

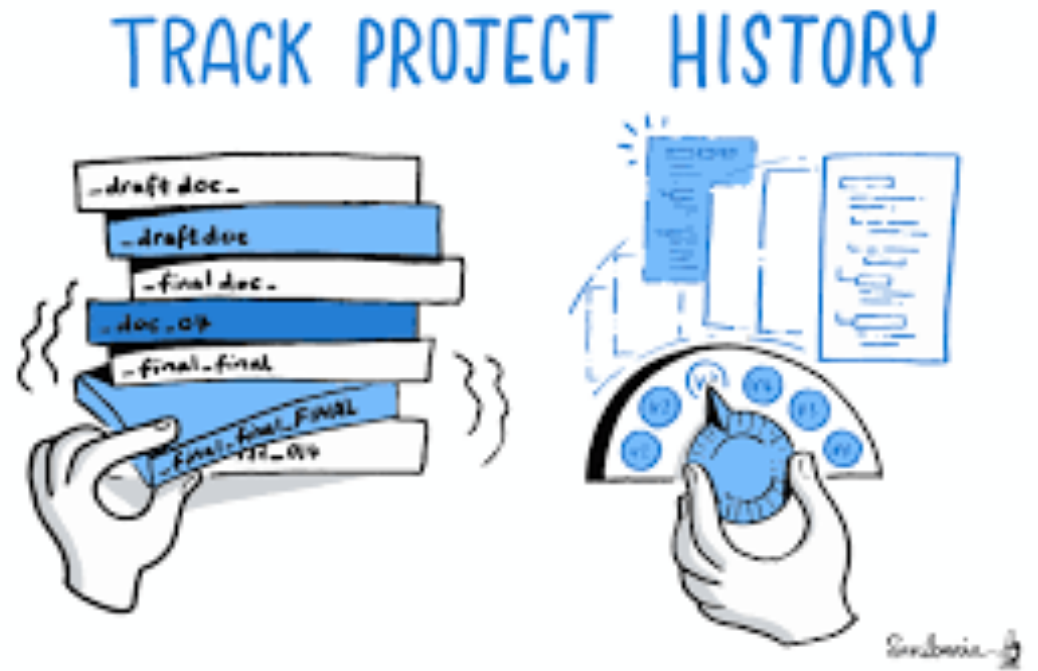
# INTRO TO GIT AND GITHUB

Selin Group Meeting  
24 February 2023  
Emmie Le Roy



# Version control systems

- Track the history of changes to a set of files and its contents
- Facilitate collaboration
- Maintains meta-data (author, timestamp of changes)
- Supports reproducible research!



# Today's tutorial

- Intro to Git Internals
- Basic Git commands
- Git demo
- GitHub demo

The missing semester of your CS education  
<https://missing.csail.mit.edu/2020/version-control/>



<https://xkcd.com/1597/>

# GIT'S DATA MODEL

Solo Group Meeting










24 February 2023

Emmie Le Ray

# A “snapshot” of a generic project folder

```
folder/  
├── subfolder/  
│   └── file1.txt  
│       "hello world"  
└── file2.txt  
    "git is awesome"
```

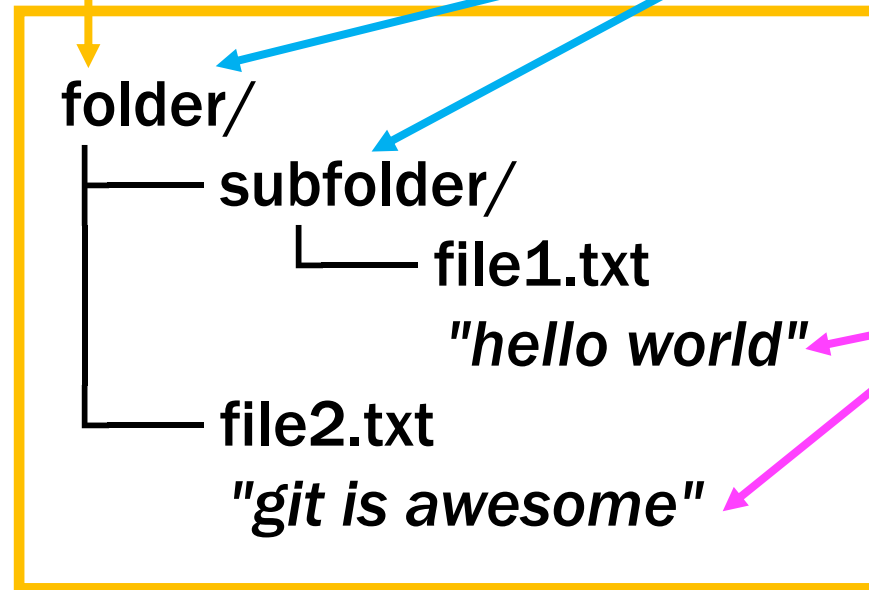
# “Save As” is one way to take snapshots of your project

Today		
	20220102_precip_project_v1	>
	20220302_precip_project_v2	>
	20220514_precip_project_v3	>
	20220616_precip_project_v4	>
	20220708_precip_project_v5	>
	20220922_precip_project_v6	>
	20221001_precip_project_v6_final	>
	20221018_precip_project_v6_final_FINAL	>
	20221201_precip_project_v7_FINAL	>
	20221204_precip_project_v7_final_with_edits	>

# Git's version of "Save As" stores data as three object types: blobs, trees, and commits

"commit" = snapshot + metadata

"tree" = folder contents

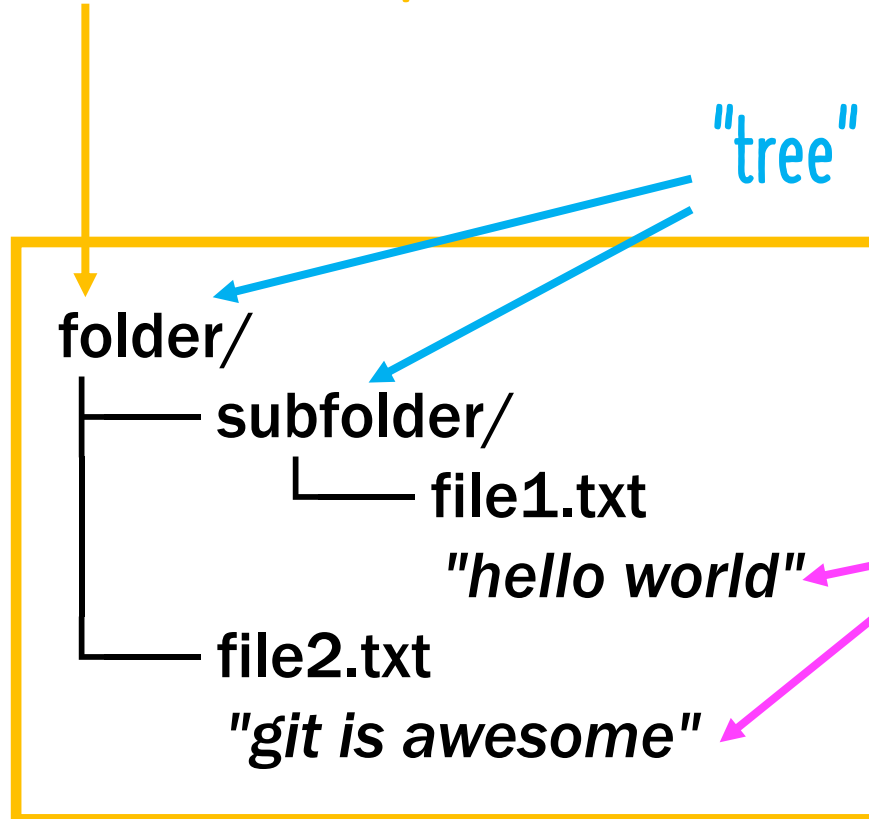


"blob" = file contents

A **commit** is a “snapshot” of what the entire directory looks like at a given time

```
type = struct{  
  snapshot: top-level tree  
  parents: array<commit>,  
  author: string,  
  message: string,  
}
```

“commit” = snapshot + metadata



“tree” = folder contents

type = map<string, tree | blob>

“blob” = file contents

type = array<byte>



commit →

```
~/tutorials/demo/.git/objects/3b single_line  
git cat-file -p 8426bf9a7042b8dabf531b0b91772f8fab0db2fa  
tree 68aba62e560c0ebc3396e8ae9335232cd93a3f60  
author emmleroy <47359313+emmleroy@users.noreply.github.com> 1677040055 -0500  
committer emmleroy <47359313+emmleroy@users.noreply.github.com> 1677040055 -0500  
  
Initial commit
```

tree →

```
~/tutorials/demo/.git/objects/3b single_line  
git cat-file -p 68aba62e560c0ebc3396e8ae9335232cd93a3f60  
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad    hello.txt
```

blob →

```
~/tutorials/demo/.git/objects/3b single_line  
git cat-file -p 3b18e512dba79e4c8300dd08aeb37f8e728b8dad  
hello world
```

What is “a1337c4664981e4397625791c8ea3bbb5f2279a3”?

Each object is identified by the **SHA-1 hash** of its contents

commit [a133]



```
[cef7] folder/  
├── [f92a] subfolder/  
│   └── file1.txt  
│       [73d8] "hello world"  
└── file2.txt  
    [ef32] "git is awesome"
```

A **hash function** takes variable sized data as input and returns a fixed-size hash value as output

Rossby waves, also known as planetary waves, are a type of inertial wave naturally occurring in rotating fluids. They were first identified by Sweden-born American meteorologist Carl-Gustaf Arvid Rossby. They are observed in the atmospheres and oceans of planets owing to the rotation of the planet.

```
8950 4e47 0d0a 1a0a 0000 000d 4948 4452 0000  
000a 0000 000a 0806 0000 008d 32cf bd00 0000  
98dc 1bc1 f62c e660 310f a99c 469a dafc 8d7a
```

```
def my_function():  
    print("Hello from a function")
```

Hash Function

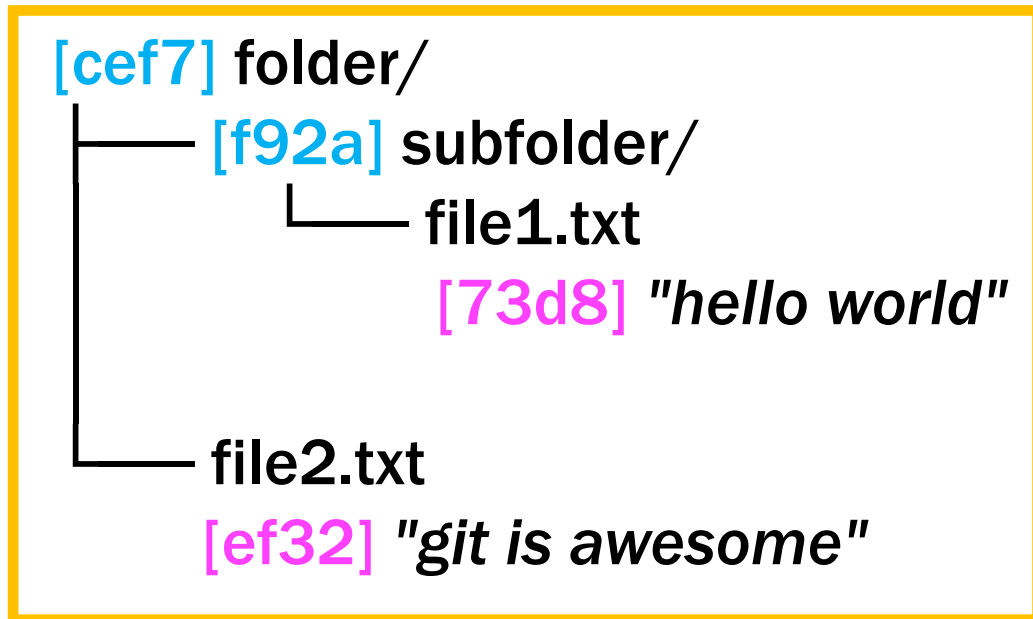
```
9bceaac05c772aff0815074033398f431aab689a
```

```
8426bf9a7042b8dabf531b0b91772f8fab0db2fa
```

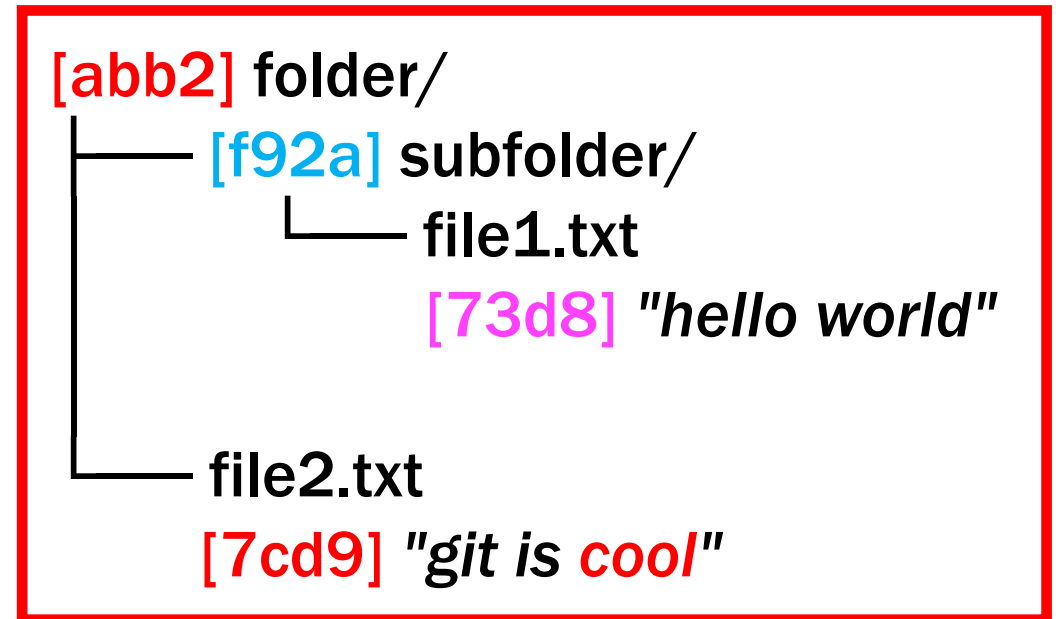
```
fdfe486da15df52af3478ec8eeeeac049b7a9762
```

# Git can quickly check if two objects are identical and only store unique objects

commit [a133]



commit [b64f]



# A **reference** is a human-readable name for a git commit

References are simply pointers to commits.

commit [a133]



```
[cef7] folder/  
├── [f92a] subfolder/  
│   └── file1.txt  
│       [73d8] "hello world"  
└── file2.txt  
    [ef32] "git is awesome"
```

commit [b64f]



master/main

```
[abb2] folder/  
├── [f92a] subfolder/  
│   └── file1.txt  
│       [73d8] "hello world"  
└── file2.txt  
    [7cd9] "git is cool"
```

```
~/tutorials/demo/.git/refs/heads → single_line  
cat main  
916f13a398331c11d83267213ddcab36809032a2
```

A **git repository** is a collection of objects and references

commit [a133]



[cef7] folder/  
├── [f92a] subfolder/  
│ └── file1.txt  
│ [73d8] "hello world"  
└── file2.txt  
 [ef32] "git is awesome"


commit [b64f]

master/main



[abb2] folder/  
├── [f92a] subfolder/  
│ └── file1.txt  
│ [73d8] "hello world"  
└── file2.txt  
 [7cd9] "git is cool"

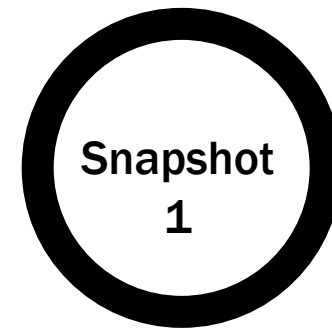
```
~/tutorials/demo single_line  
ls .git  
COMMIT_EDITMSG config hooks info objects  
HEAD description index logs refs
```



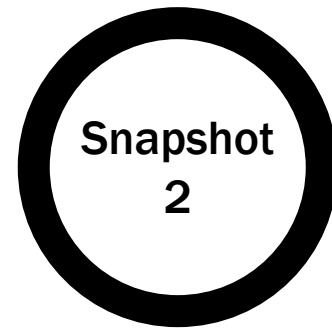


# Git's Data Model

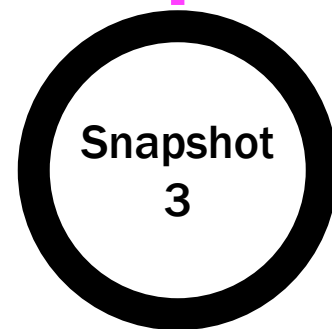
- A commit is a “snapshot” in time
- To track history, each commit points to a set of parent commits
- A git repository is a collection of commits and references



commit [a133]  
tree: cef7  
parents  
author: Emmie



commit [b64f]  
tree: abb2  
parents: a133  
author: Emmie



commit [d7ee]  
tree: a912  
parents: b64f  
author: Emmie

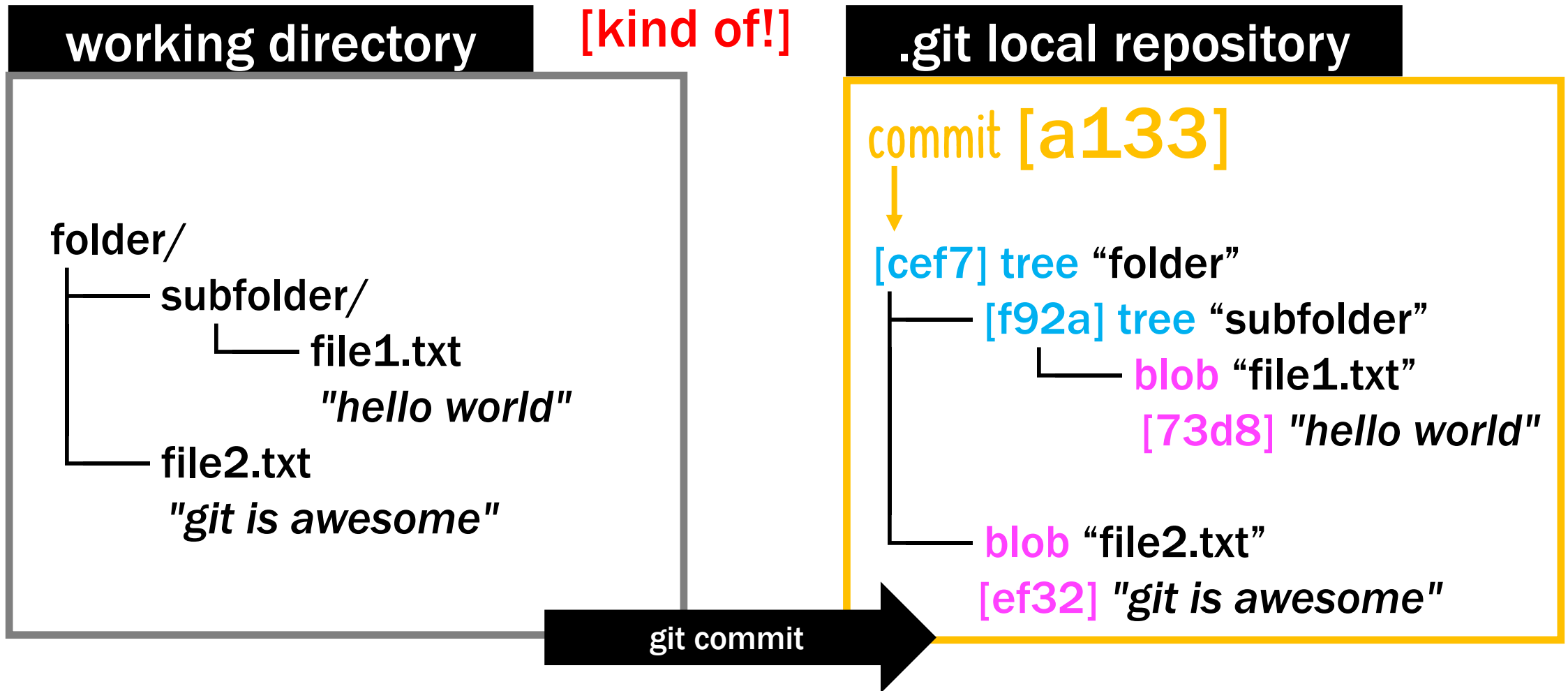
# USEFUL GIT COMMANDS

Solo Group Meeting

24 February 2023

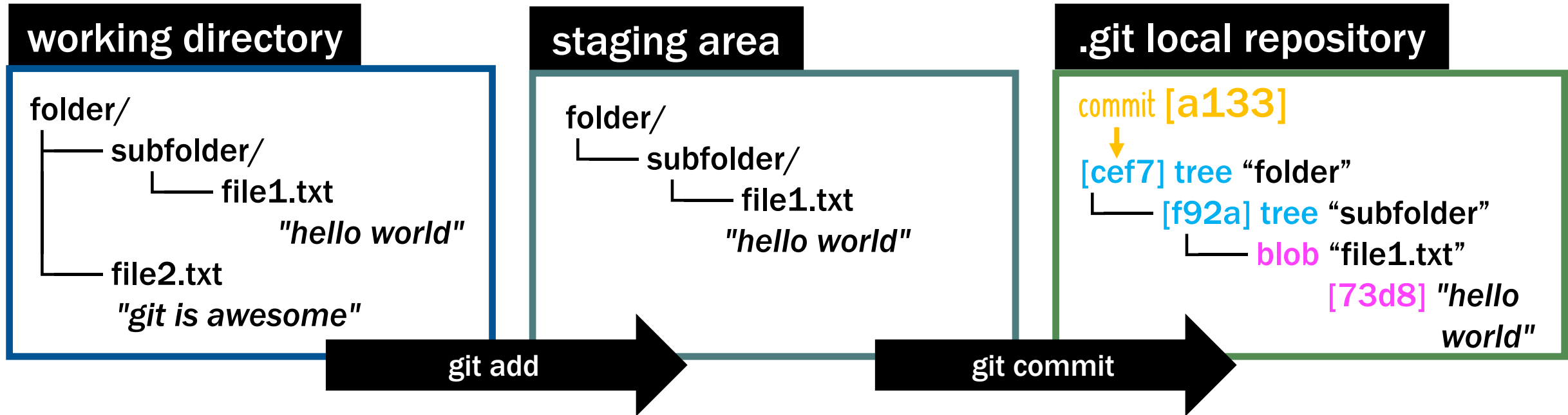
Emmie Le Roy

**git commit** creates a snapshot of your current working directory  
*It saves a collection of objects in the local .git repository*



# But Git has a **staging area**!

*For flexibility!*

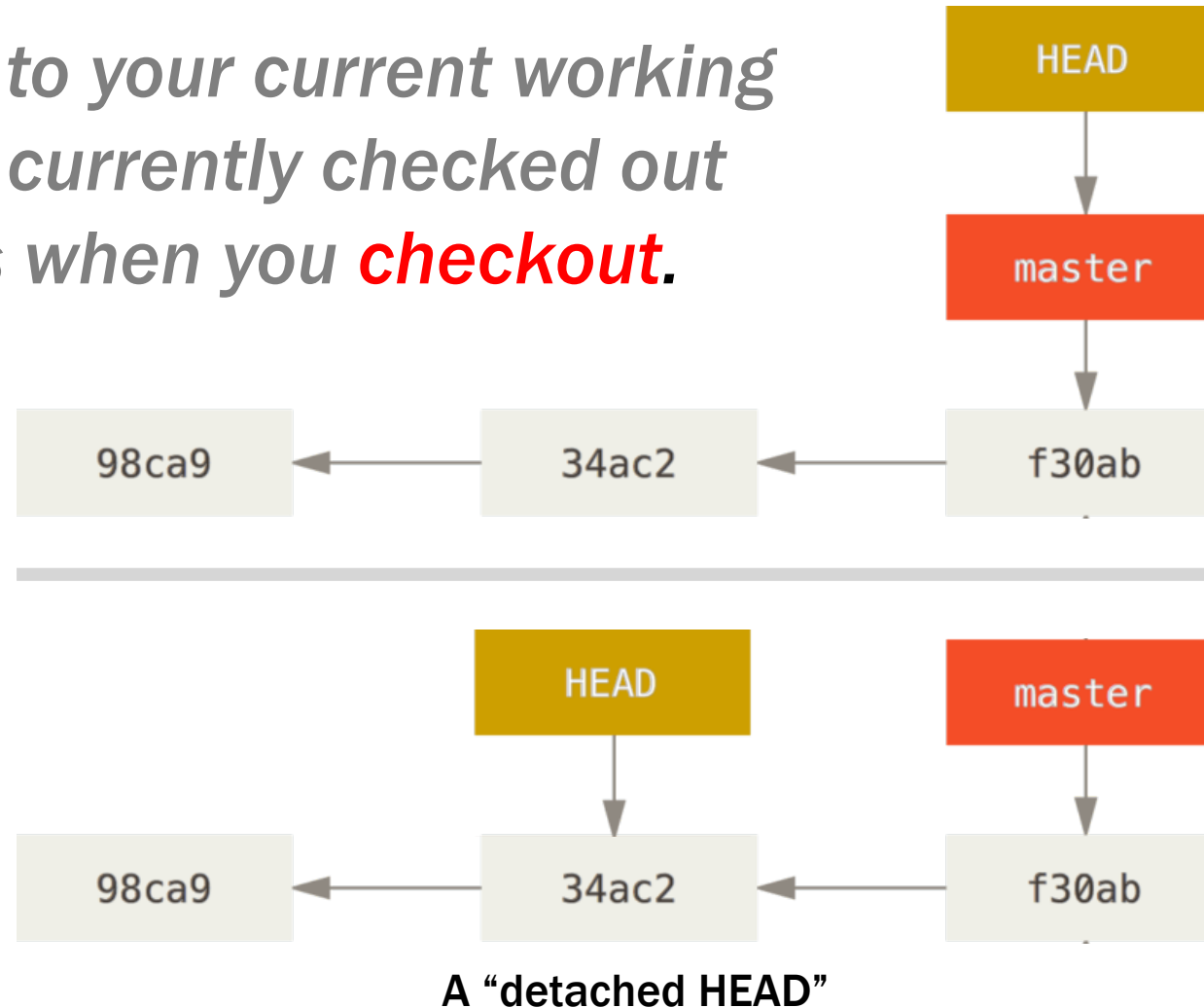


**git add** tells git what changes to include in the next commit

**git commit** commits the staged changes to the local repository

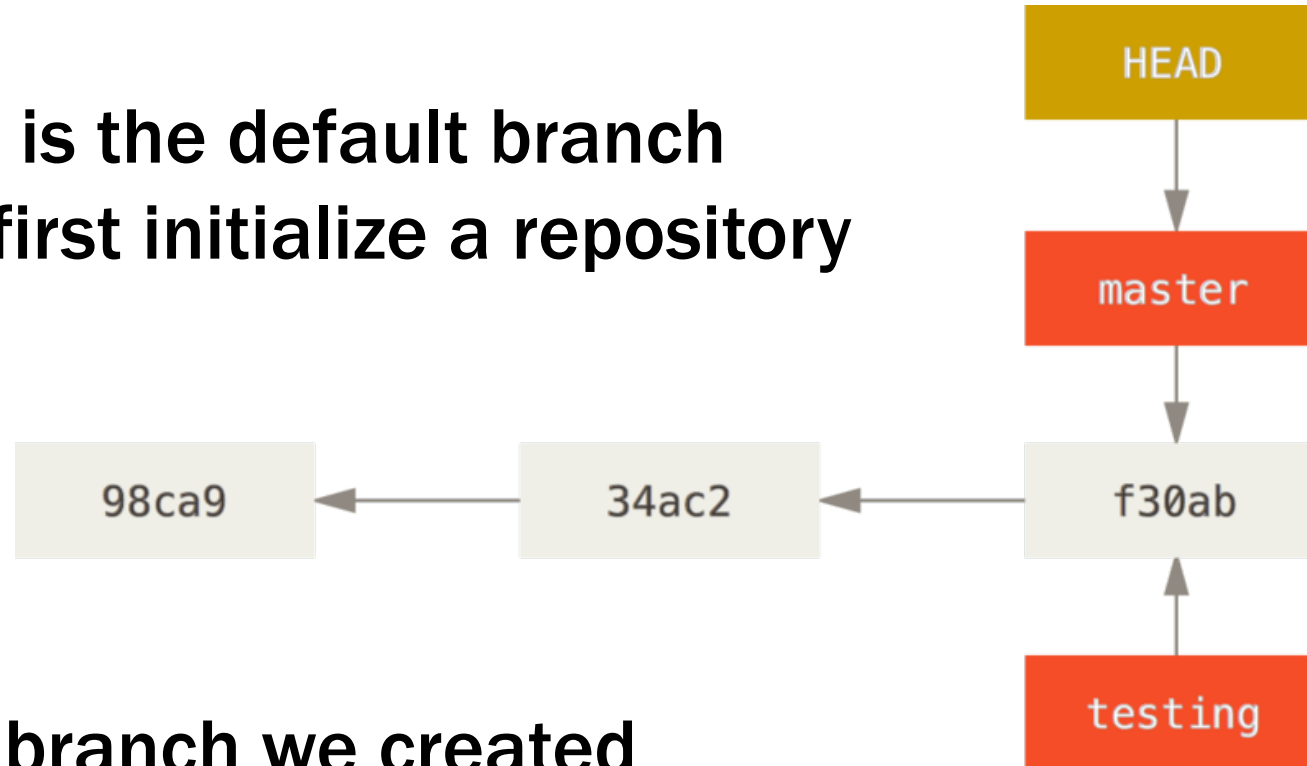
**git checkout** allows you to move around in your version history

**HEAD** is a pointer to your current working directory (i.e., the currently checked out commit), it moves when you **checkout**.



**git branch** allows you to experiment without breaking the project.  
*Creating a branch creates a pointer to a particular commit*

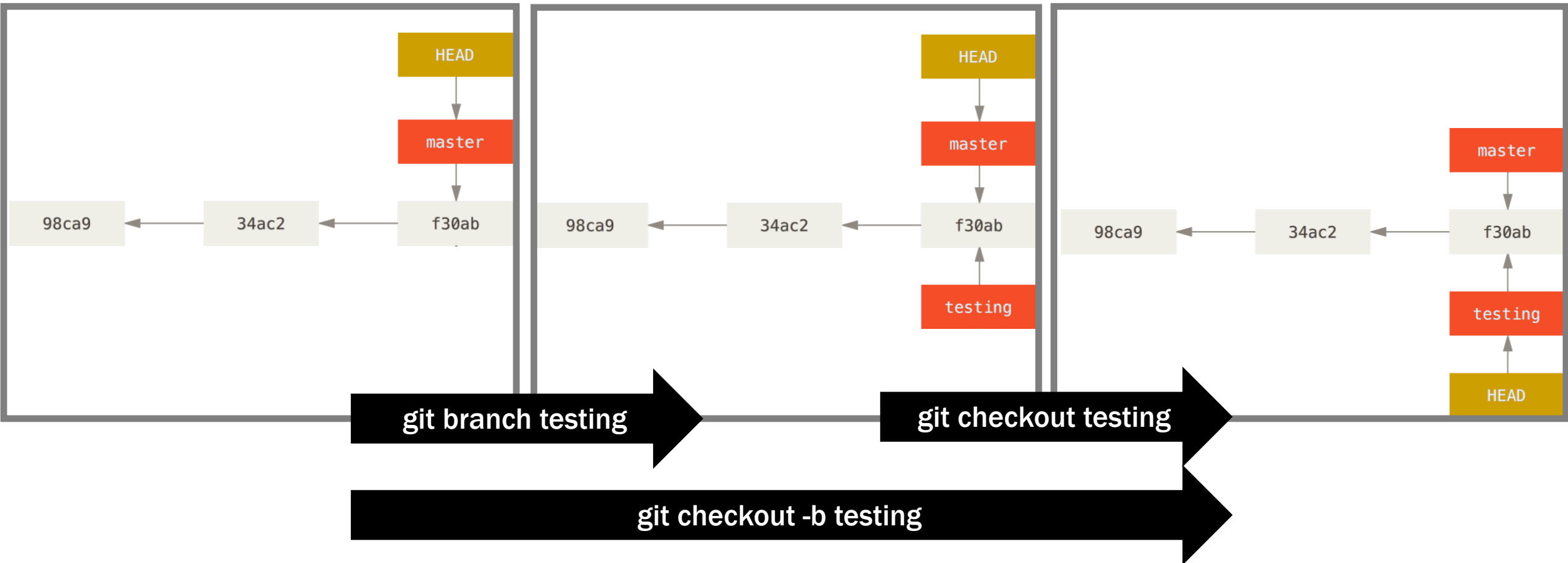
**master (or main)** is the default branch name when you first initialize a repository



**testing** is a NEW branch we created while checking out the master branch

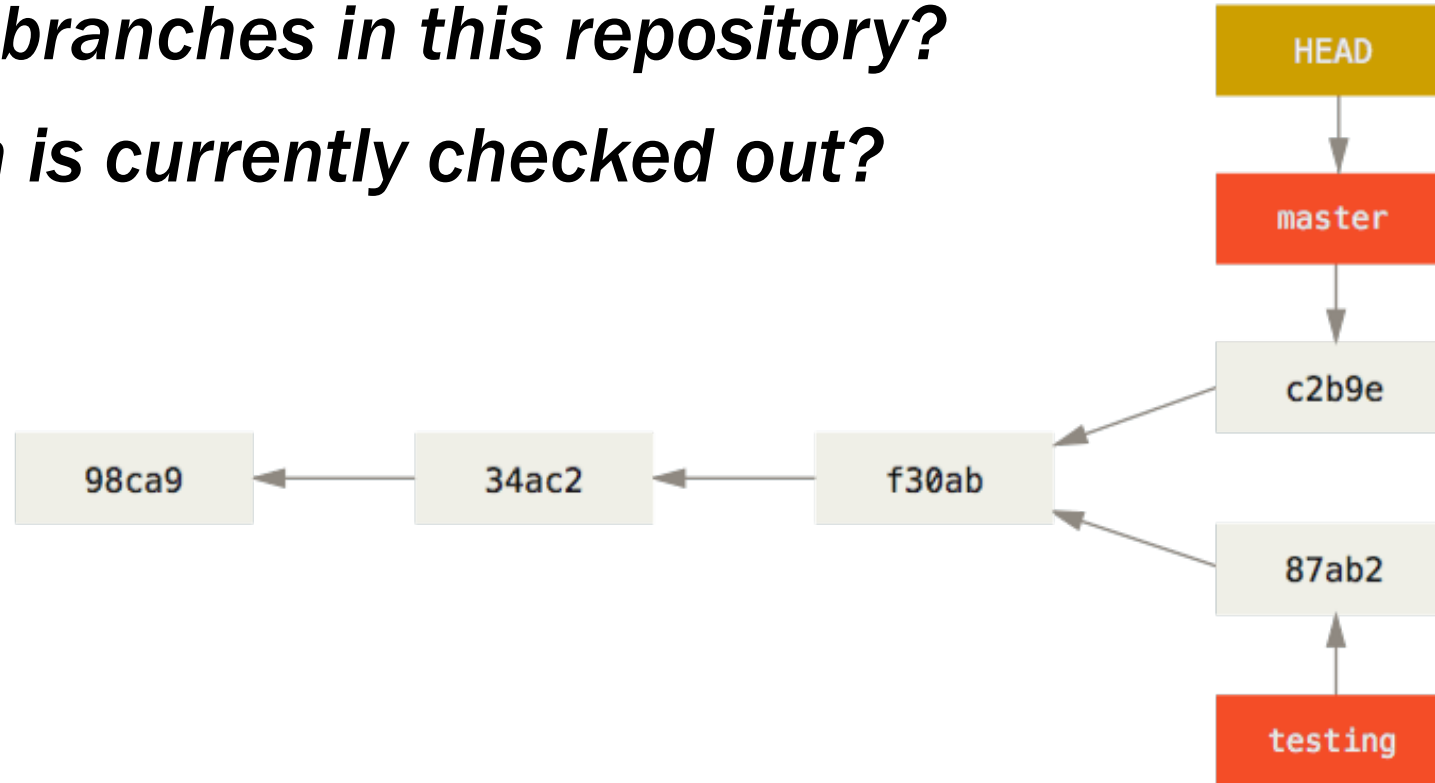
`git checkout -b <new_branch>`

creates a new branch and checks it out



***What are the branches in this repository?***

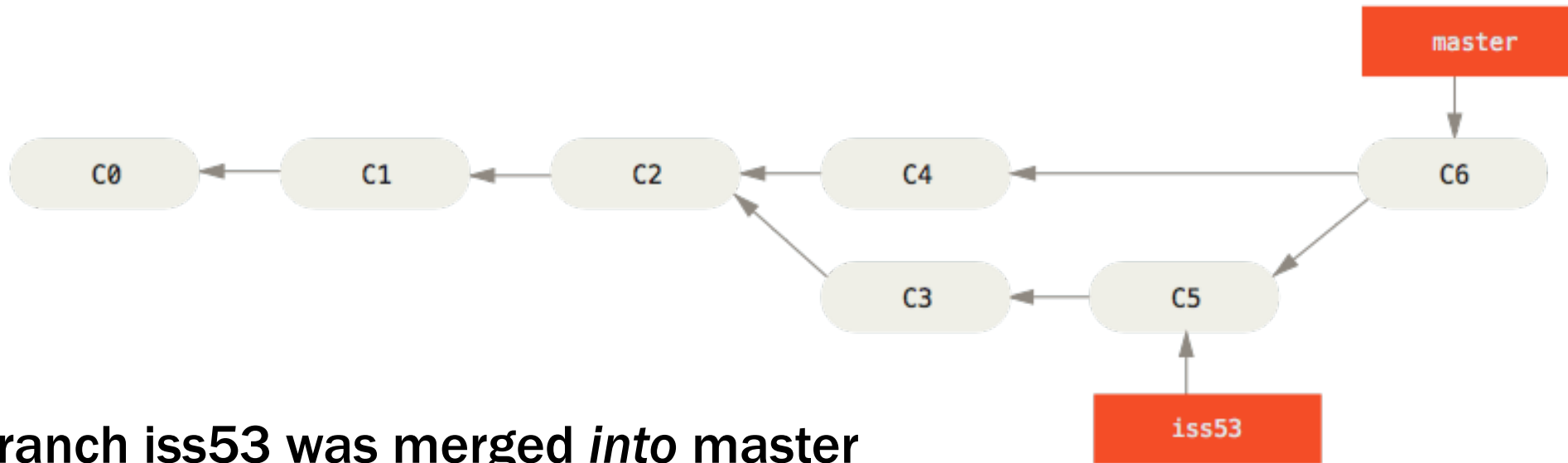
***Which branch is currently checked out?***





**git merge** creates a commit with multiple parents

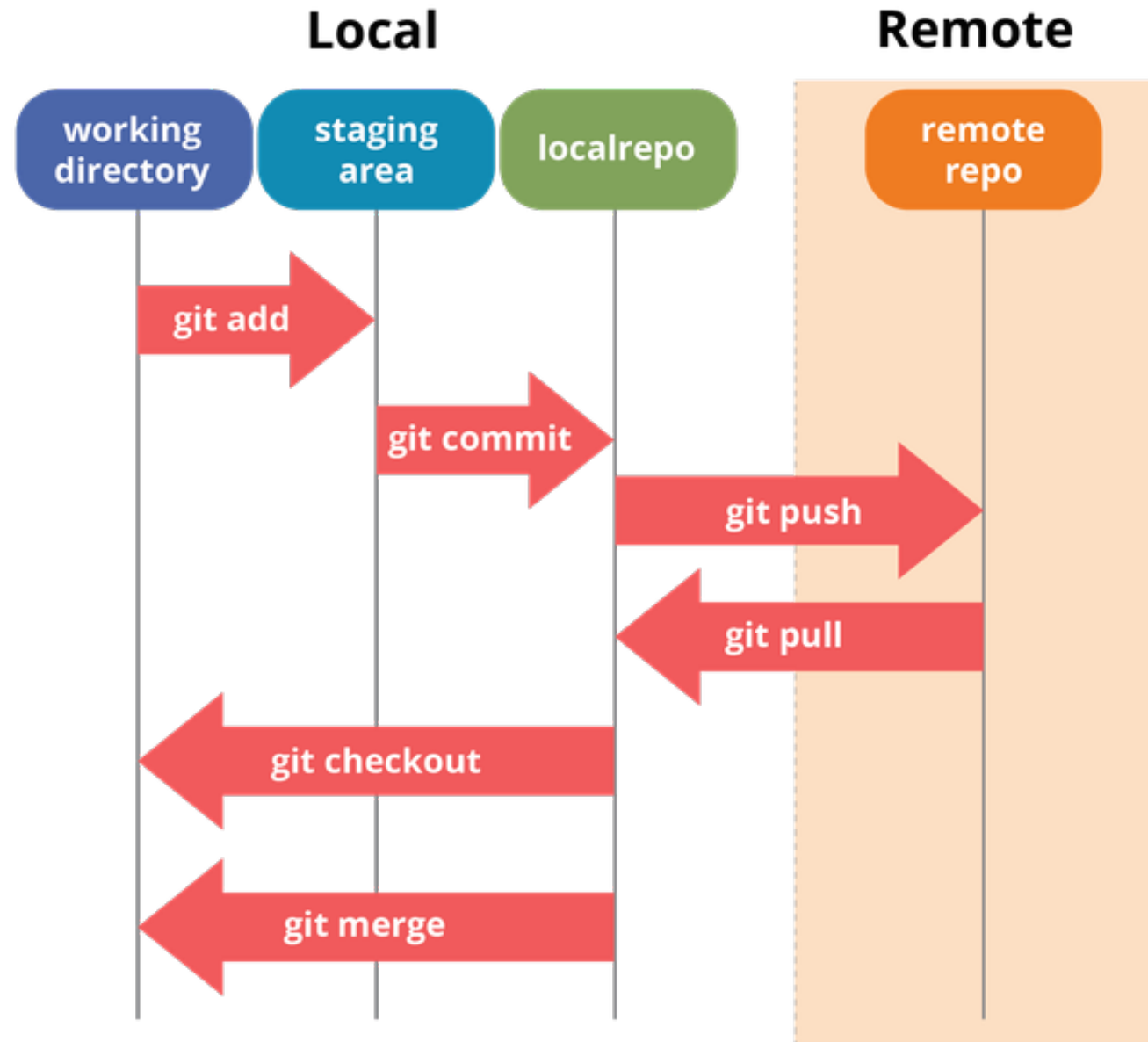
Rule of thumb: *You always merge into the current branch*



branch **iss53** was merged *into* master

\$ git checkout master (checkout the branch you want to merge *into*)

\$ git merge iss53 (merge the branch you want to merge *from*)





# GIT COMMANDS

- git **checkout** switches the working directory to a different branch by moving HEAD
- git **branch** gives a name to a particular commit to branch off a new line of development
- git **merge** creates a commit with multiple parents
- **HEAD** is a pointer to the current working directory

**git help <command>: get help for a git command**

**git init: initialize a git repository**

**git add <filename>: add files to staging area**

**git diff <filename>: shows changes you made relative to the staging area**

**git commit -m <commit\_message>: create a new commit**

**git status: tells you what's going on**

**git log: show a flatten log of history**

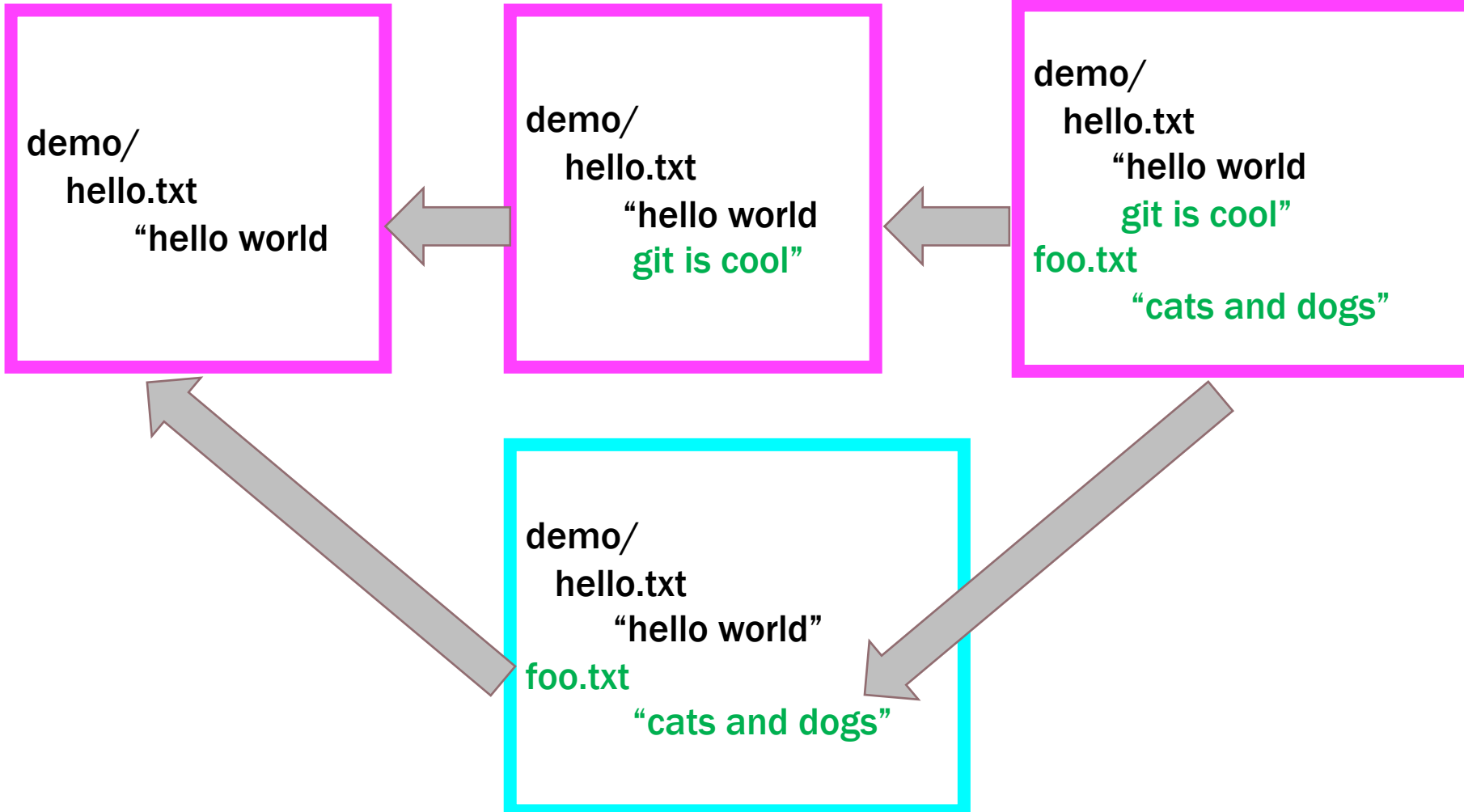
**git branch: list the local branches**

**git branch <new\_branch> : creates a branch**

**git checkout <existing\_branch >: switch current branch and HEAD**

**git checkout -b <new\_branch>: create a branch and switch to it**

**git merge <existing\_branch>: merge existing\_branch into current branch**



```
$ mkdir tutorial/  
$ mkdir demo/  
$ cd demo  
$ echo "hello world" > hello.txt  
$ ls  
$ git init  
$ ls  
$ ls -a  
$ cd .git  
$ cd ..  
$ git status  
$ git add hello.txt  
$ git status  
$ git commit -m "Initial commit"
```



**demo/  
hello.txt  
“hello world”**

```
$ git status
$ git log
$ git cat-file -p <SHA-1 hash>
$ echo "git is cool" >> hello.txt
$ cat hello.txt
$ git status
$ git diff hello.txt
$ git commit -m "Add another line to
hello.txt"
$ git add "hello.txt"
$ git commit -m "Add another line to hello.txt"
$ git status
$ git log
```

demo/  
hello.txt  
"hello world  
git is cool"

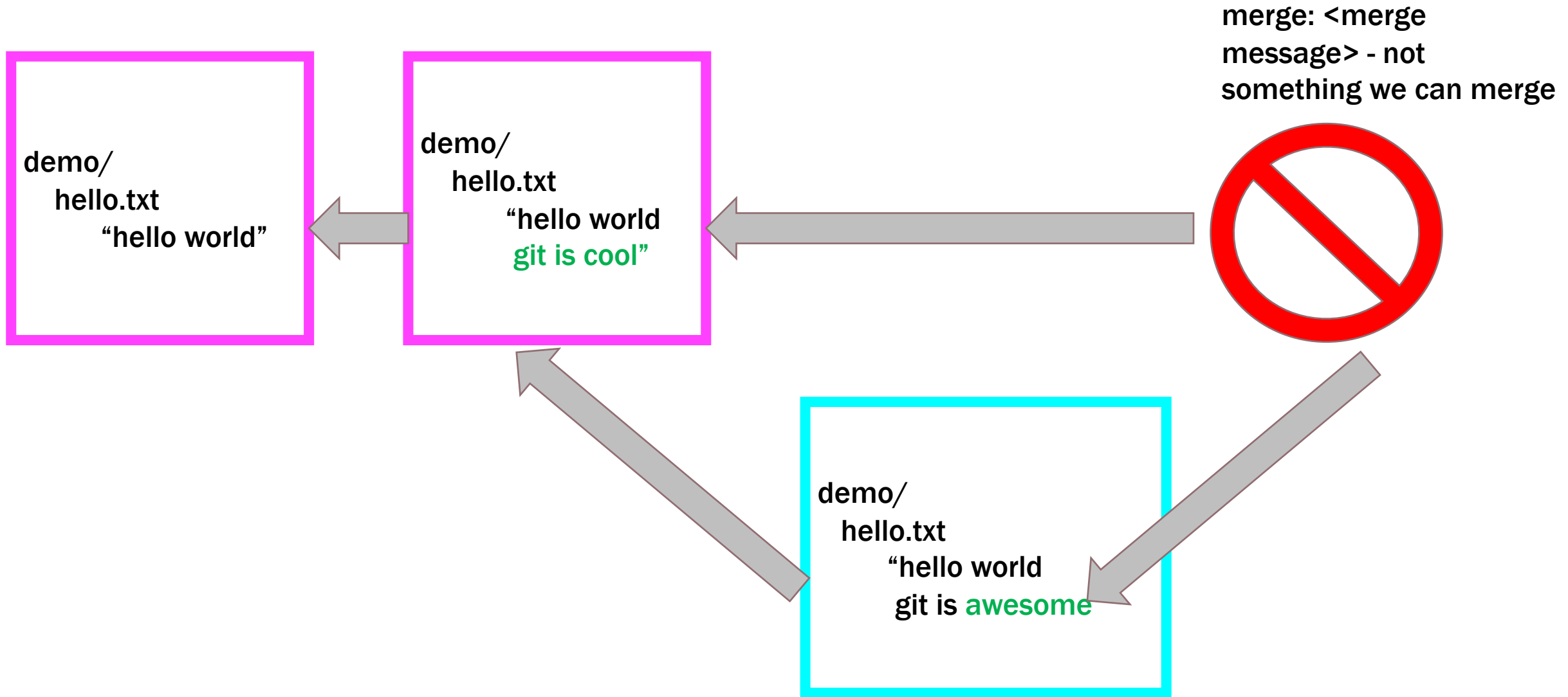
```
$ git checkout 5c6979d38e203e58a9eb16f55444b6f4e89e80ad
$ git log
$ git status
$ git branch dev
$ git log
$ git status
$ git checkout dev
$ git log
$ git status
$ echo "cats and dogs" > foo.txt
$ ls
$ git status
$ git add foo.txt
$ git commit -m "Create foo.txt"
$ git status
$ git log
$ cat hello.txt
```

**demo/**  
**hello.txt**  
    **"hello world"**  
**foo.txt**  
    **"cats and dogs"**



```
$ git checkout master  
$ git merge dev -m "Merge dev into master"  
$ git log --all --graph -decorate  
$ git branch -d dev
```

```
demo/  
  hello.txt  
    "hello world  
      git is cool"  
foo.txt  
  "cats and dogs"
```





# GitHub

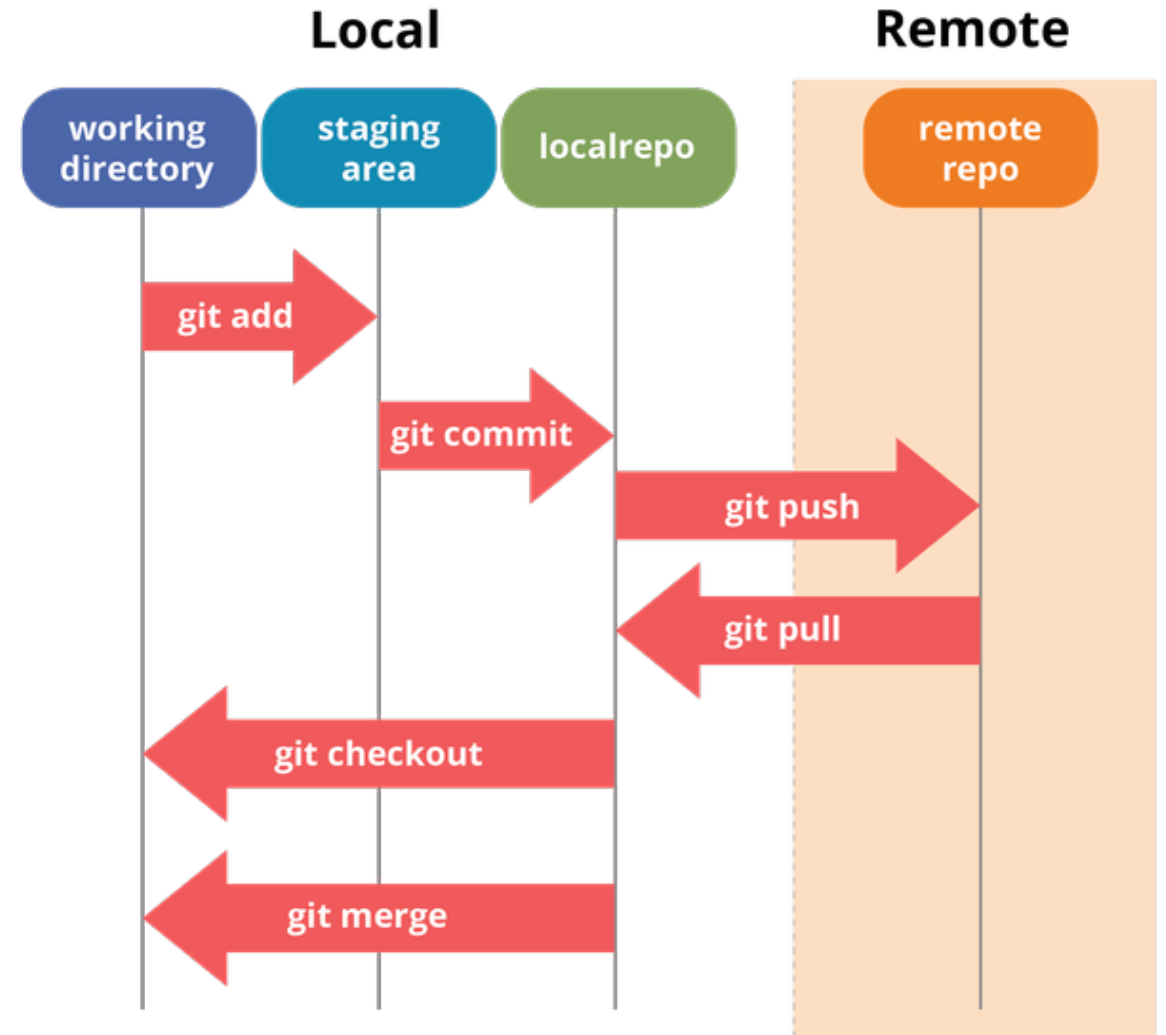
Solo Group Meeting

24 February 2023

Emilia Le Ray

# What is GitHub?

- GitHub is not Git
- GitHub is a sharing and collaboration platform
- GitHub is an online hosting service for Git repositories



# Local vs Remote repositories

**Remote** repo: lives on GitHub,  
where others can make a copy

## REMOTE

A GitHub repo  
github.com/emmleroy



**Local** repo: lives on your machine,  
where we make our changes and  
do our development

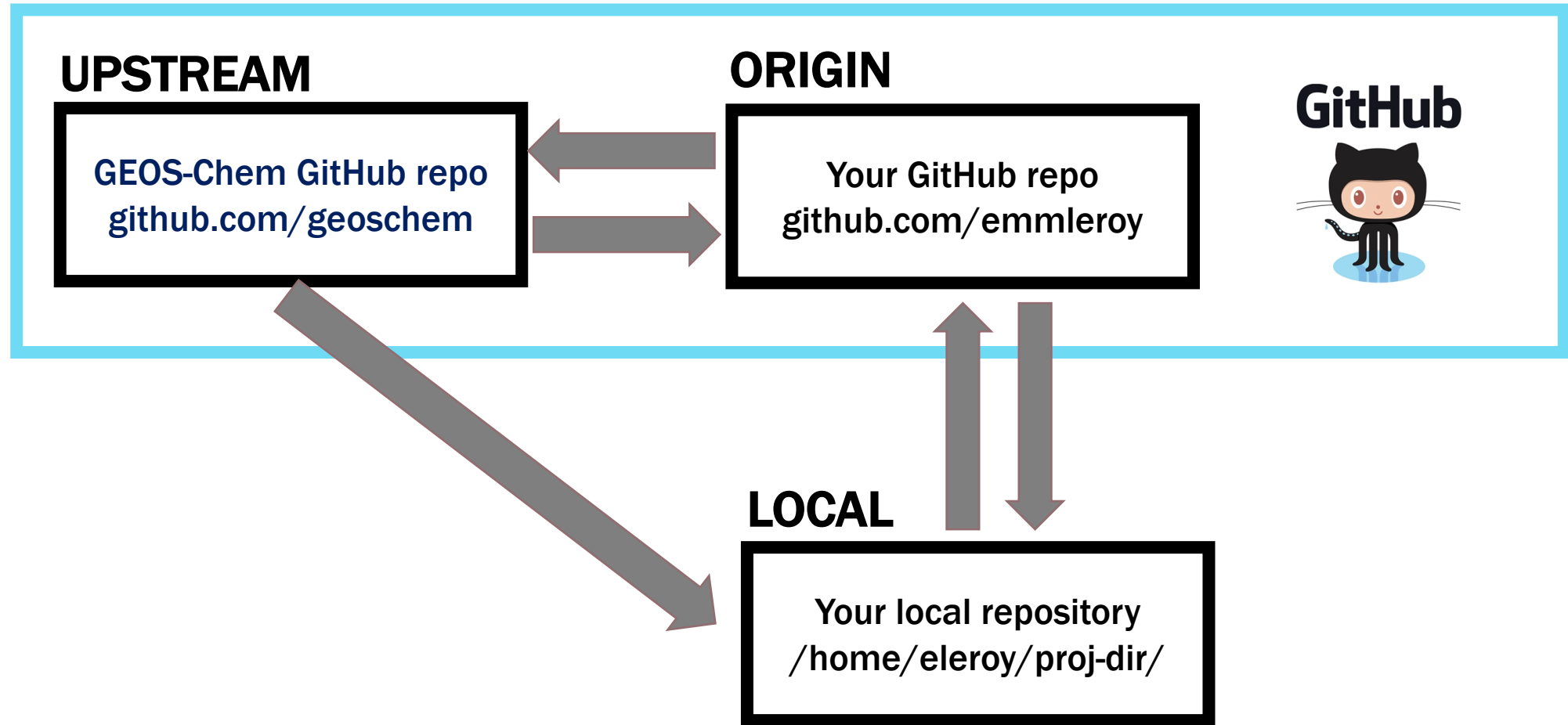
## LOCAL

Your local repository  
/home/eleroy/proj-dir/



Remote repositories provide us with a backup copy hosted online and allows us to share and publish our code

# Remote repositories can belong to you or to someone else



# GitHub

**Part 1.** I have a local project that I want to share with others

**Part 2.** I want to create my own changes to someone else's project

# GitHub

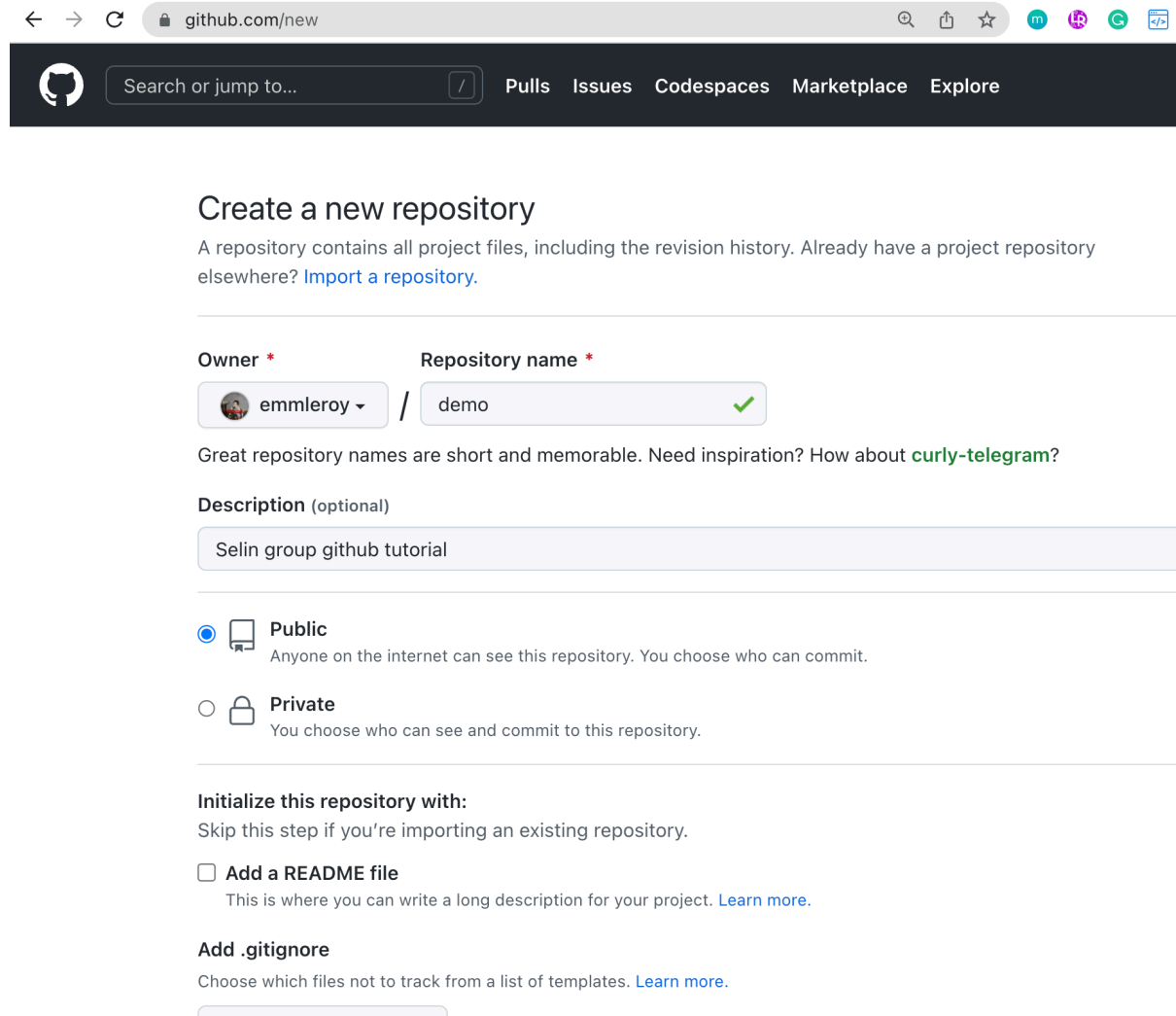


# Getting Started

1. Create a free account on GitHub.com
2. Generate a personal access token at <https://github.com/settings/tokens>



# Create a GitHub repository at `github.com/new`



The screenshot shows the GitHub 'Create a new repository' page. At the top is a dark navigation bar with the GitHub logo, a search bar, and links for Pulls, Issues, Codespaces, Marketplace, and Explore. The main heading is 'Create a new repository', followed by a subtext explaining that a repository contains project files and revision history, with a link to 'Import a repository' if one already exists elsewhere. The form fields include: 'Owner' (a dropdown menu showing 'emmleroy'), 'Repository name' (a text box with 'demo' and a green checkmark), and 'Description' (optional, with a text box containing 'Selin group github tutorial'). Below these are two radio button options for visibility: 'Public' (selected, with a subtext 'Anyone on the internet can see this repository. You choose who can commit.') and 'Private' (unselected, with a subtext 'You choose who can see and commit to this repository.'). The next section is 'Initialize this repository with:', which includes a subtext 'Skip this step if you're importing an existing repository.' and a checkbox for 'Add a README file' (unselected, with a subtext 'This is where you can write a long description for your project. Learn more.'). The final section is 'Add .gitignore', with a subtext 'Choose which files not to track from a list of templates. Learn more.' and a partially visible dropdown menu.

github.com/new

Search or jump to... / Pulls Issues Codespaces Marketplace Explore

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

**Owner \*** **Repository name \***

emmleroy / demo ✓

Great repository names are short and memorable. Need inspiration? How about [curly-telegram?](#)

**Description** (optional)

Selin group github tutorial

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

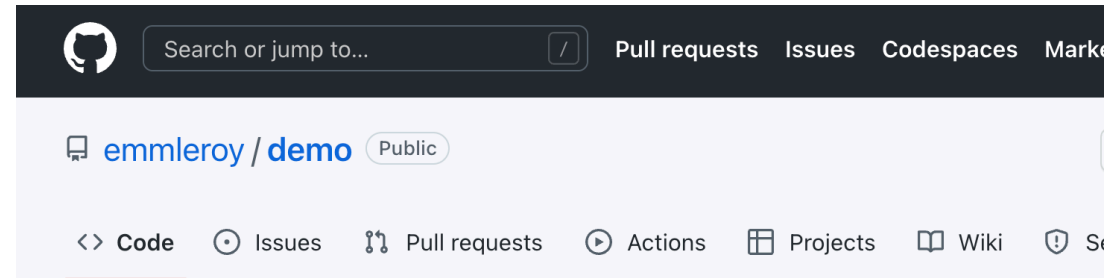
**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)


**Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

GitHub provides two URLs: a **HTTPS URL** and a **SSH/git URL**

Copy the *HTTPS URL*



#### Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

HTTPS

SSH

`https://github.com/emmleroy/demo.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository

#### ...or create a new repository on the command line

```
echo "# demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/emmleroy/demo.git
git push -u origin main
```

#### ...or push an existing repository from the command line

```
git remote add origin https://github.com/emmleroy/demo.git
git branch -M main
git push -u origin main
```

**git remote add <remote> <URL>** to link an existing local repository to a remote repository

```
$ cd demo/  
$ git remote add origin https://github.com/<myUserName>/demo.git  
$ git remote -v
```

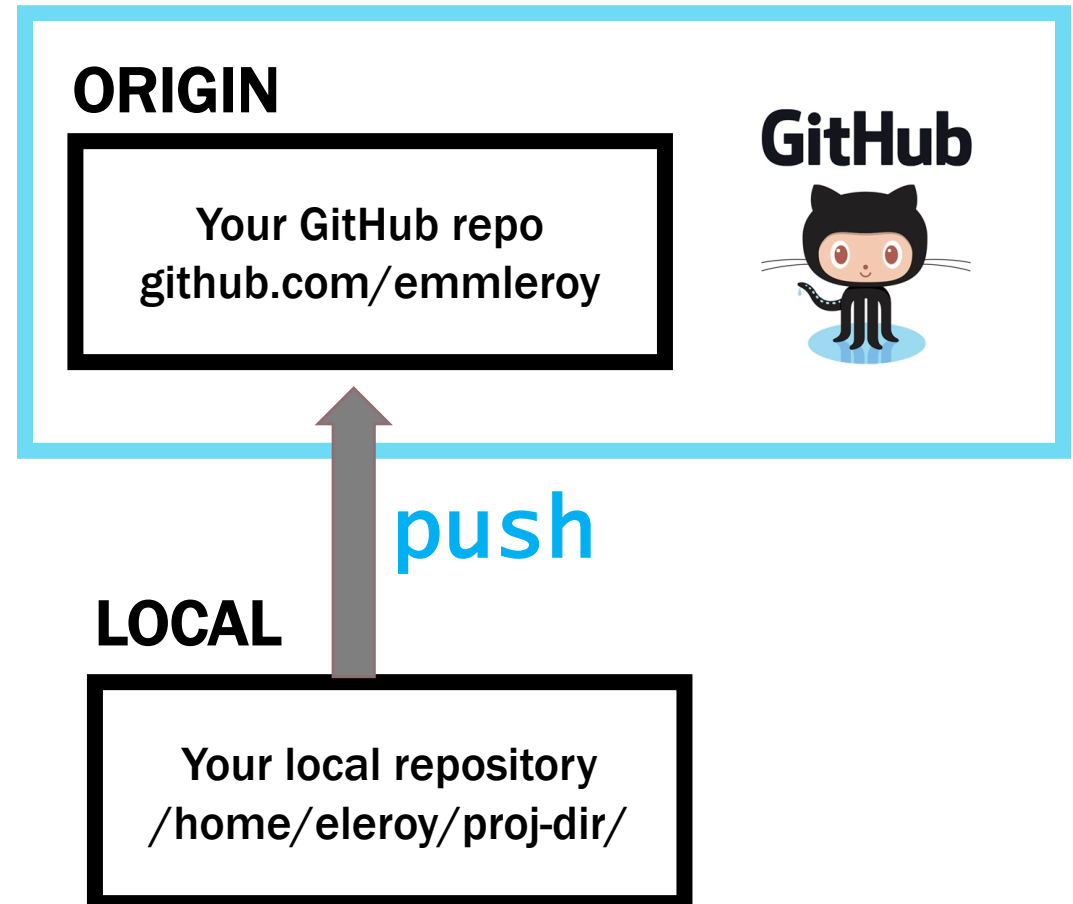
```
(base) [eleroy@fs03 demo]$ git remote add origin https://github.com/emmleroy/demo.git  
(base) [eleroy@fs03 demo]$ git remote -v  
origin https://github.com/emmleroy/demo.git (fetch)  
origin https://github.com/emmleroy/demo.git (push)  
(base) [eleroy@fs03 demo]$
```

**origin** is the conventional short name for your remote repository (i.e. <remote>)

# **git push** to upload content from your local repository to the remote repository

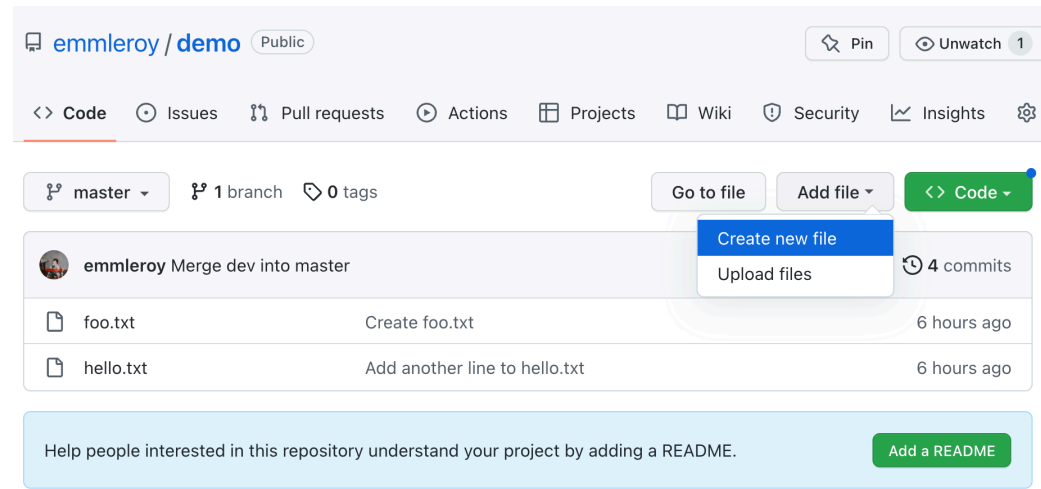
```
$ git push -u <remote> <local_branch>  
$ git push -u origin master
```

The **-u** flag sets origin/branch as the remote (upstream) target so pushes after that can be called with just “git push” (but it doesn’t hurt to be explicit!)

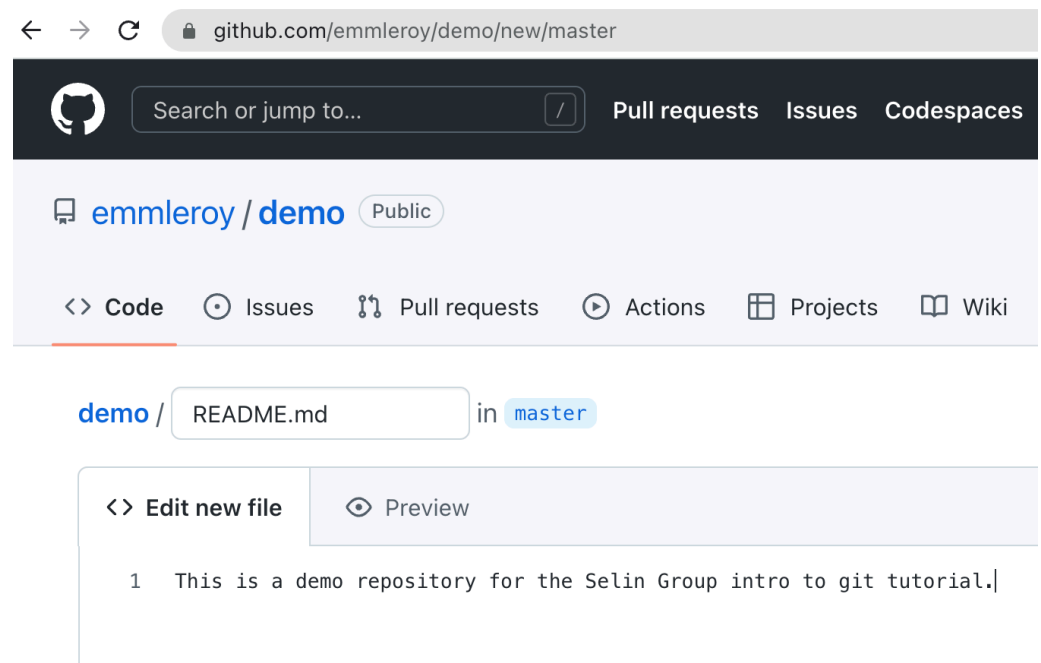


## 6. Create a README.md file on GitHub.com

*In practice, you might not make changes to your files on GitHub.com, but the point here is to simulate a remote change that you will then pull to your local repository.*



The screenshot shows the GitHub repository page for 'emmleroy / demo'. The 'Add file' dropdown menu is open, showing 'Create new file' and 'Upload files'. The repository has 1 branch (master) and 0 tags. The commit history shows two commits: 'foo.txt' (Create foo.txt) and 'hello.txt' (Add another line to hello.txt), both 6 hours ago. A banner at the bottom encourages adding a README.

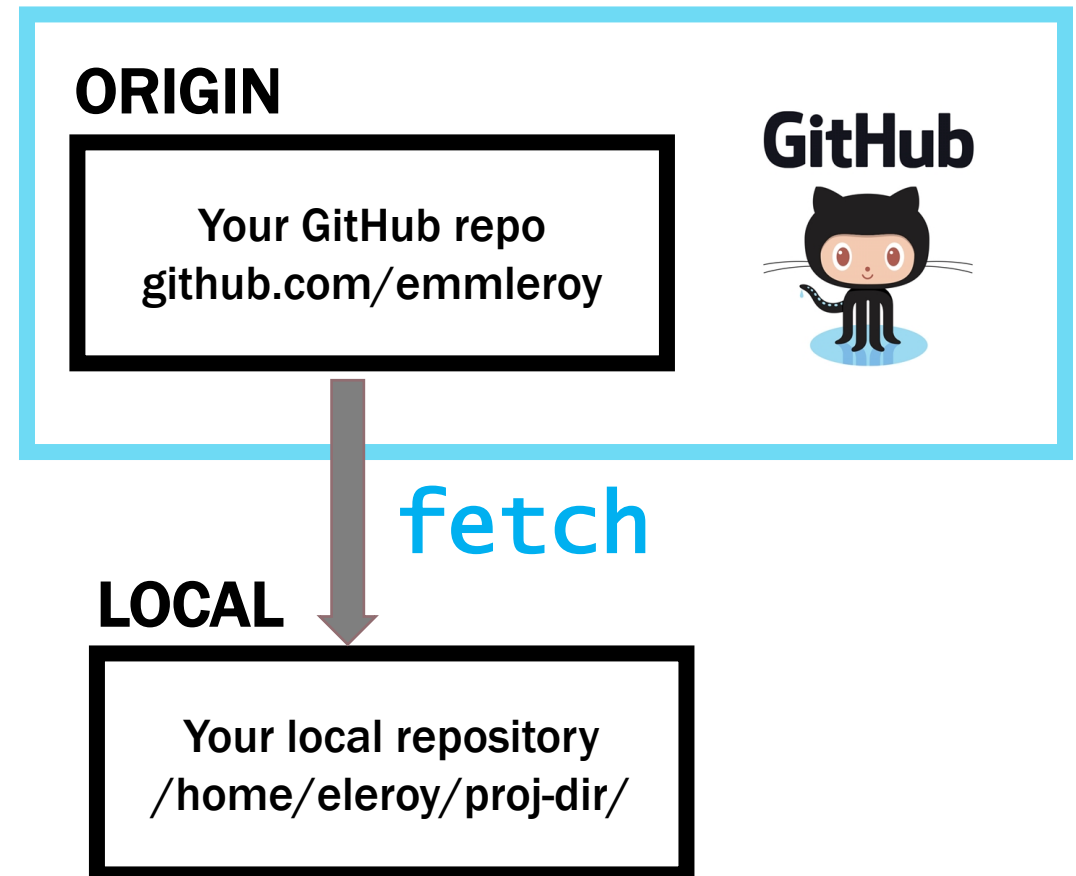


The screenshot shows the 'README.md' file being edited in the 'demo' branch. The file content is: '1 This is a demo repository for the Selin Group intro to git tutorial.' The 'Edit new file' tab is active.

**git fetch** to see what everybody else has been working on *without* merging those changes

```
$ git status
$ git fetch <remote> <remote_branch>
$ git fetch origin master
$ git status
```

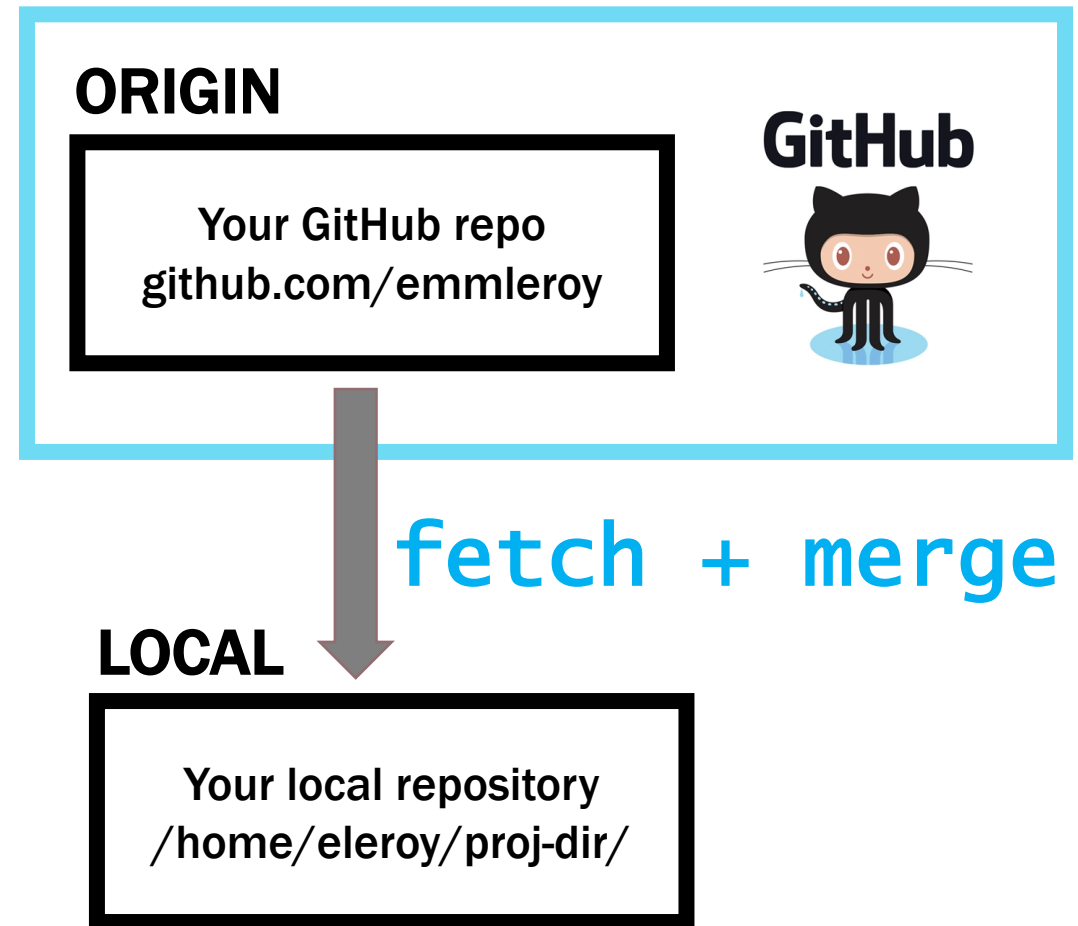
```
(base) [eleroy@fs03 demo]$ git fetch origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 750 bytes | 375.00 KiB/s, done.
From https://github.com/emmleroy/demo
* branch          master      -> FETCH_HEAD
  74a6d94..8ec7fd4 master      -> origin/master
(base) [eleroy@fs03 demo]$
```



# **git merge** to merge the upstream changes into the currently checked out branch

```
$ git merge <remote>/<remote_branch>  
$ git merge origin/master
```

```
(base) [eleroy@fs03 demo]$ git merge origin/master  
Updating 74a6d94..8ec7fd4  
Fast-forward  
 README.md | 1 +  
 1 file changed, 1 insertion(+)  
 create mode 100644 README.md  
(base) [eleroy@fs03 demo]$ ls  
foo.txt  hello.txt  README.md  
(base) [eleroy@fs03 demo]$
```



# git pull = git fetch + git merge

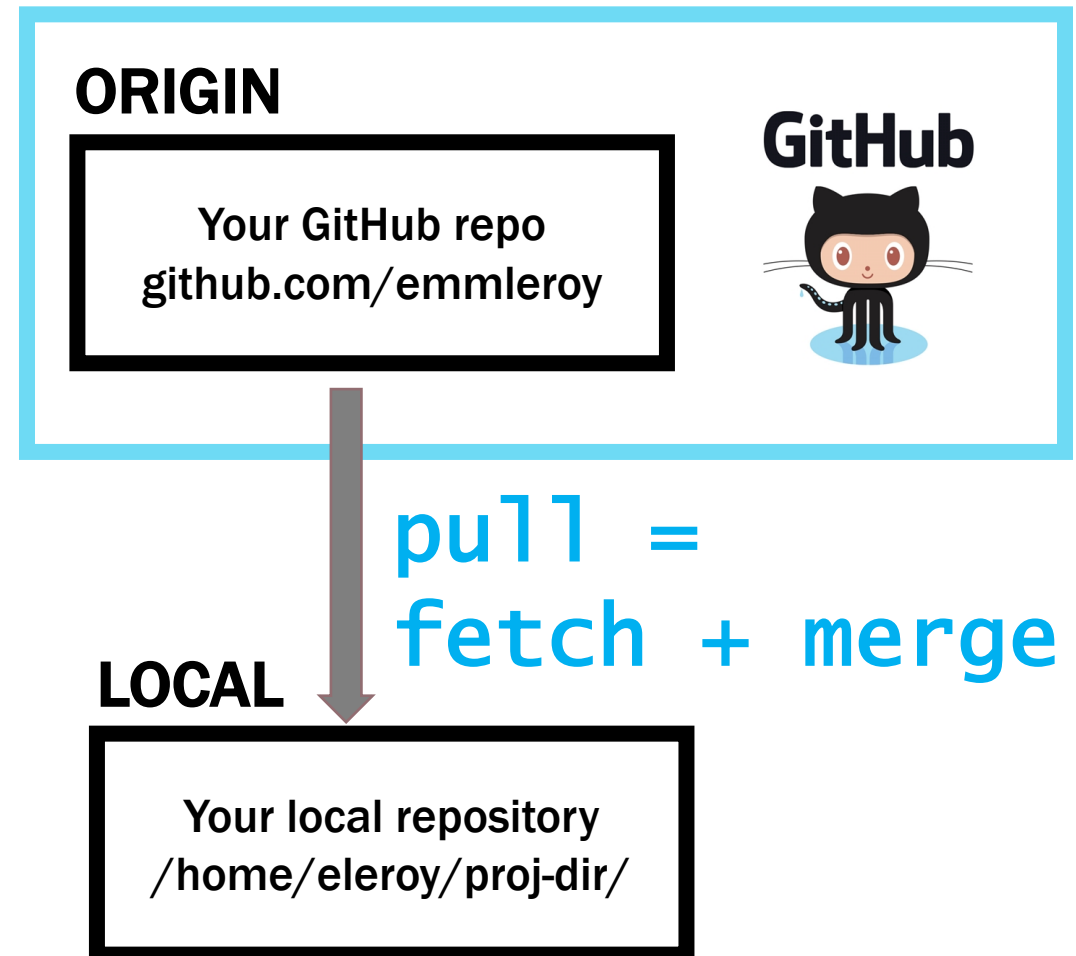
```
$ git pull <remote> <remote_branch>
or
$ git fetch <remote> <remote_branch>
$ git merge <remote>/<remote_branch>
```

```
(base) [eleroy@fs03 demo]$ git pull
warning: Pulling without specifying how to reconcile divergent branches is
discouraged. You can squelch this message by running one of the following
commands sometime before your next pull:
```

```
git config pull.rebase false # merge (the default strategy)
git config pull.rebase true  # rebase
git config pull.ff only      # fast-forward only
```

```
You can replace "git config" with "git config --global" to set a default
preference for all repositories. You can also pass --rebase, --no-rebase,
or --ff-only on the command line to override the configured default per
invocation.
```

```
Updating 8ec7fd4..b6839ea
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```





```
<remote> = origin  
<remote_branch> = master
```

Why does the syntax differ?

```
$ git fetch origin master
```

```
$ git merge origin/master
```

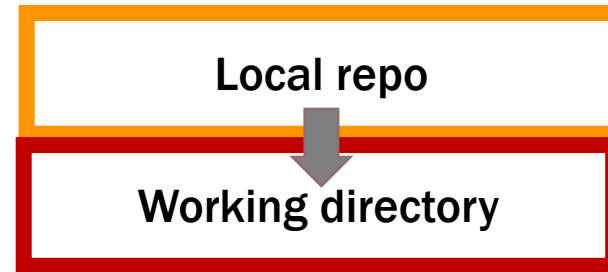
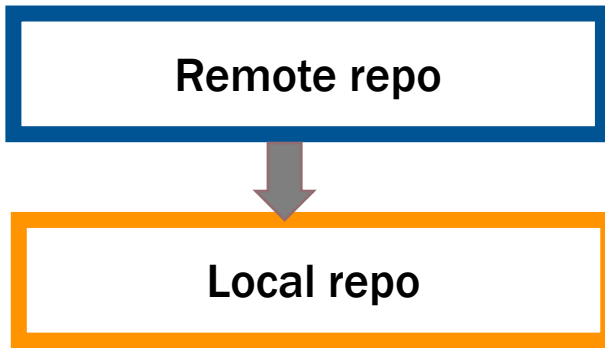
What does main refer to in each case?

`$ git push origin main`

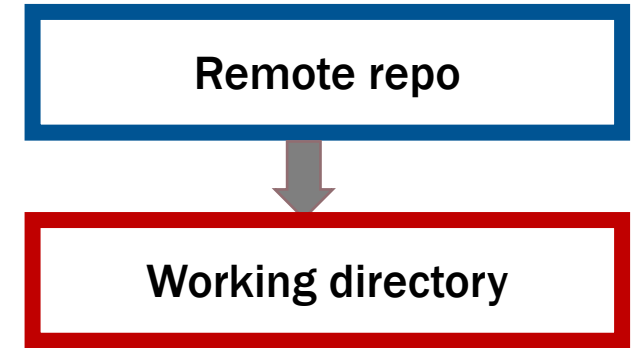
`$ git pull origin main`

<remote> = origin  
<remote\_branch> = master

\$ git fetch origin master    \$ git merge origin/master



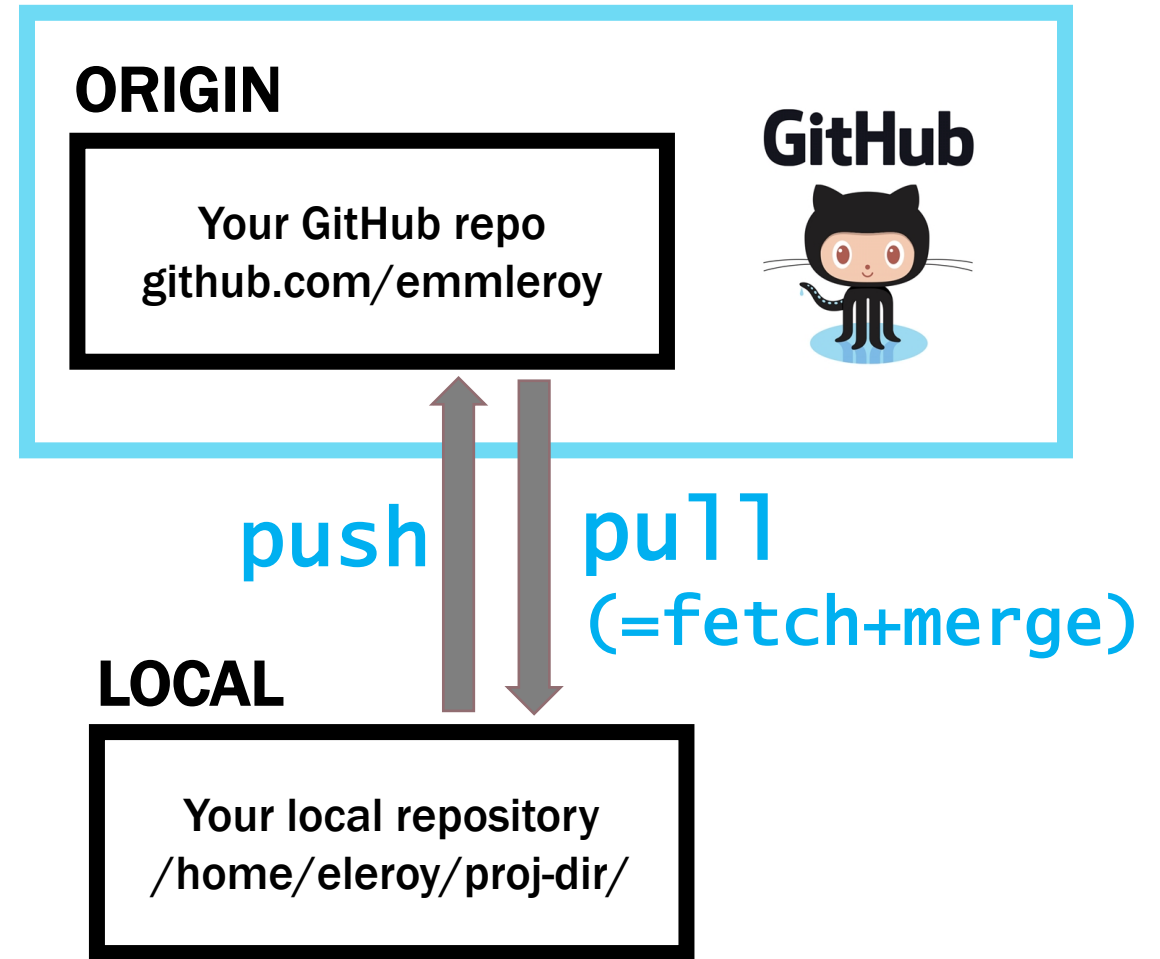
\$ git pull origin master



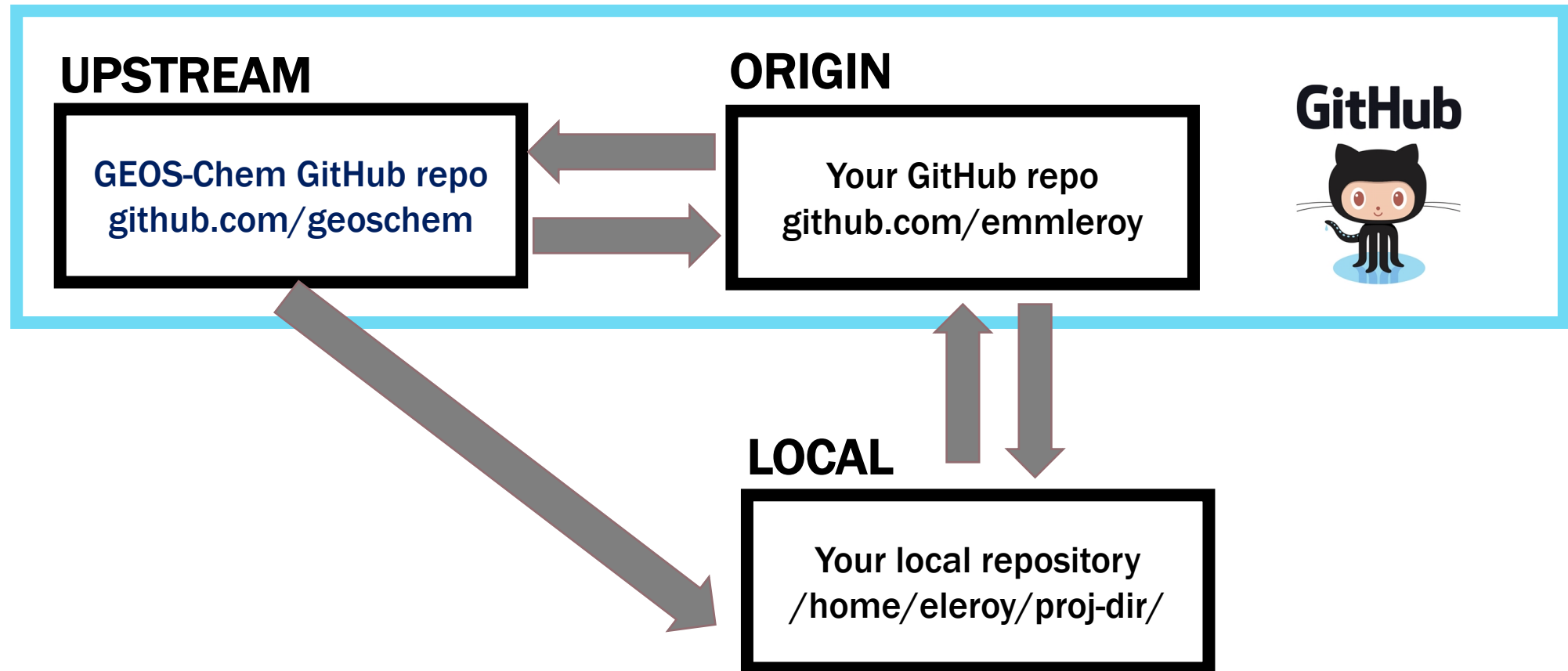
```
$ git pull <remote> <remote_branch>  
or  
$ git fetch <remote> <remote_branch>  
$ git merge <remote>/<remote_branch>
```

# GitHub

Part 1. I have a local project that I want to share with others



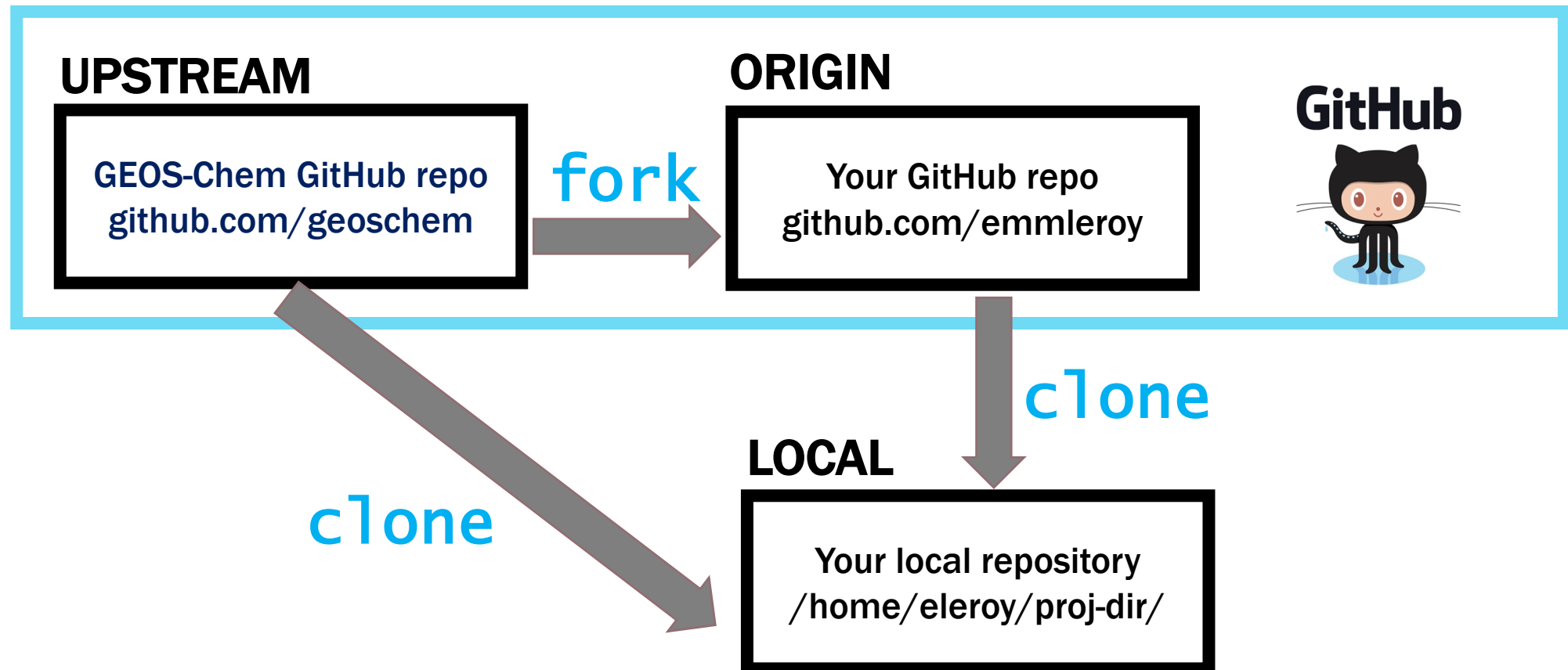
# Part 2. I want to create my own changes to someone else's project!



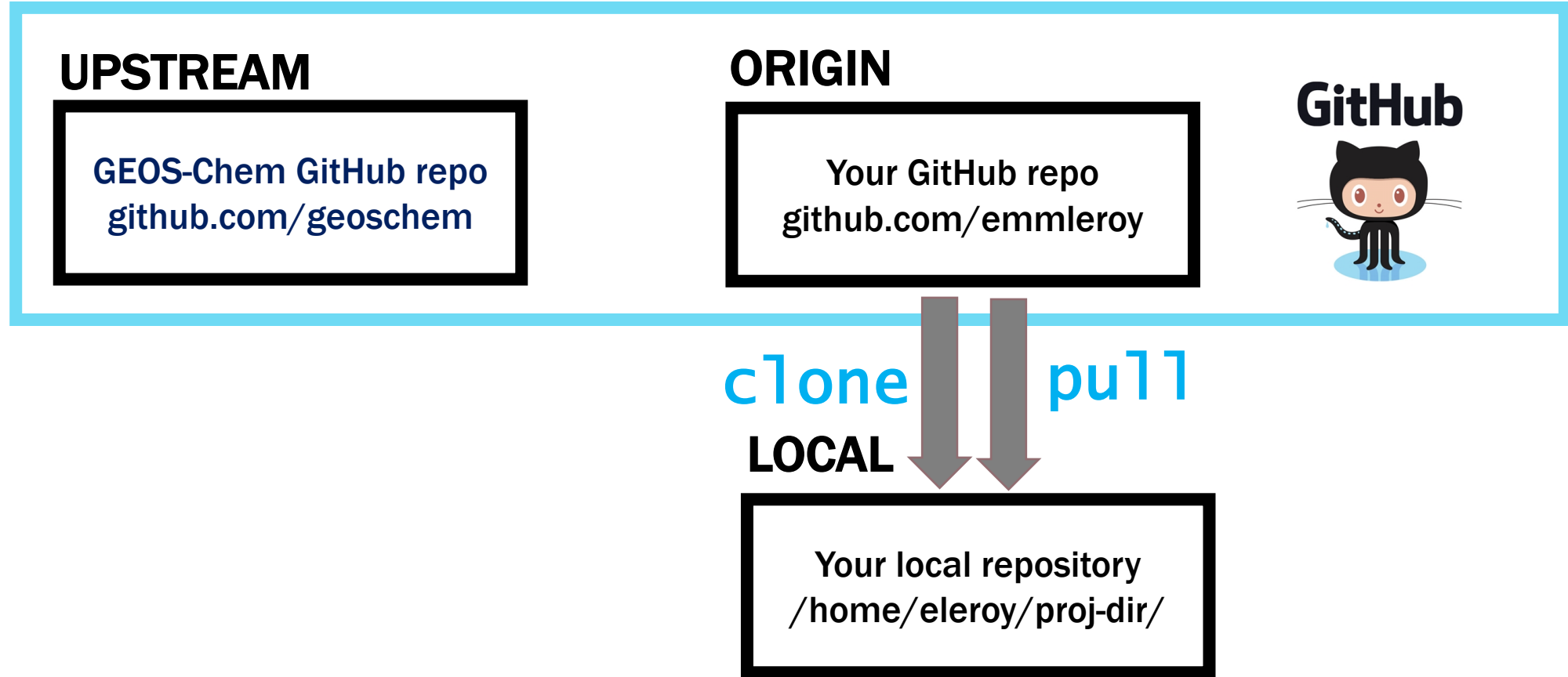
# Forking vs Cloning

A **forked project** is on your online repository

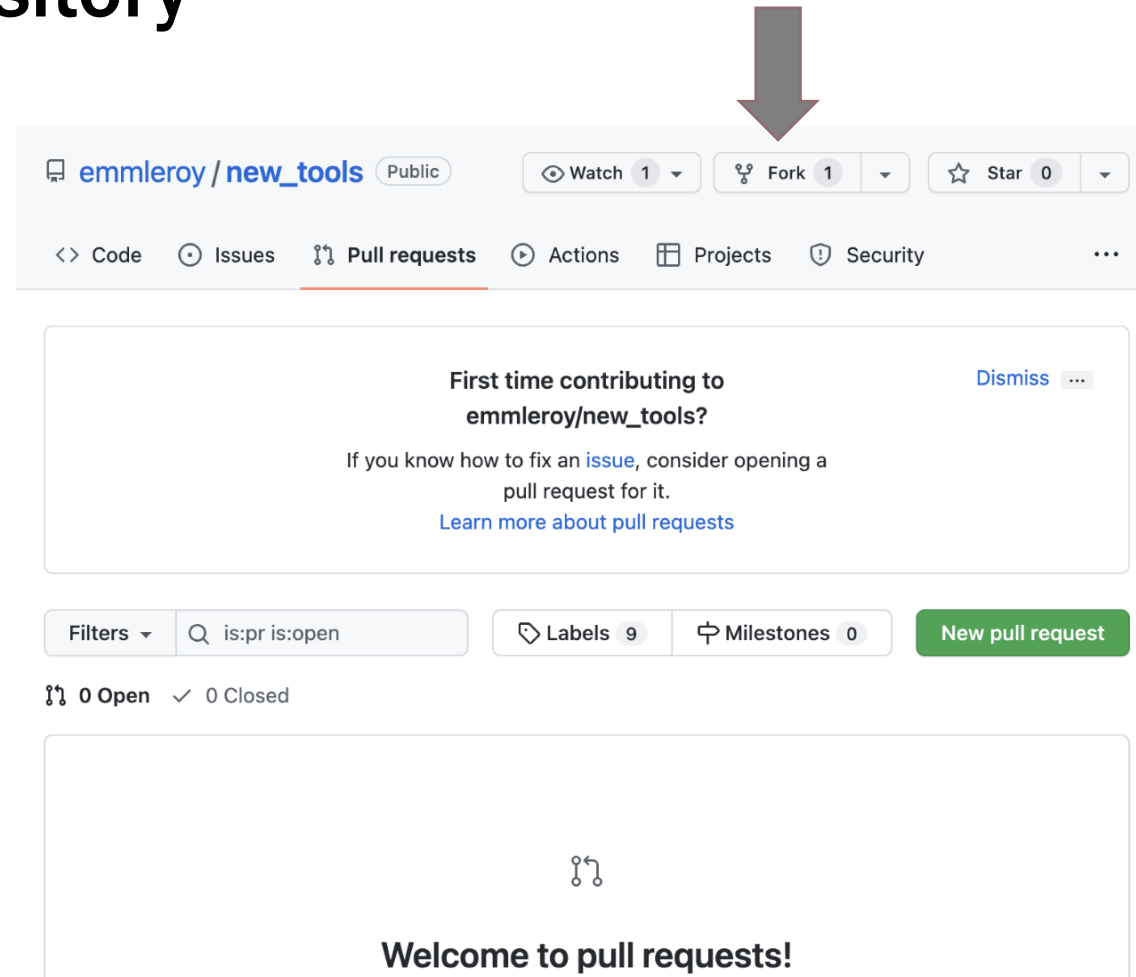
A **cloned project** is on your local machine (you can clone from someone else's remote or from your own remote)



# What's the difference between clone and pull?



# Go to [https://github.com/emmleroy/new\\_tools](https://github.com/emmleroy/new_tools) and fork the repository





## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

---

Owner \*



emmleroy ▾

Repository name \*



new\_tool



By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

example repository for Selin group github tutorial

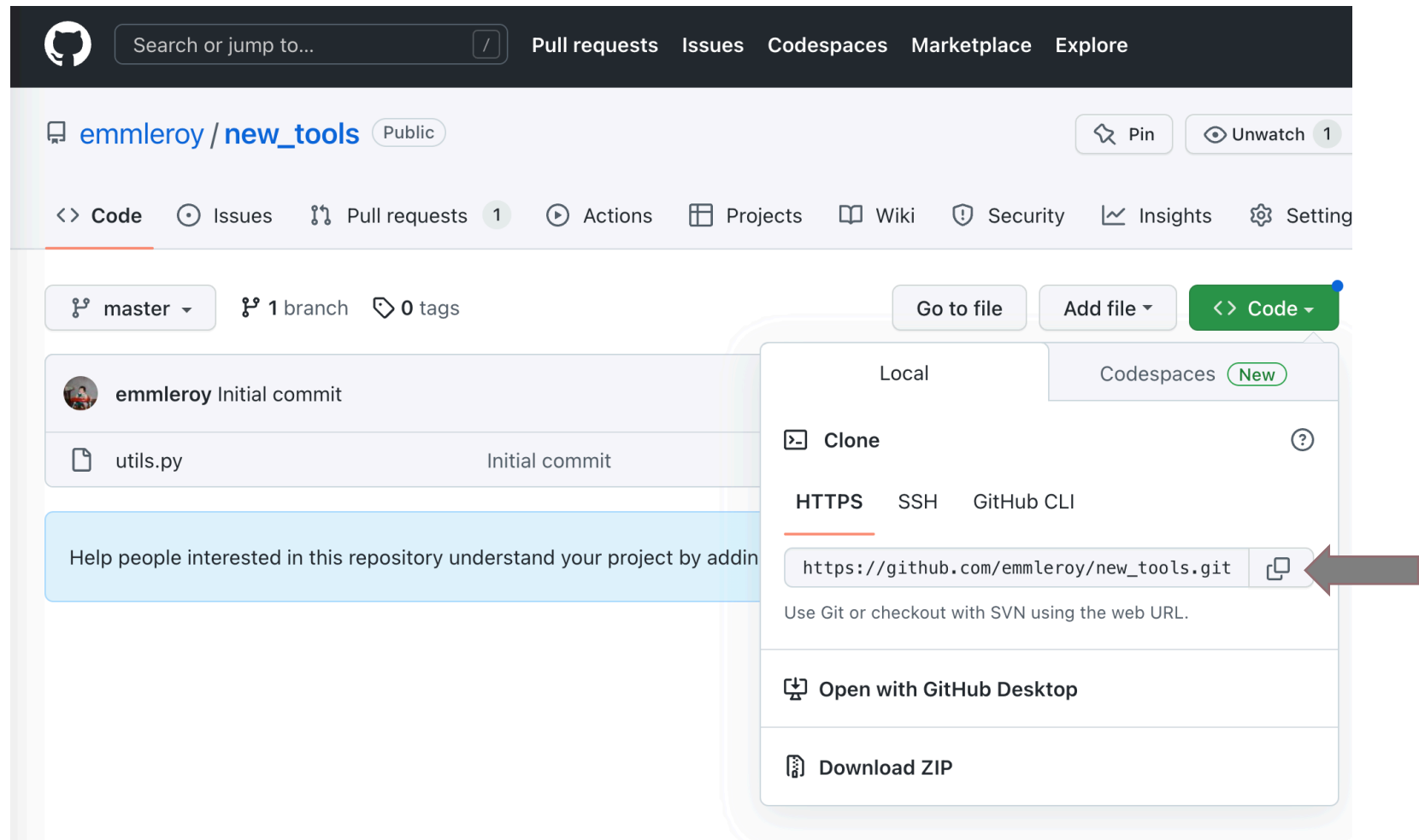
☒ Copy the `main` branch only

Contribute back to NCAR/geocat-comp by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

Create fork

# Copy the https URL of the forked repository



# git clone <remote\_URL>

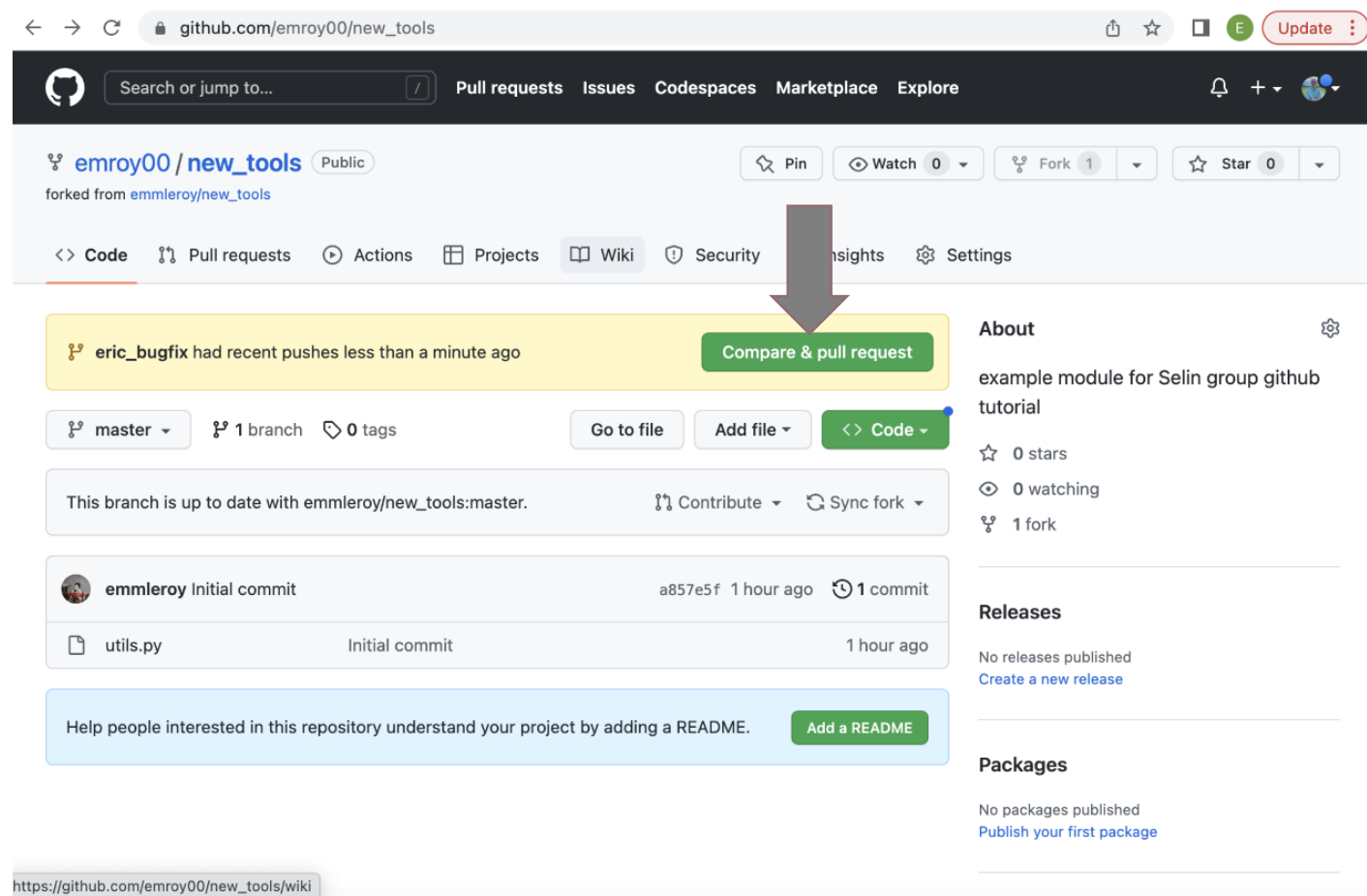
creates a local copy of a remote repository

```
$ cd tutorials/  
$ git clone https://github.com/emmleroy/new\_tools.git  
$ cd new_tools/  
$ git remote -v  
origin      https://github.com/emmleroy/new_tools.git (fetch)  
origin      https://github.com/emmleroy/new_tools.git (push)
```

# Checkout a new branch, make and commit a change, upload the change to your remote forked repository

```
$ git checkout -b Emmie_bugfix
$ echo "# Emmie's contribution" >> utils.py
$ git add
$ git log
$ git commit -m "Add Emmie's contribution"
$ git remote -v
$ git push -u origin Emmie_bugfix
```

# Create a pull request!



The screenshot shows the GitHub interface for a repository named 'emroy00/new\_tools'. The repository is public and was forked from 'emmleroy/new\_tools'. The main navigation bar includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. The repository's main content area shows a comparison between the 'master' branch and a recent push by 'eric\_bugfix'. A large green button labeled 'Compare & pull request' is prominently displayed, with a large grey arrow pointing to it from above. Below this, the repository's file structure is visible, including a file named 'utils.py'. The right sidebar contains information about the repository, including the number of stars (0), watchers (0), and forks (1), as well as sections for Releases and Packages.

github.com/emroy00/new\_tools

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

emroy00 / new\_tools Public

forked from emmleroy/new\_tools

Pin Watch 0 Fork 1 Star 0

Code Pull requests Actions Projects Wiki Security Insights Settings

eric\_bugfix had recent pushes less than a minute ago

Compare & pull request

master 1 branch 0 tags

Go to file Add file <> Code

This branch is up to date with emmleroy/new\_tools:master. Contribute Sync fork

emmleroy Initial commit a857e5f 1 hour ago 1 commit

utils.py Initial commit 1 hour ago

Help people interested in this repository understand your project by adding a README. Add a README

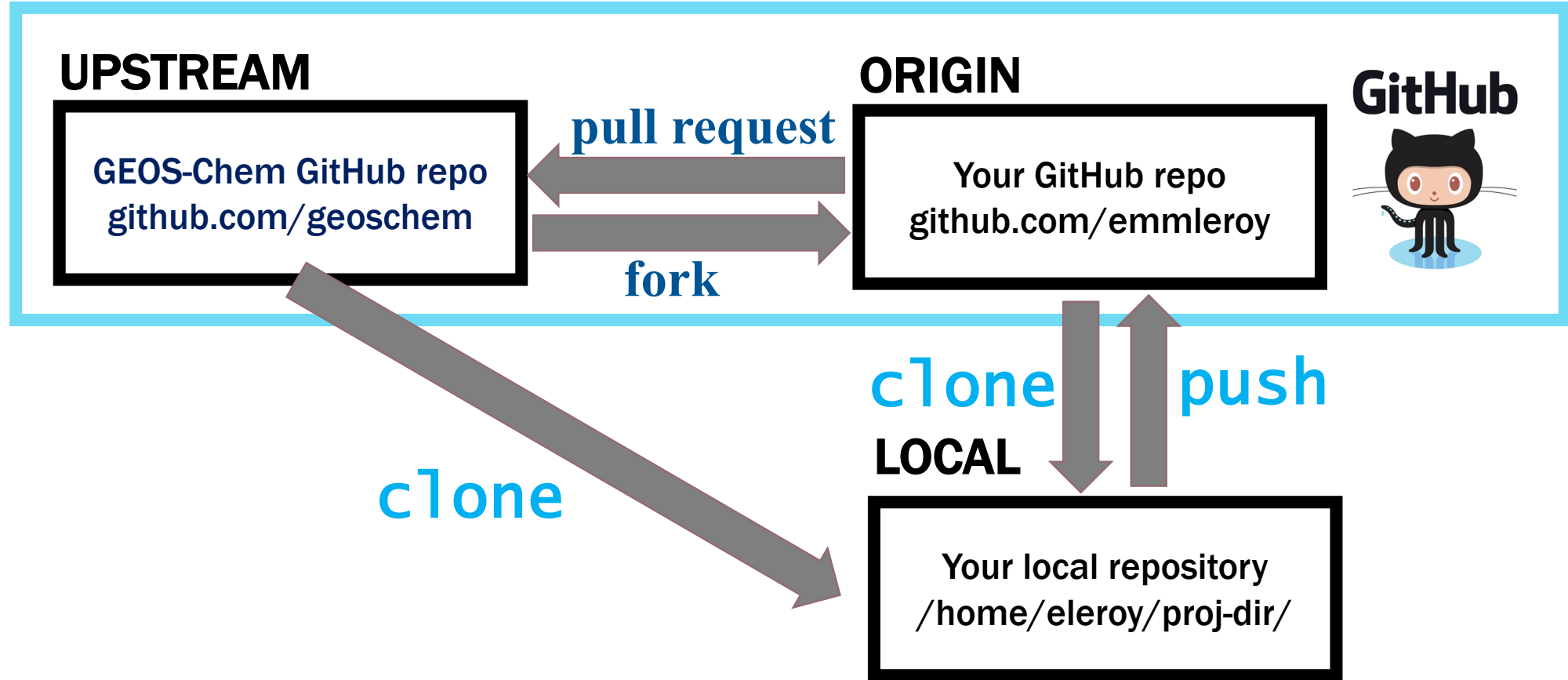
About example module for Selin group github tutorial

0 stars 0 watching 1 fork

Releases No releases published Create a new release

Packages No packages published Publish your first package

https://github.com/emroy00/new\_tools/wiki



```
$ git config --global user.name "Emmie Le Roy"  
$ git config --global user.email "eleroy@gmail.com"
```