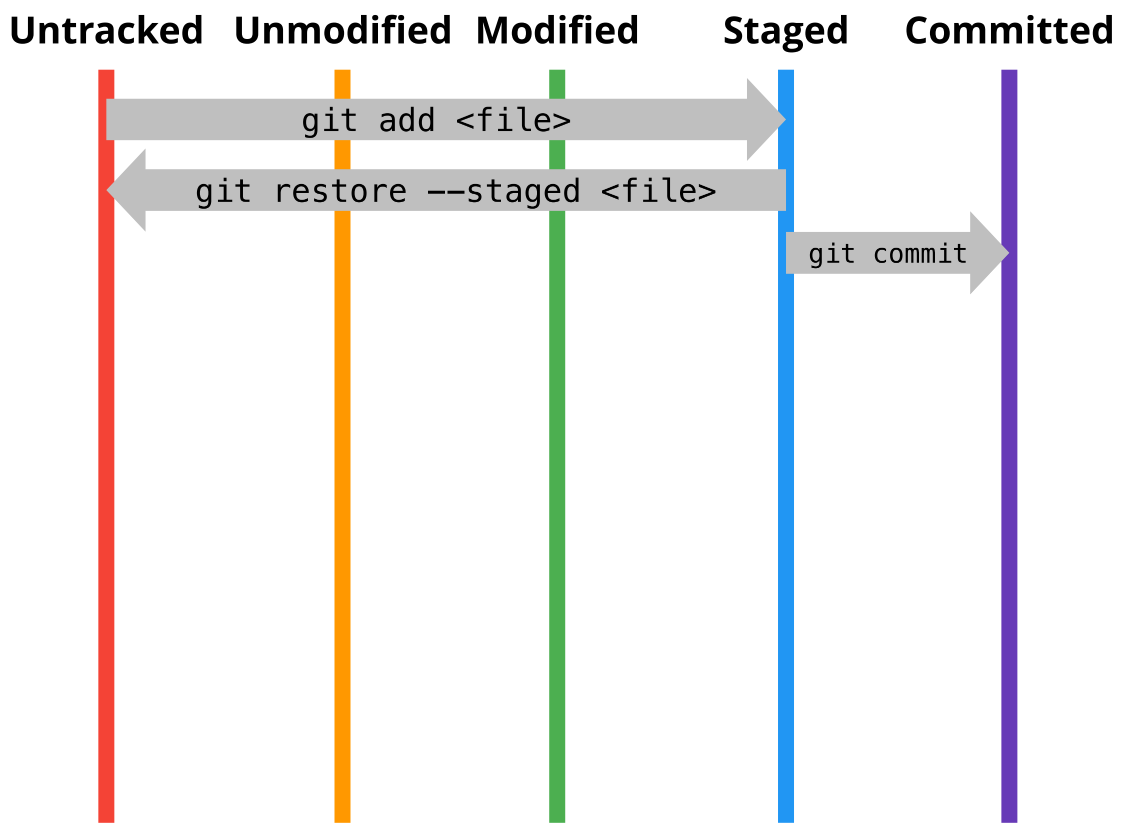
- Enter a commit message such as “Add README.txt”, **SAVE**, and close the tab.
- If VS Code is not launched, try the command on the first line (replace the path to Code.exe)


MINGW64:/d/pratice

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git config core.editor "'C:\Program Files\Microsoft VS Code\Code.exe' -w"

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git commit
hint: Waiting for your editor to close the file...
[master 2377587] Add README.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.txt

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ 
```

Commit Changes

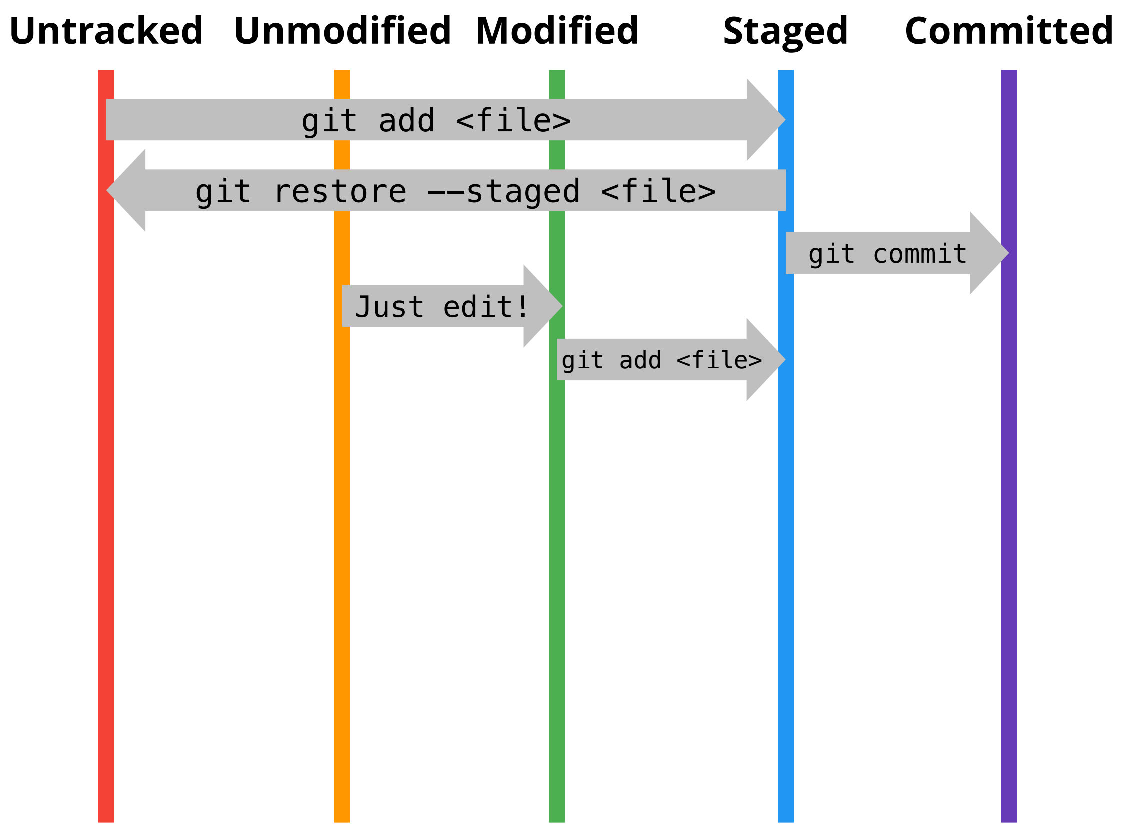
- We just committed the changes made to README.txt to our local Git repo.
- The state of README.txt changed to *committed* from *staged*.
- You created a new version, and from its point of view, README.txt is *unmodified*.
  - *unmodified == committed*



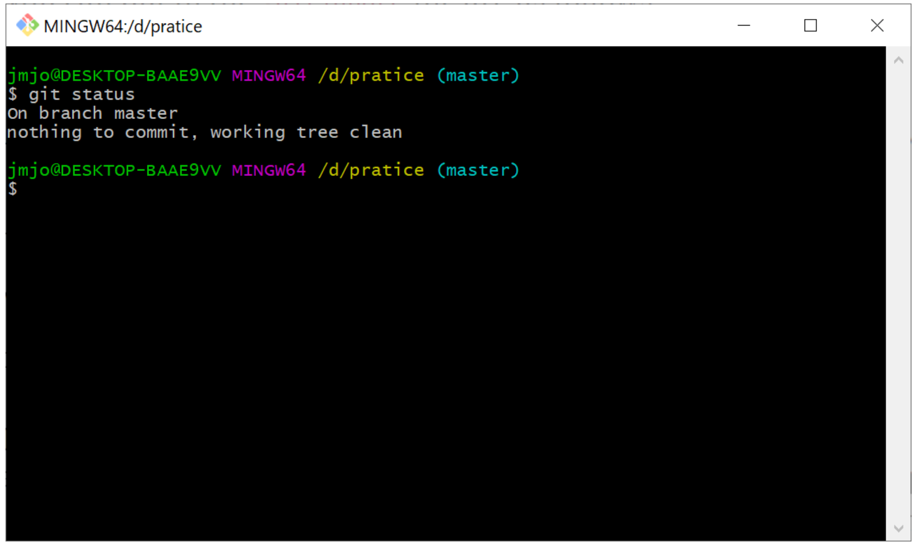
Modify a File

- Now, README.txt is in an *unmodified* state.

1. Modify README.txt.
2. `git status`
3. `git add README.txt`
4. `git status`
5. `git commit`
6. `git status`

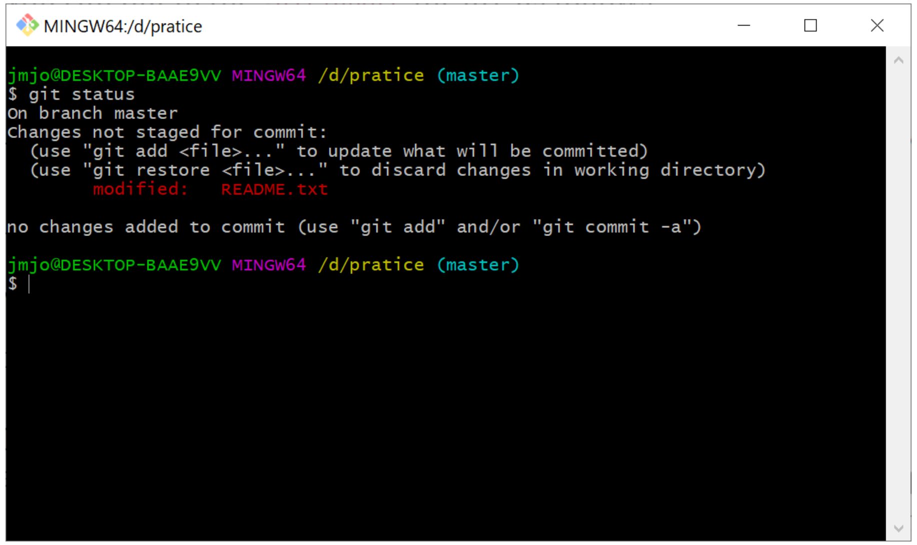


Modify a File



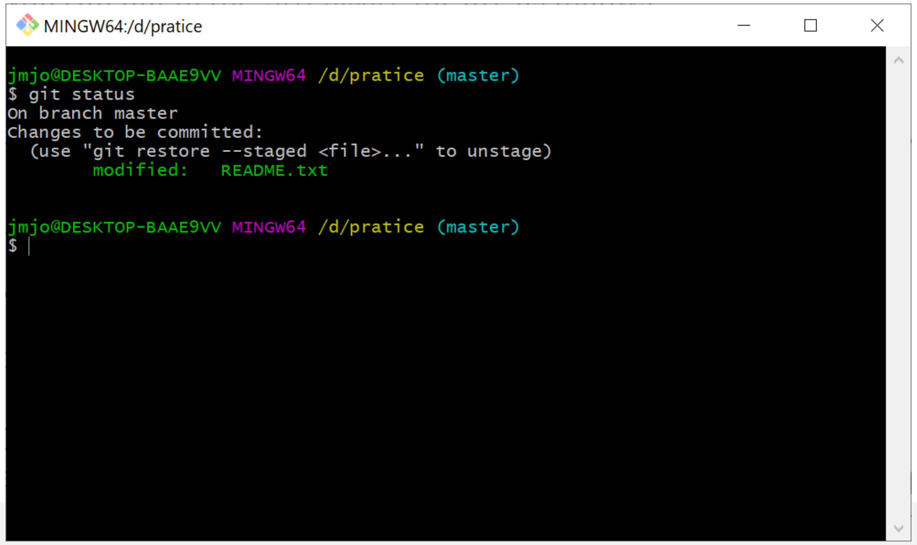
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
nothing to commit, working tree clean

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

modify



```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.txt

no changes added to commit (use "git add" and/or "git commit -a")

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

`git add .`



```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.txt


jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

`git commit`

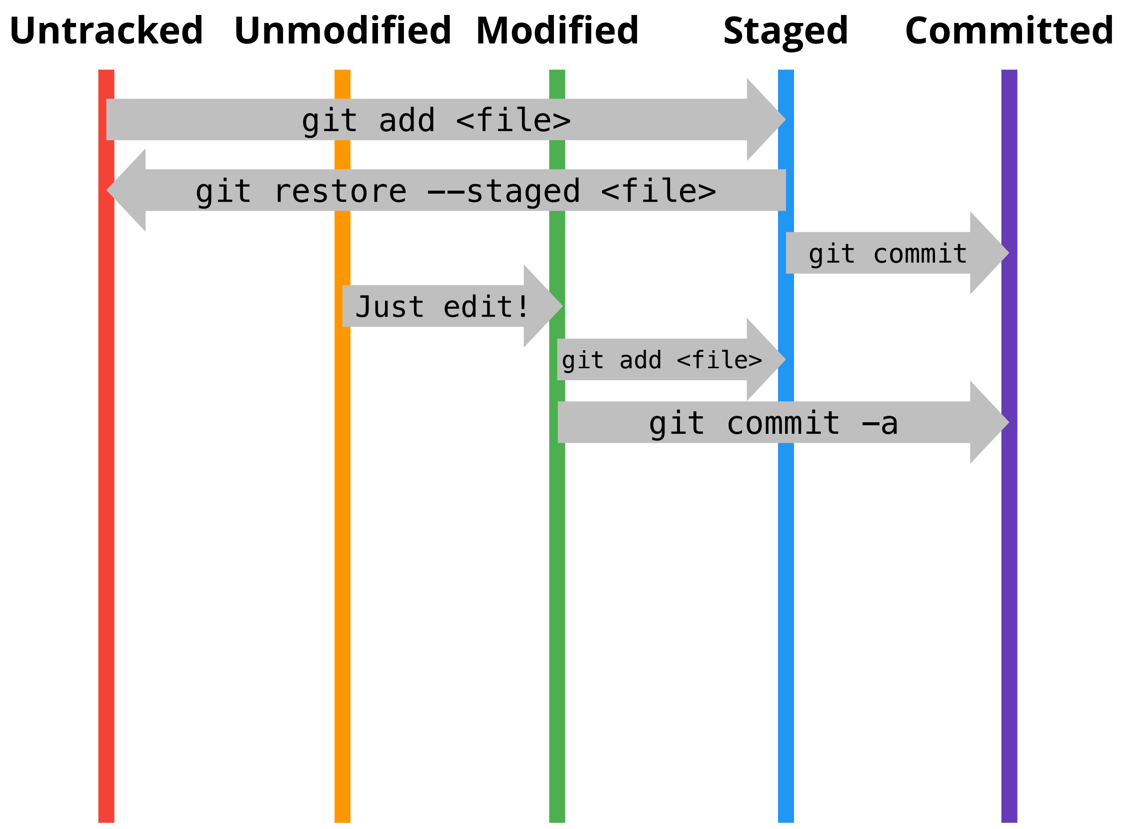
Modify a Modified File

- `git add` adds the snapshot of files to the index.
- You must add a file again to reflect the changes made after adding the file to the index.

- Edit a.txt (first)
- Add a.txt
- Edit a.txt (second)
- Add a.txt (again!)

Add + Commit at Once

- To commit, you must add a file.
- If you want to commit all the modified files, use `git commit –a`
  - It skips staging.
  - `git commit –a == git add . and git commit`
- For untracked files, you must add them first.



Remove a File

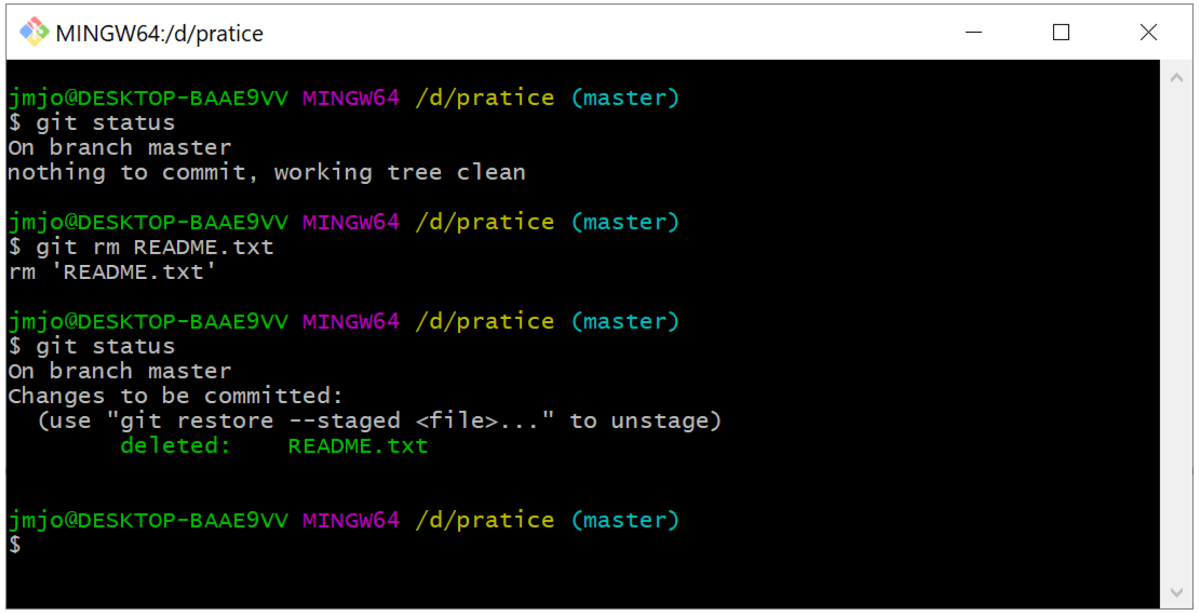
- Removing a file from a repo can mean two things:

- Remove the file from the disk (and from the repo).
  - The file will be gone completely.
  1. Just remove the file (e.g., with the Del key), `git add <file>`, and `git commit`.
  2. `git rm <file>` and `git commit`

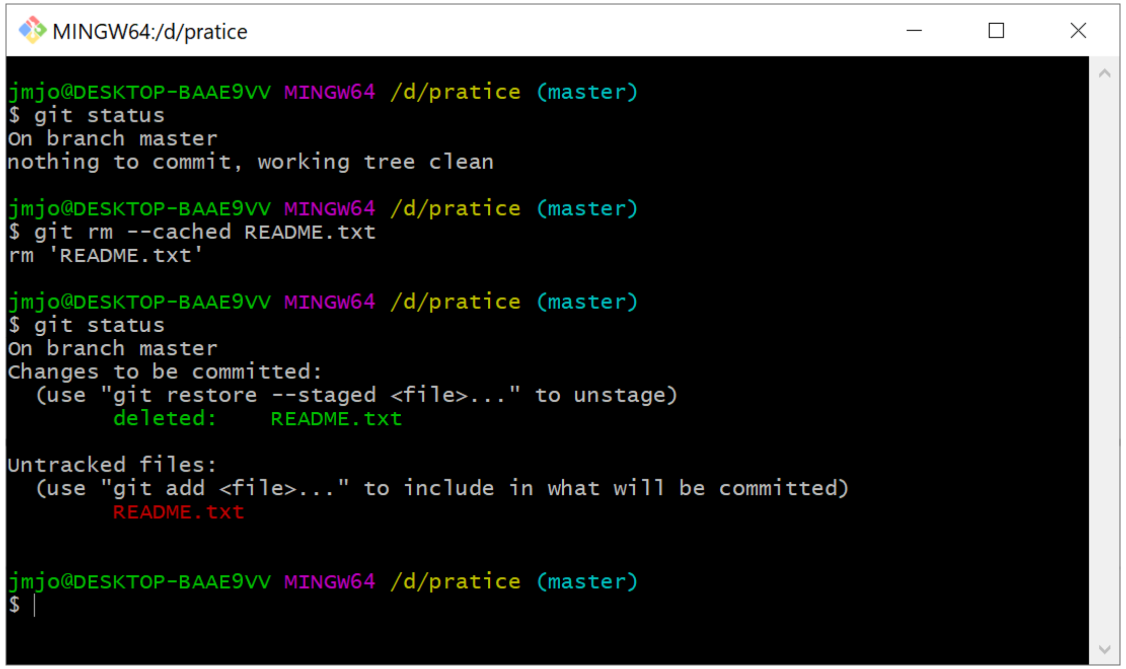
- Remove the file from the repo but keep it locally.
  - The file will be *untracked*.
  - The changes made to the file will be kept.
  - `git rm ––cached <file>` and `git commit`

Remove a File

- `git rm vs git rm --cached`
- `git rm <path> == rm <path> && git add <path>`



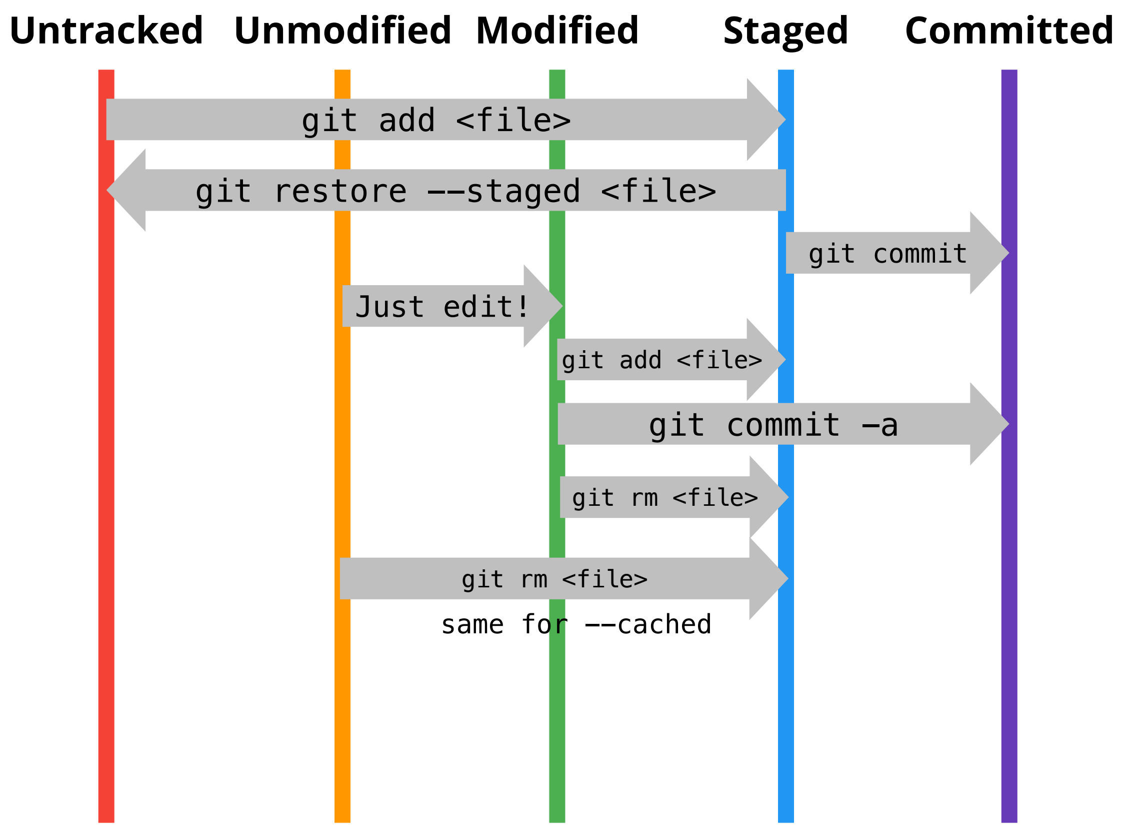
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
nothing to commit, working tree clean

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git rm README.txt
rm 'README.txt'

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    README.txt


jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```



```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
nothing to commit, working tree clean

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git rm --cached README.txt
rm 'README.txt'

jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    README.txt

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


jmjo@DESKTOP-BAAE9VV MINGW64 /d/pratice (master)
$
```

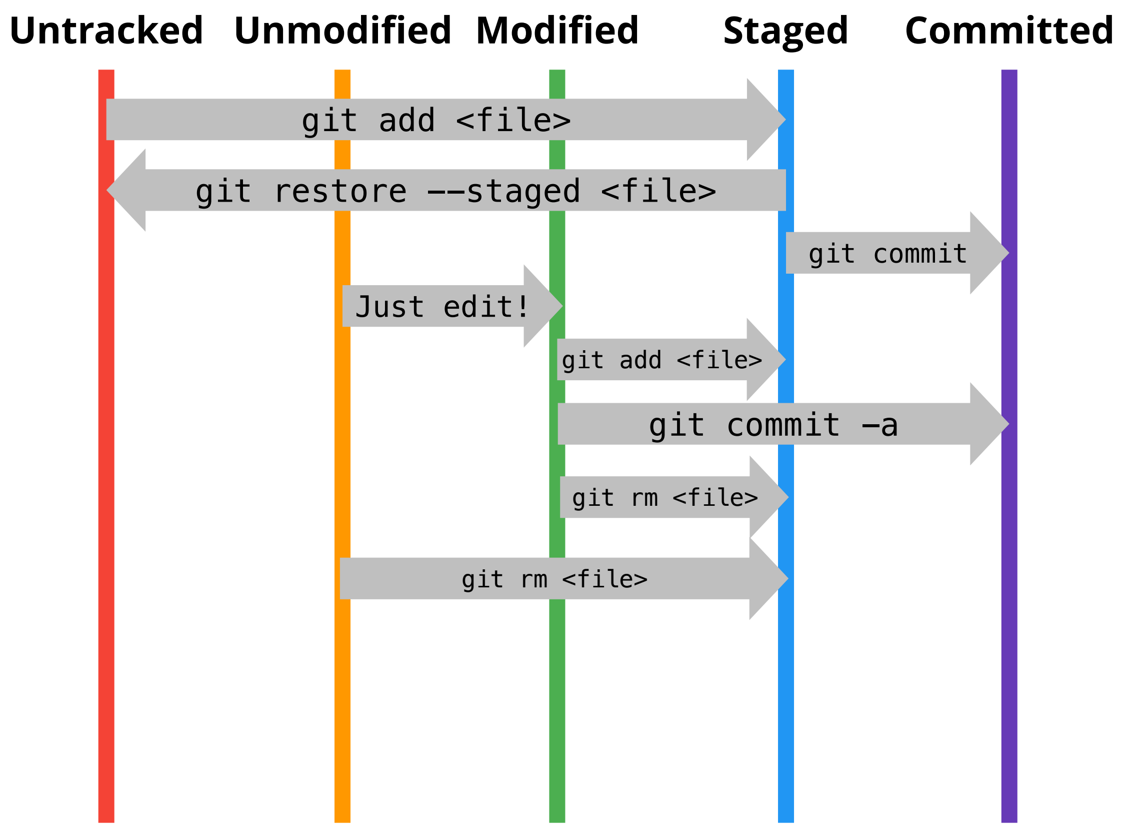
Remove a File

- Note that `git rm` does not remove a file from the index immediately.
  - You need to commit after executing `git rm`.
- It writes a commit like "this file will be removed."
- Some students may be confused.
  - Why does `git rm --cached` change the state from *modified* to *staged*, not from *modified* to *untracked*?
- Untracked when committed



Remove a File

- `git rm` removes a file and stages it for removal.
- You can approve the staged changes by `git commit`.
- When addition/modification is approved, the state becomes *unmodified*.
- When removal is approved, the state becomes *untracked*.



Undoing Things

- Today, we will learn three scenarios of undoing:
  - Editing a commit
  - Unstaging a staged file
  - Unmodifying a modified file

- Note that these should be done before you commit.
  - We will learn how to undo committed things later.

Editing a Commit

- Scenario:
  - You staged files and committed them.
  - After committing, you found you forgot to add some files.

- `git add <forgotten files>`
- `git commit --amend`

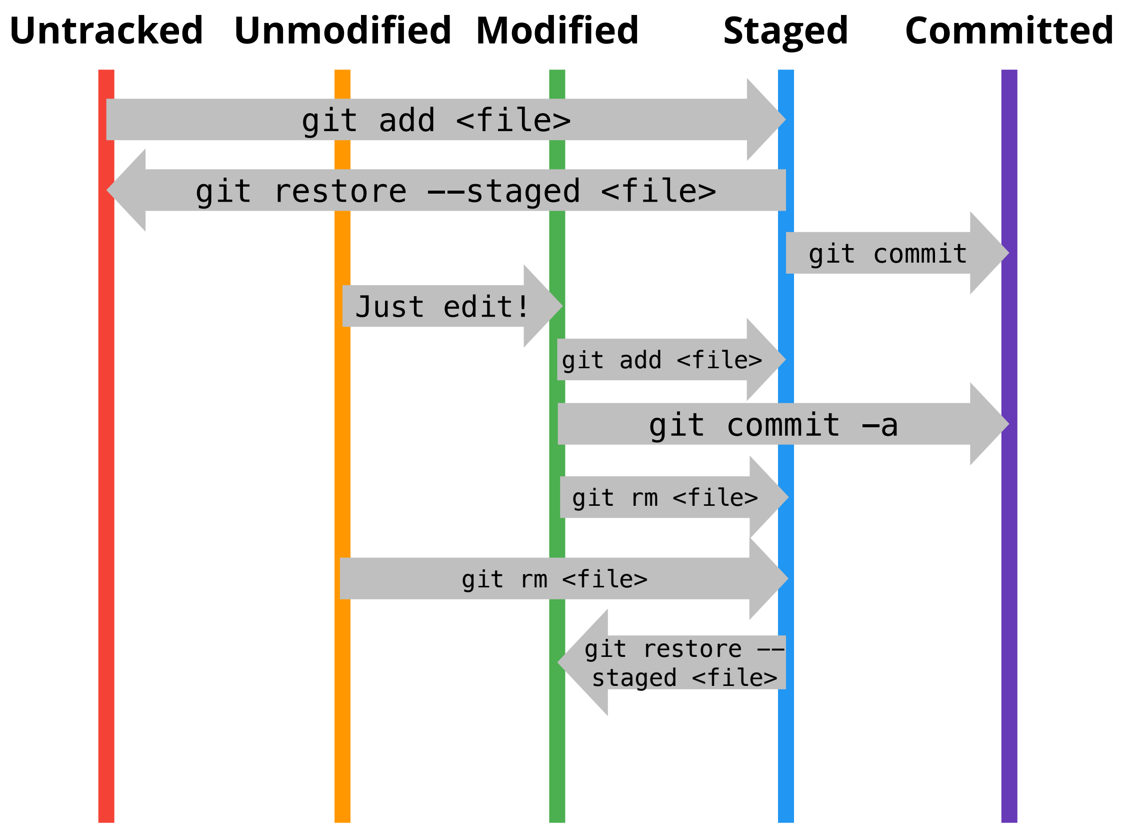
- The commit-message editor will pop up with the previous commit message.
- This will end up with a single commit.
- This is only possible for the last commit you made.

Unstaging a Staged File

- Scenario:
  - You staged files to commit using `git add`.
  - You found some files that should not be included in the commit.

- We already know the solution.
- `git restore --staged <file>`

- Also possible through `git reset HEAD <file>` although not recommended.



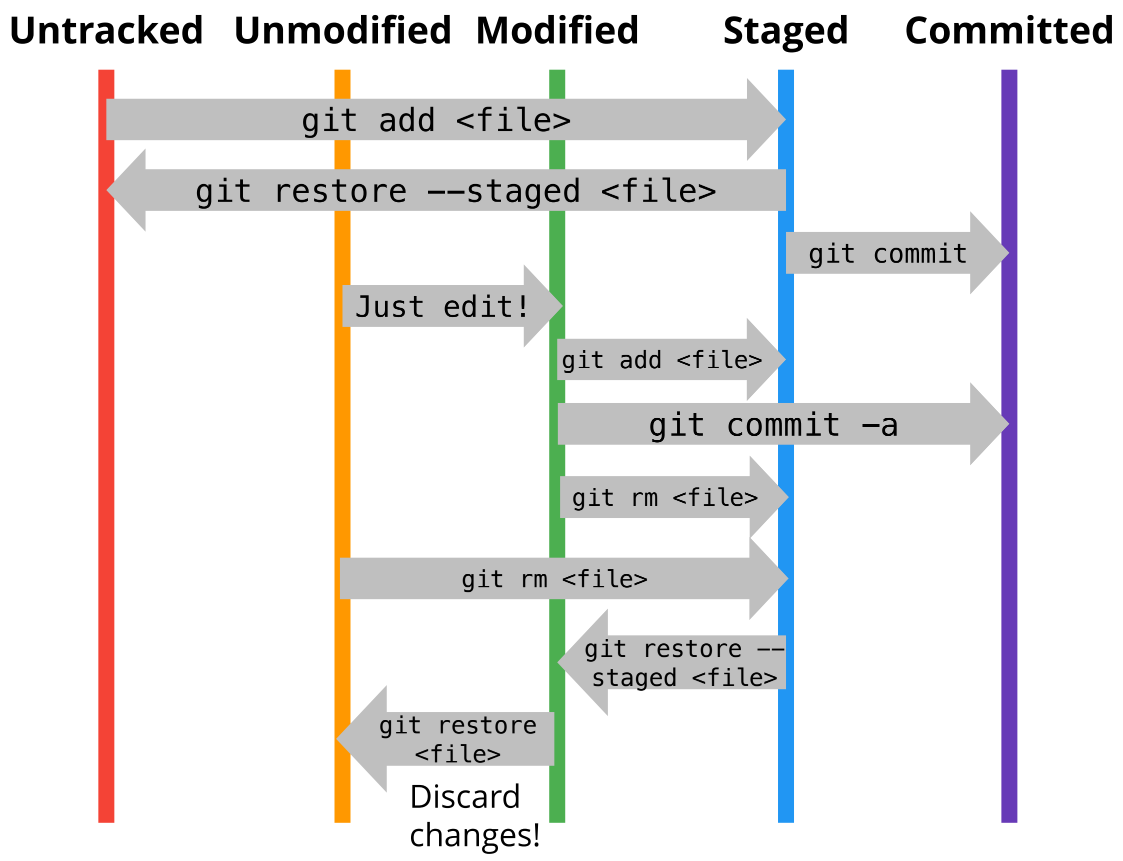
Unmodifying a Modified File

- Scenario:
  - You modified a file, but the program doesn't work anymore. Something went wrong.
  - You want to reset the file to its ***last committed*** state.

- `git restore <file>`

- **Changes will be discarded.**

- Also possible through `git checkout -- <file>` although not recommended.



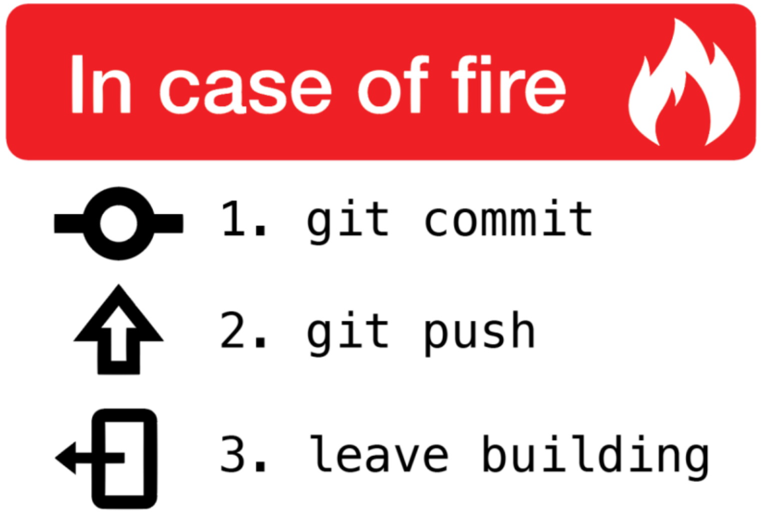
Quiz

- `git restore --staged <file>`

- `git rm --cached <file>`

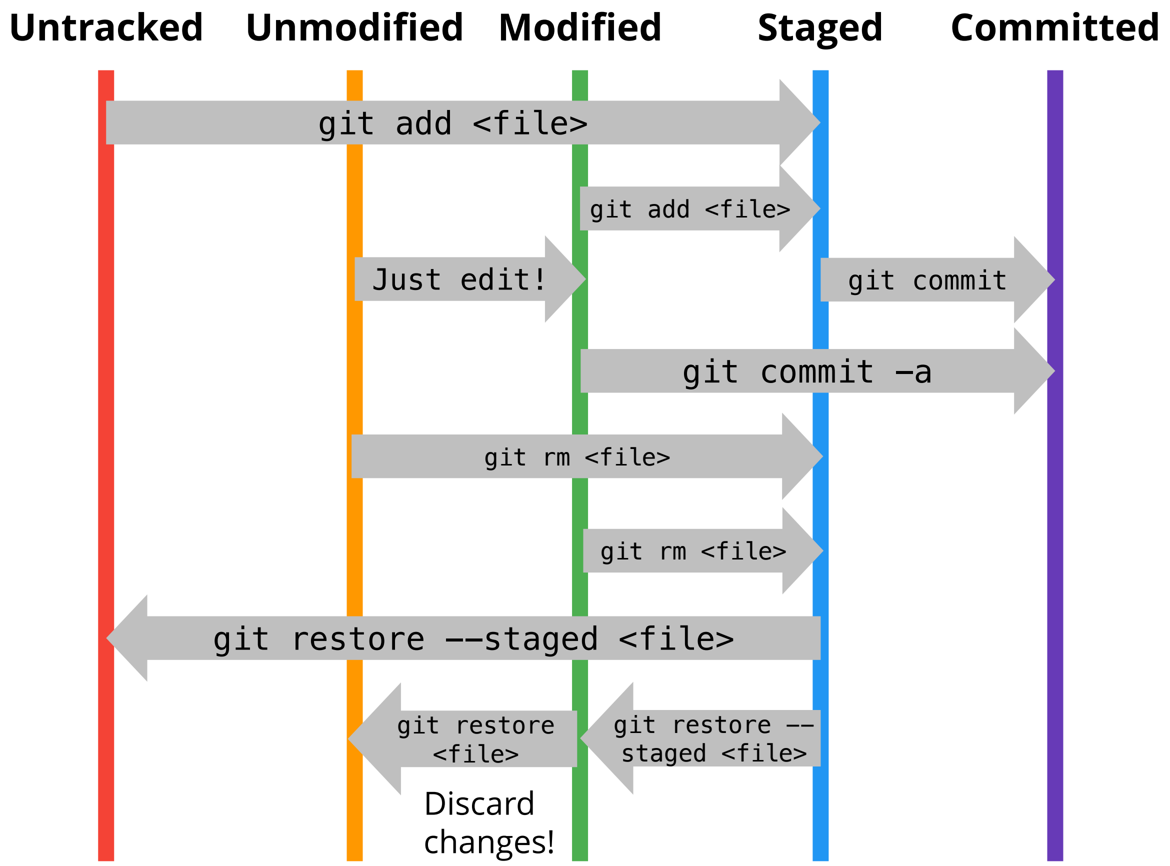
Summary: Git Basics

- Today, we learned basic commands of Git.
  - add, status, rm, commit, restore
  - .gitignore
- Note that these commands are done locally.
  - Changes are visible only to your local machine.
  - We did not even use a remote Git service like GitHub.



Summary: Git Basics

- Reordered for readability


Untracked
Unmodified
Modified
Staged
Committed
git add <file>
git add <file>
Just edit!
git commit
git commit -a
git rm <file>
git rm <file>
git restore --staged <file>
git restore <file>
git restore --staged <file>
Discard changes!