Git workflows for physicists

Using version control in your research

Have you done the following?

- Commented out code?
- Emailed code to someone?
- Avoided changing code so you don’t break it?
- Regretted making a change to your code?
- Appended version numbers to files (e.g. fast_code_v1.py, fast_code_v2.py,…)?
- Forgotten why you added or changed code??
- Needed to work across multiple machines?

There’s a better way!

Version Control System

AKA a **Source Code Manager (SCM)** or **Revision Control System (RCS)**

VCS/SCM/RCS: A software used to keep track of changes to files. Used by development teams to collaborate on code. Can be used by individuals to keep track of projects.

Examples of VCS Softwares

Centralized: code is stored on central server

- Concurrent Versions Systems (CVS)
- Apache Subversion (SVN)
- Perforce

Distributed: All developers get a copy

- Mercurial
- GNU Bazaar
- Piper (Google)
- Git

Reasons to use git

- Distributed: Easy to implement local workflows
- Popular: Most used VCS in software companies
- Open Source Ecosystem: Github makes it easy to collaborate on small projects.
- Resume: Github Profile is like a LinkedIn profile.

Git may not be the best VCS for your use case,
But it is the best to learn at this point in time.

Overview of Presentation

PART I

- `git init`
- `git status`
- `git add`
- `git commit`
- `git log`
- `git diff`
- `git restore`
- `Linear Workflow`

PART II

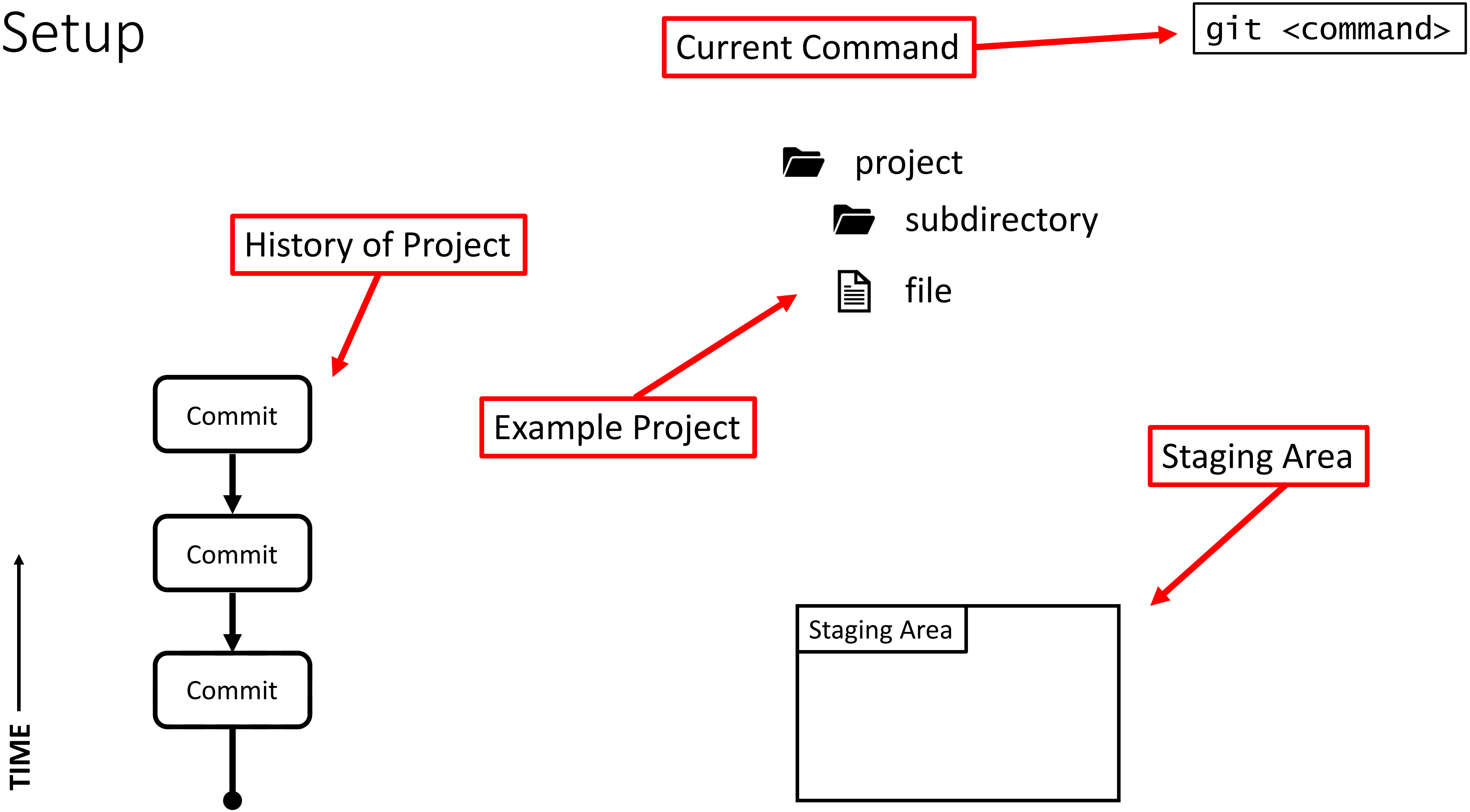
- `git branch`
- `git switch`
- `git merge`
- `git rebase`
- `Development Workflow`

PART III

- `git init --bare`
- `git remote`
- `git fetch`
- `git push`
- `Self-Collaboration Workflow`

> The **best way to learn git** is probably to first only **do very basic things** and not even look at some of the things you can do until you are confident. – Linus Torvalds

Setup



Current Command

```
git <command>
```

project

subdirectory

file

History of Project

Commit

Commit

Commit

TIME

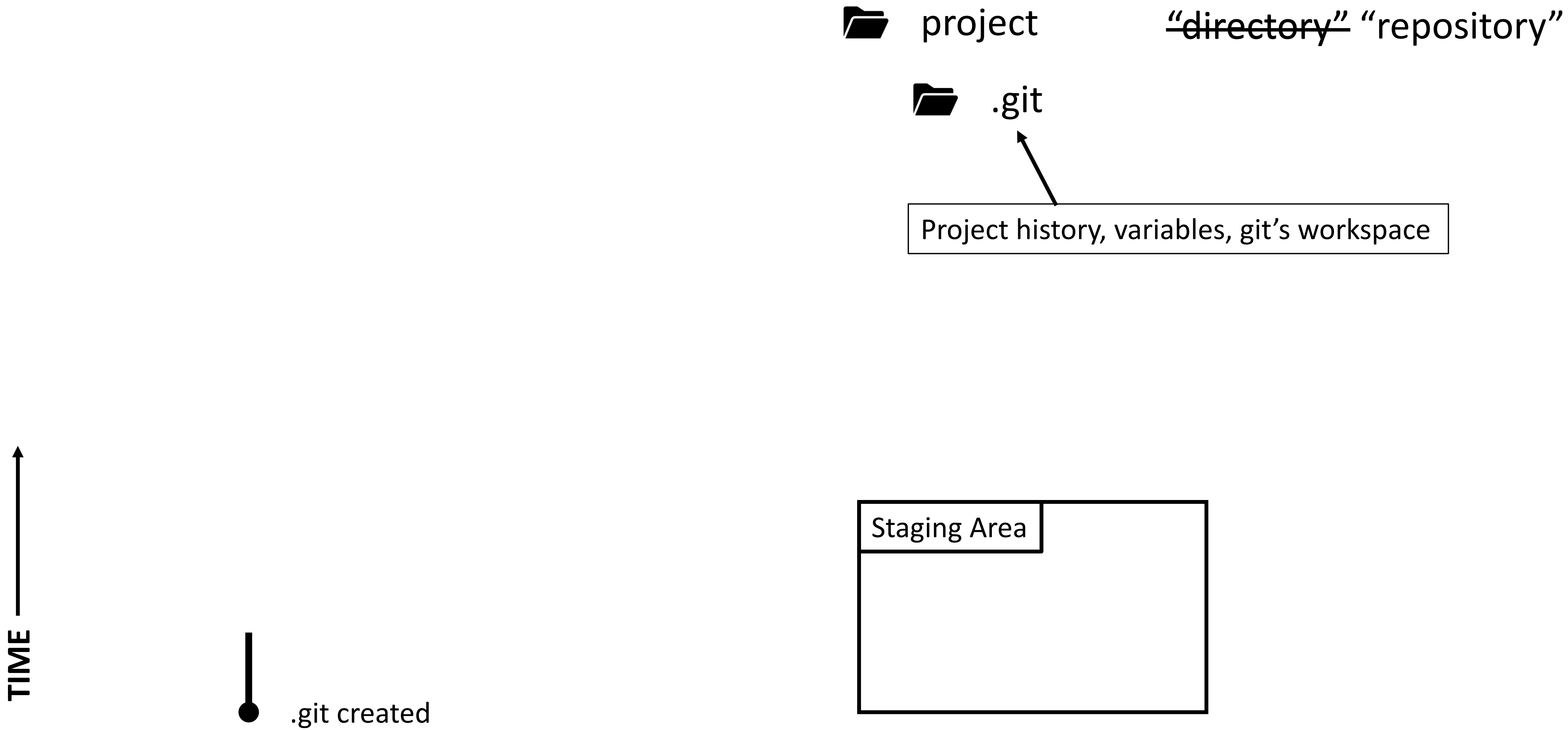
Example Project

Staging Area

Staging Area

Initializing a Repository

`git init`



project

~~“directory”~~ “repository”

.git

Project history, variables, git’s workspace

Staging Area

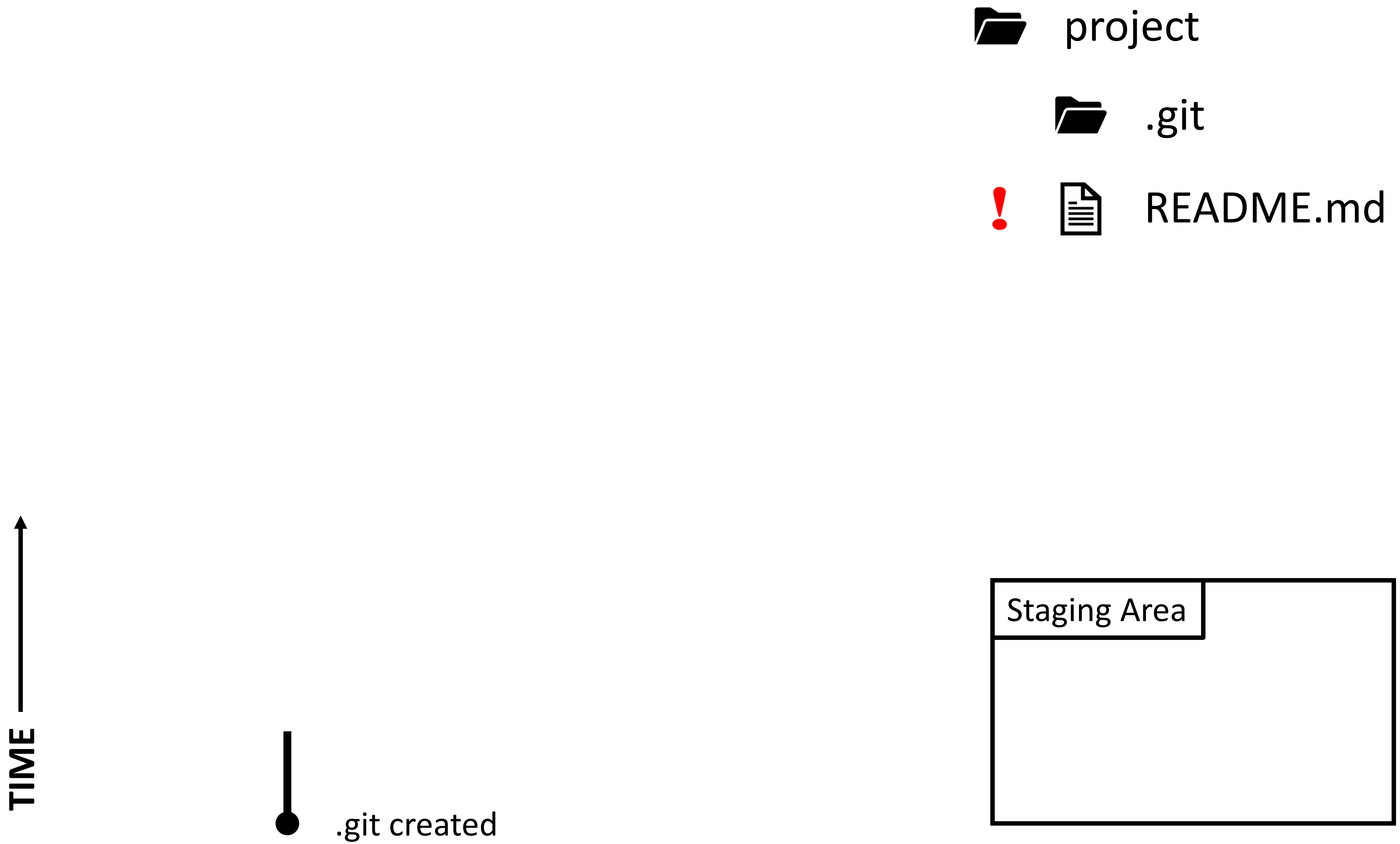
TIME

.git created

Checking Status

```
git status
```

TIP: Run git status often



project

.git

! README.md

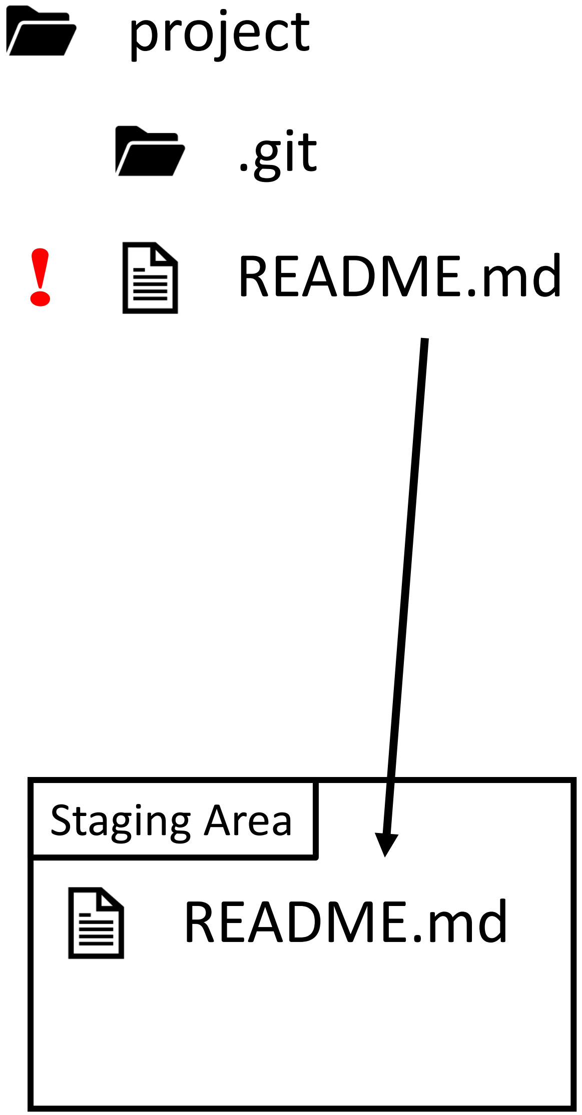
Staging Area

TIME

.git created

Staging Changes

```
git add
```



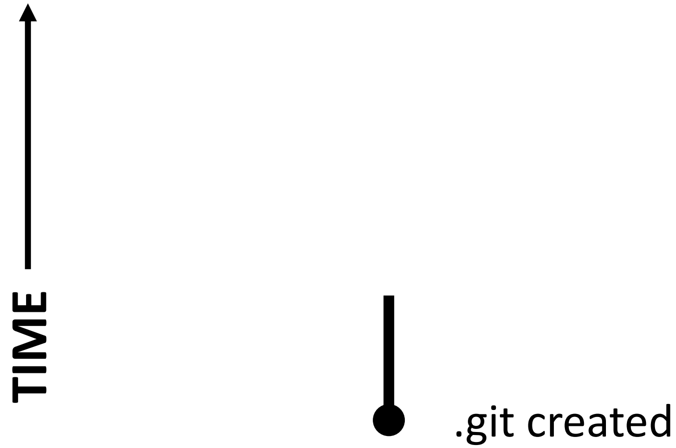
project

.git

! README.md

Staging Area

README.md



TIME

.git created

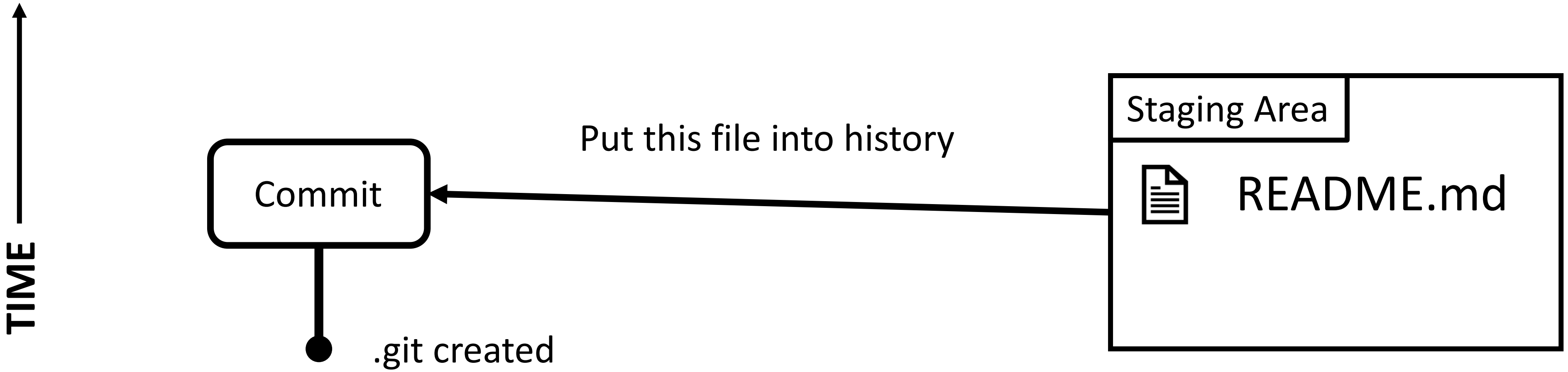
Commits

`git commit`

TIP: Present tense is considered professional in commit messages

TIP: Write your commit messages like it's for someone else.

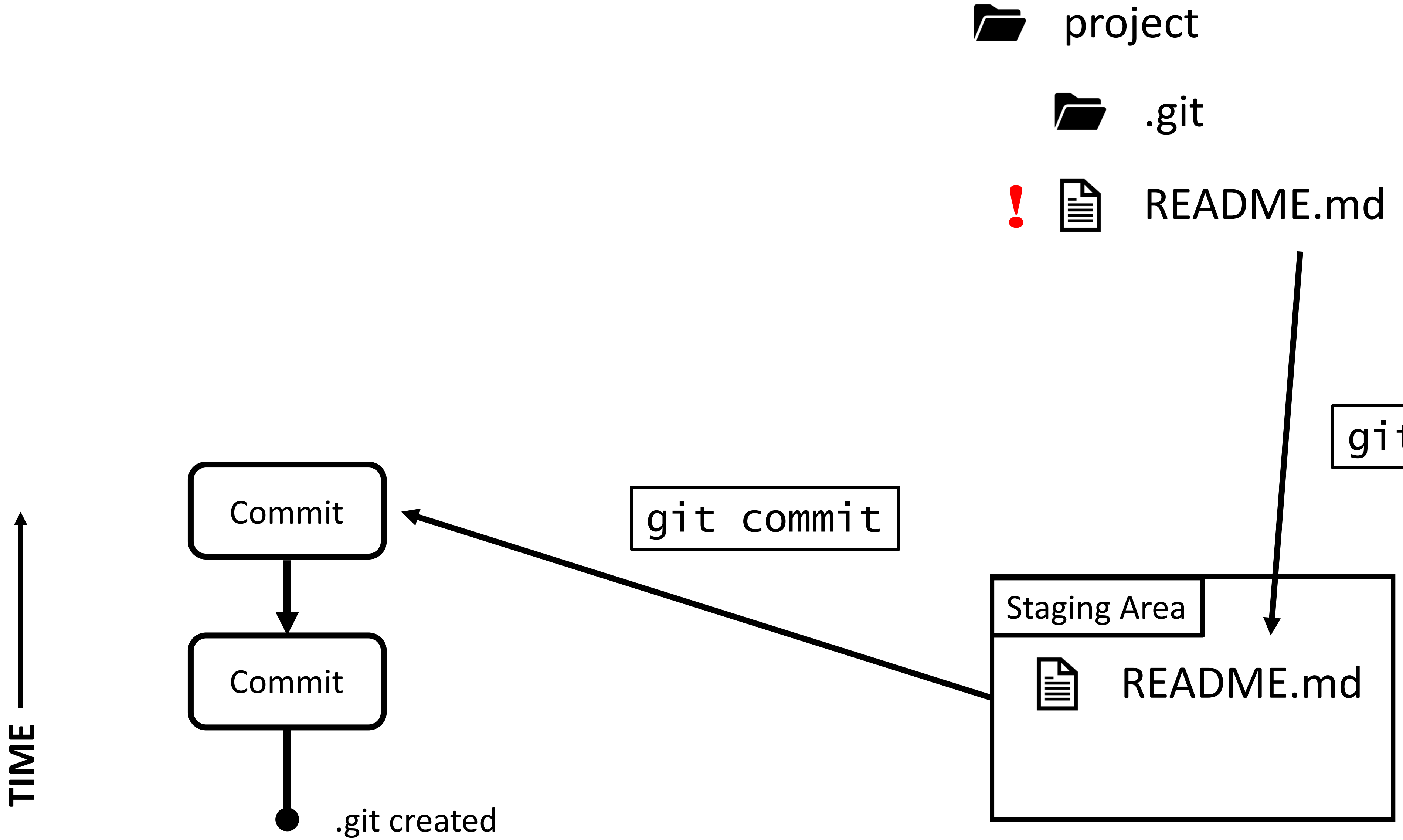
- project
  - .git
  - README.md



Viewing Changes

`git diff`

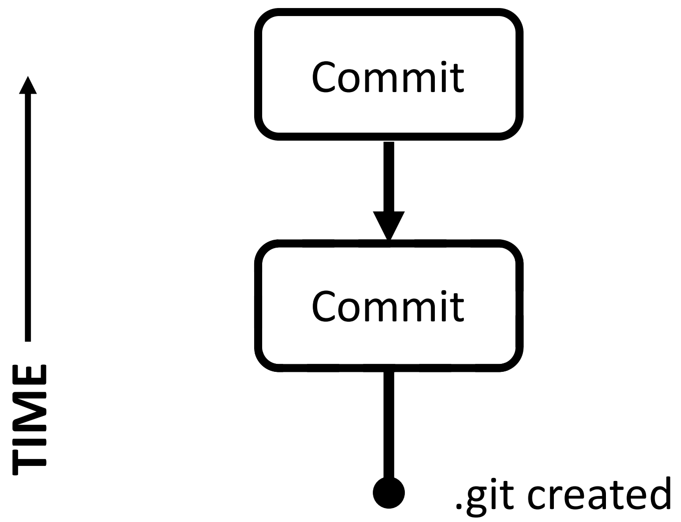
TIP: Run git diff to catch any unwanted changes you made (e.g. extra whitespace, print statements)

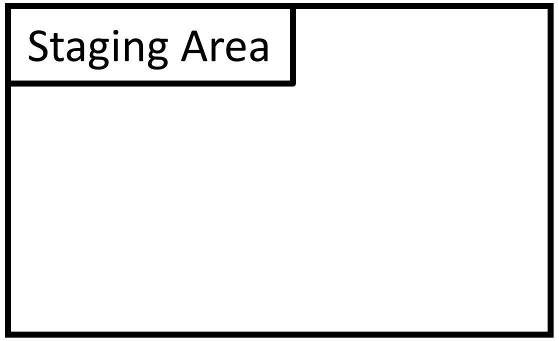


Commit History

```
git log
```

- project
  - .git
  - README.md

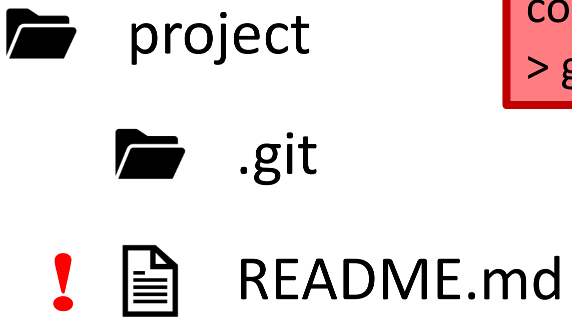


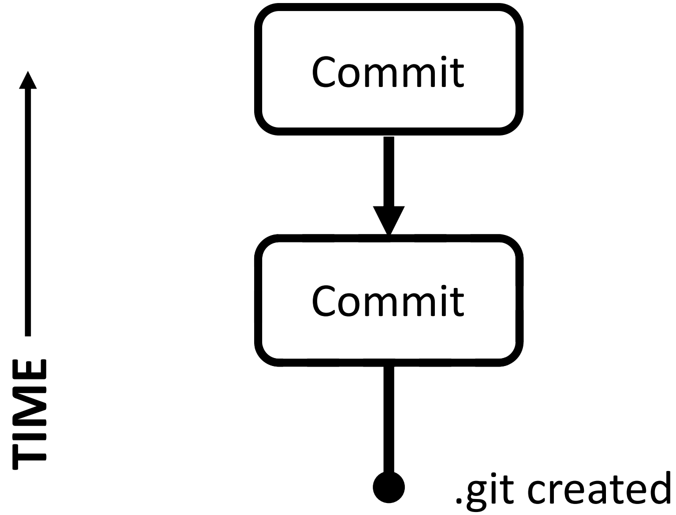


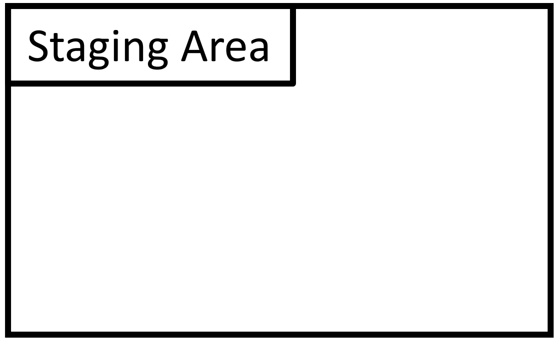
Undo Changes

```
git restore
```

WARNING: Before 2.23, this command was
> git checkout -- <file>







Workflow: Linear

Linear Workflow

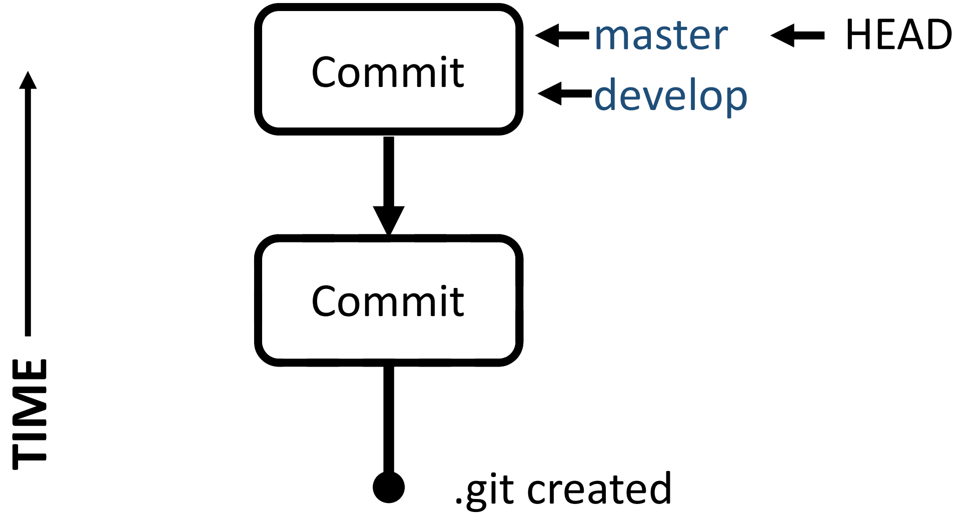
USE CASE: Project is simple (no risk of breaking anything) but you want to be able to undo changes if needed. Avoid losing code that "just worked".

	1. Initialize Repository	git init
Repeat	2. View project history to understand current state	git log
	3. Check that your directory is clean	git status
	4. Make changes or add files	Use any other software
	5. Check your changes	git status, git diff
	6. Stage changes	git add <file>
	7. Commit changes	git commit –m "message"

Creating Branches

`git branch`

- project
  - .git
  - README.md

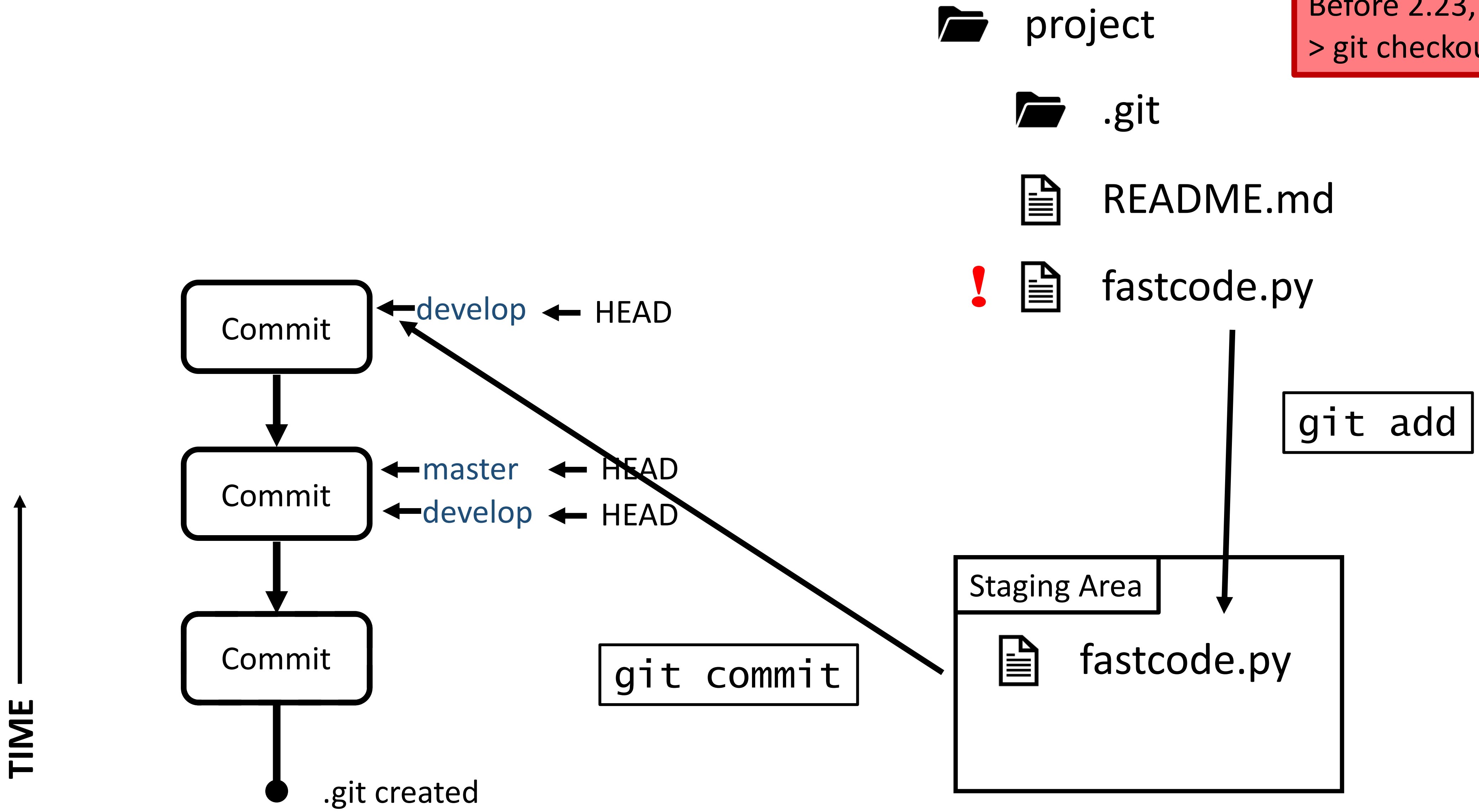


Staging Area

Check out Branches

```
git switch
```

WARNING
Before 2.23, this command was
> git checkout <branchname>

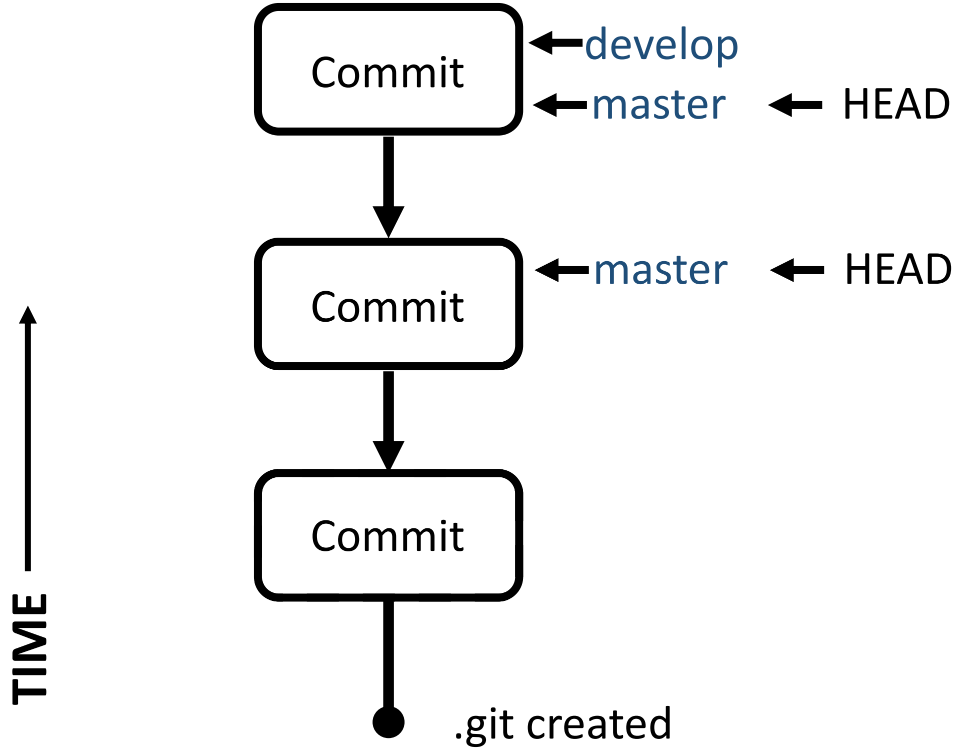


```
git add
```

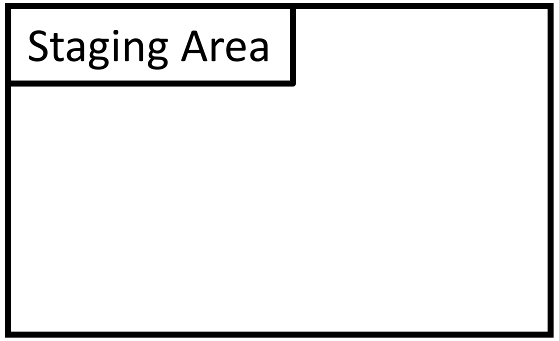
```
git commit
```

Merging Branches: Fast Forward

`git merge`



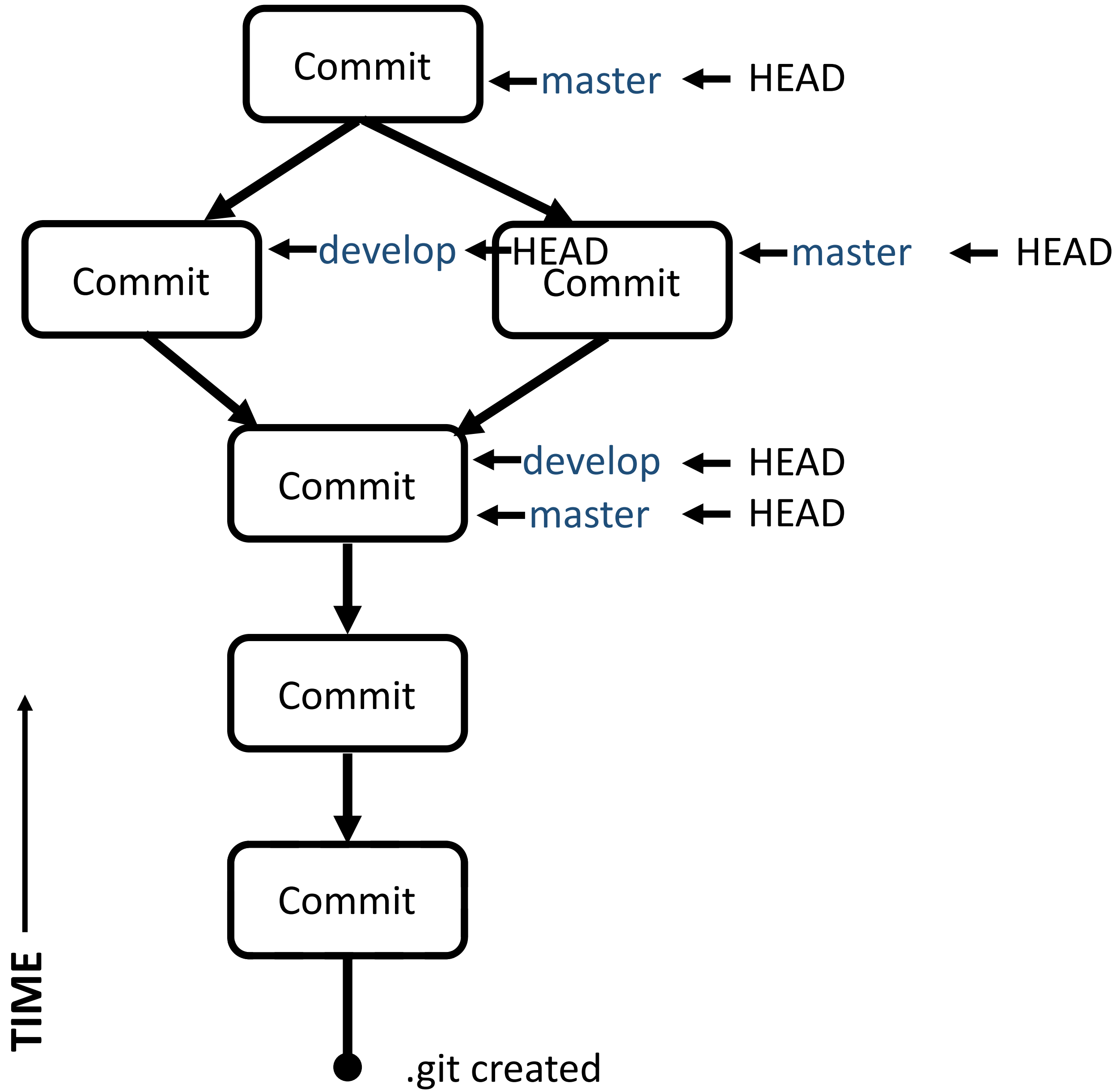
- project
  - .git
  - README.md
  - fastcode.py



Merging Branches: Three-way Merge

```
git merge
```

TIP: Switch to the branch where you want the code before merging.

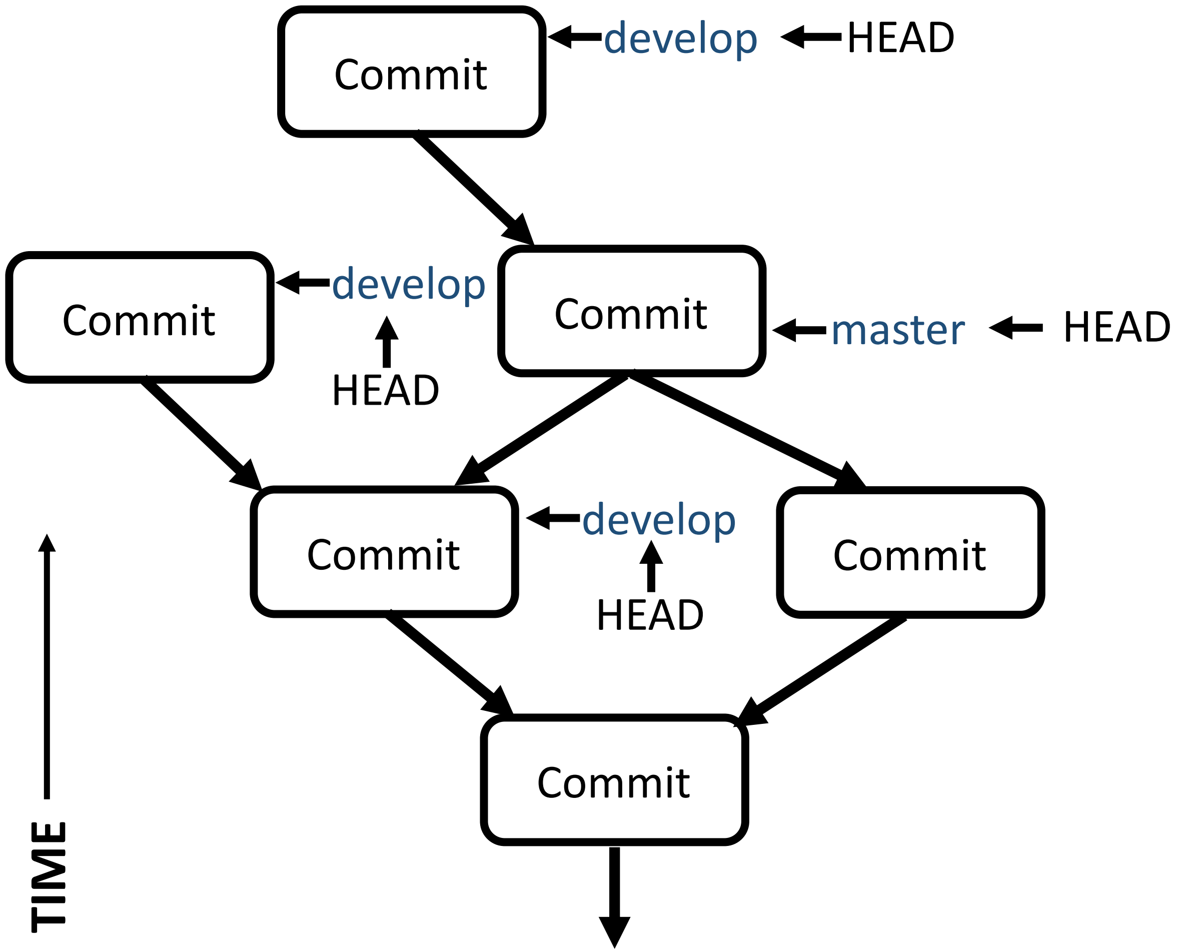


project
.git
README.md
fastcode.py

Staging Area

Rebasing Branches

`git rebase`



- project
  - .git
  - README.md
  - fastcode.py

Staging Area

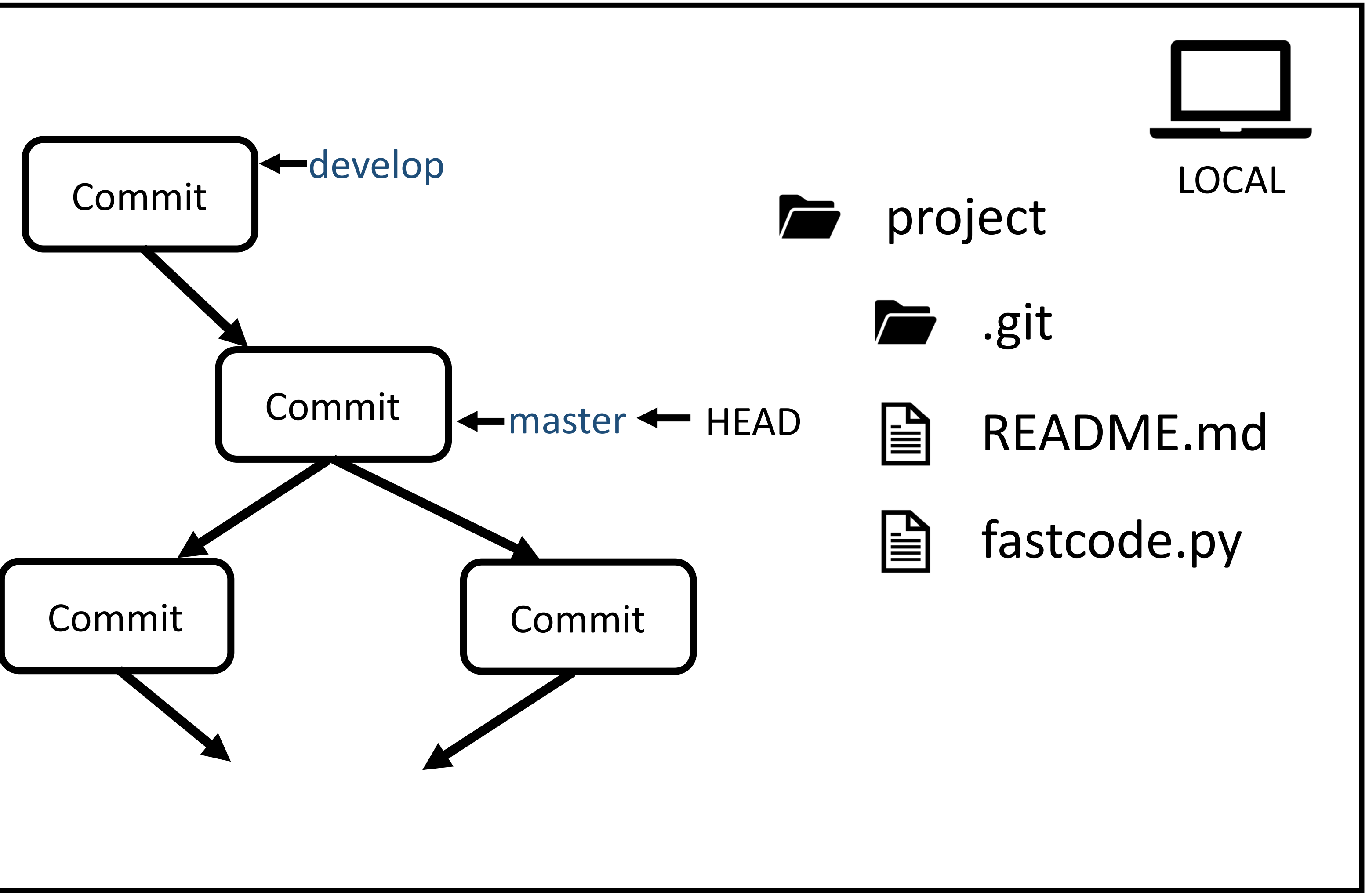
Workflow: Development

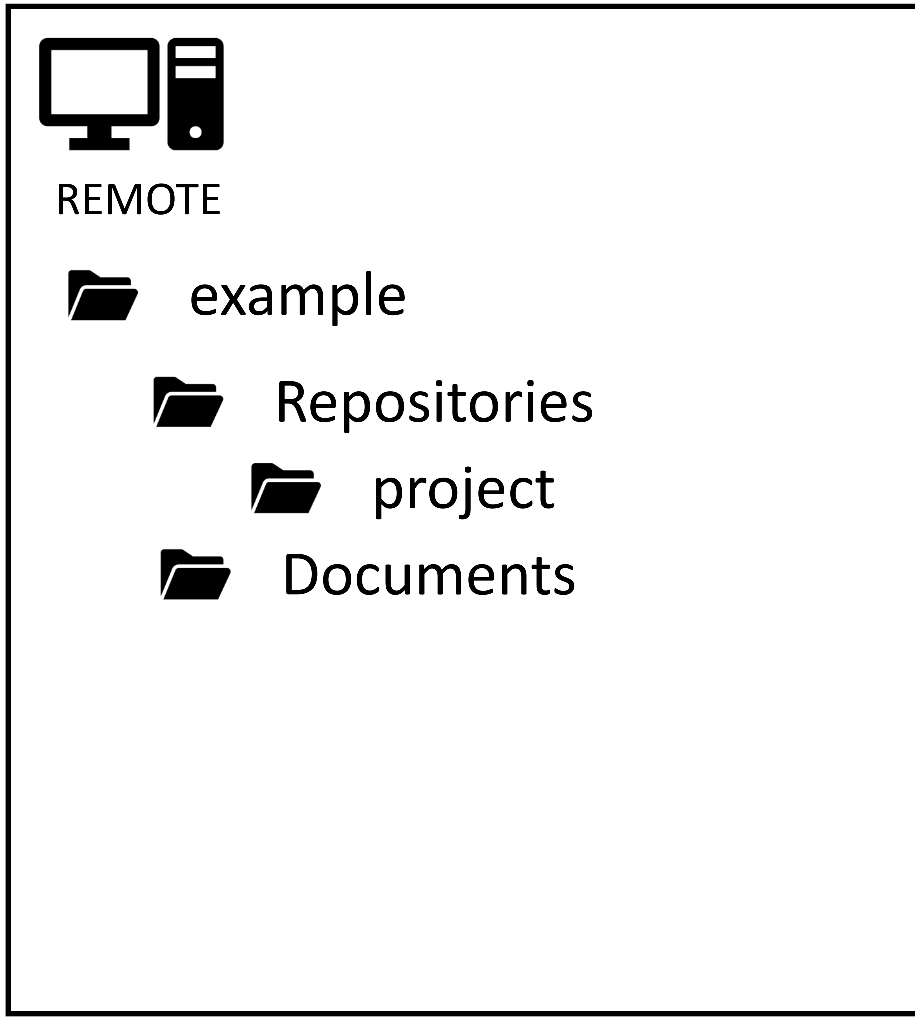
USE CASE: You need to run code and develop it at the same time (e.g. your advisor likes your analysis but you have some new ideas). Your working code is always available on the master branch.

1. Create a develop branch	Git branch develop
2. Switch to the develop branch	Git switch
3. Develop your code on develop branch	Linear Workflow
4. Occasionally rebase onto the master	Git rebase
5. Switch to master	Git switch master
6. Merge code from develop	Git merge develop

Bare Repositories

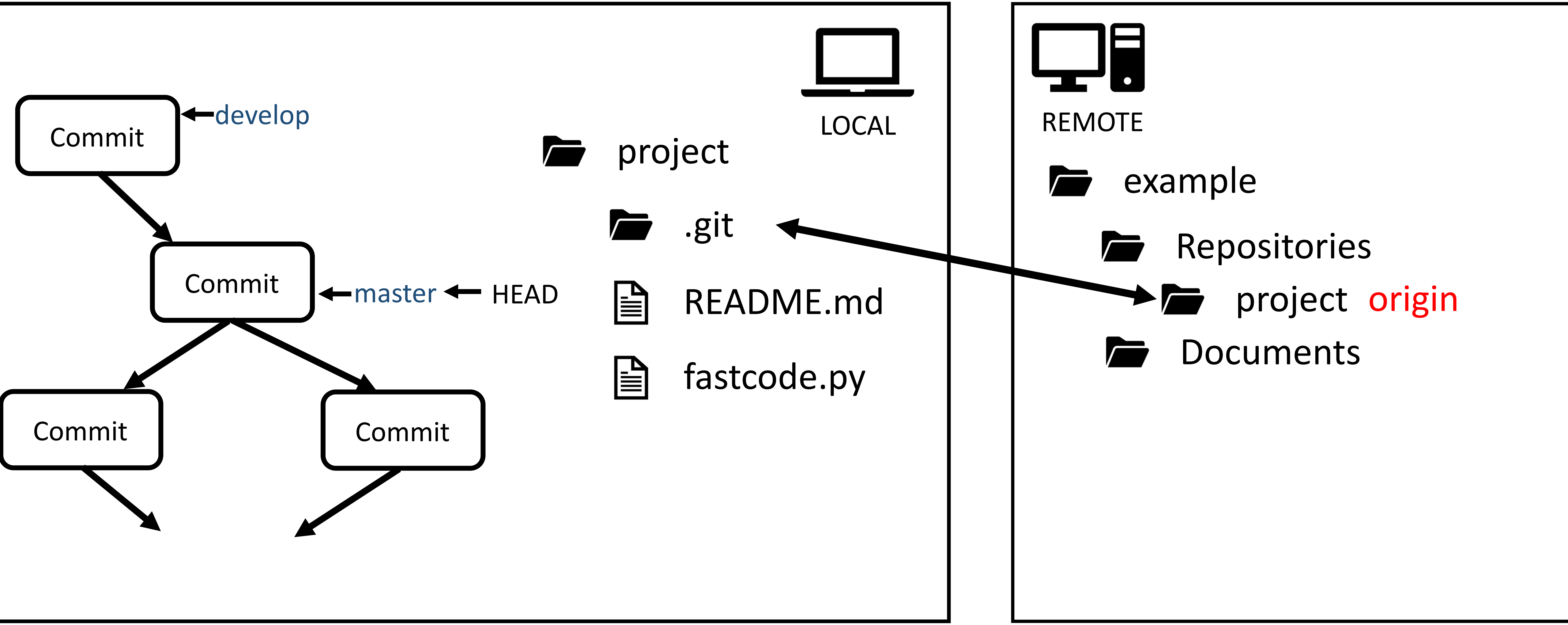
```
git init --bare
```





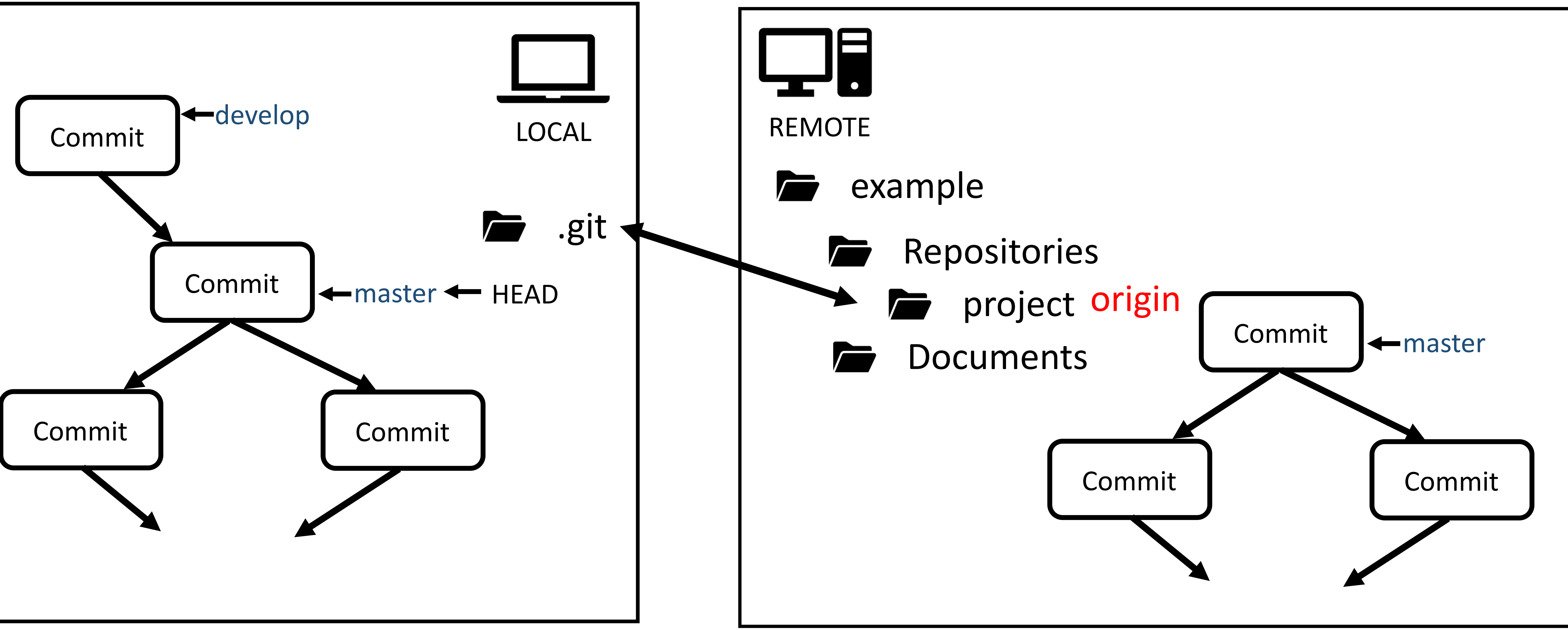
Remote Repositories

`git remote`



Pushing

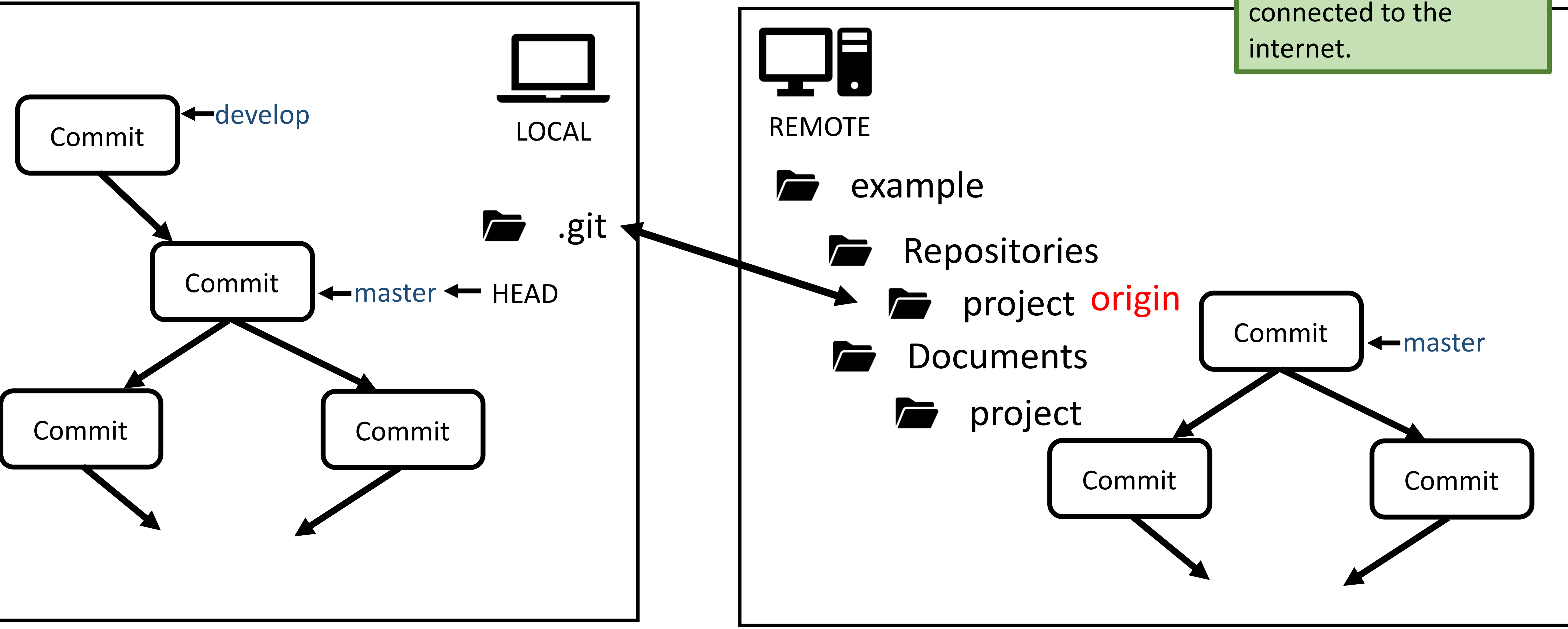
```
git push
```



Cloning

`git clone`

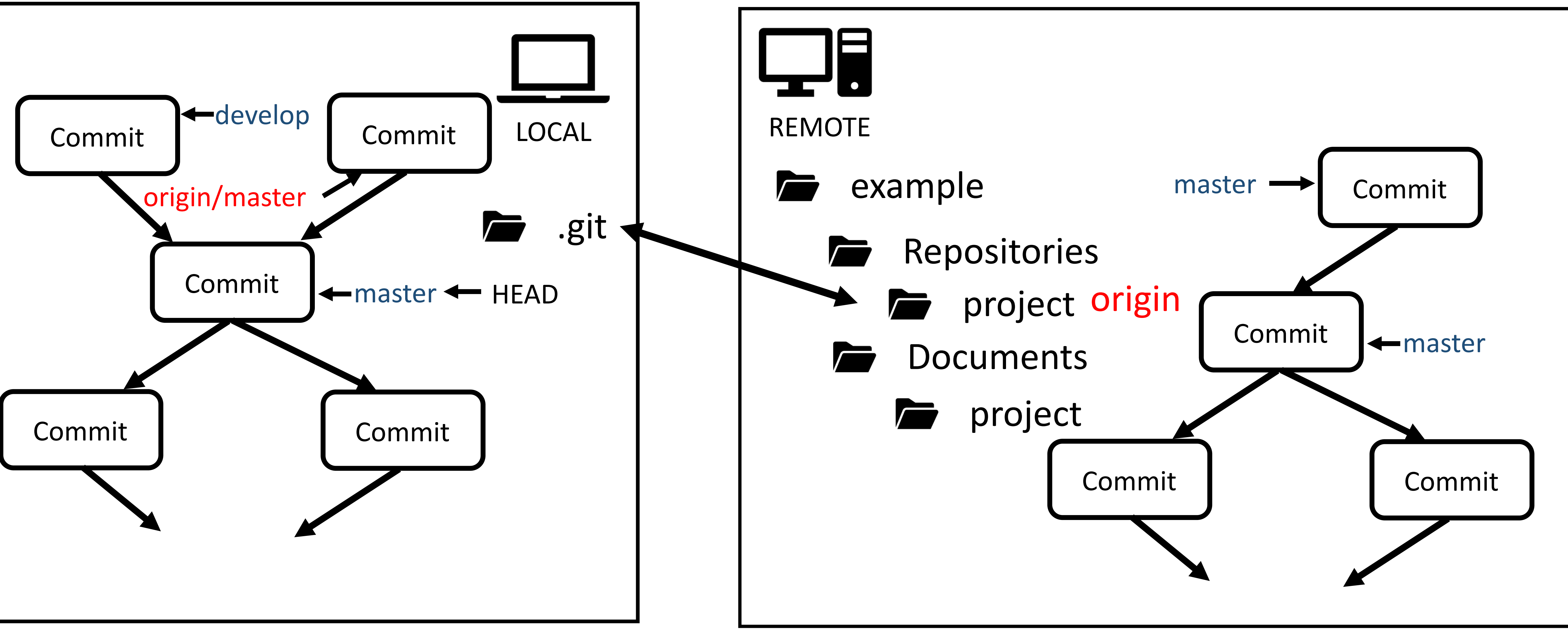
TIP: You can clone from any computer that's connected to the internet.



Fetching

WARNING
"git pull" is an overused command that will run "git fetch" then "git merge".

```
git fetch
```



Workflow: Self-Collaboration

USE CASE: Working on code across multiple machines. Develop code on your laptop, push to a personal server or Github, pull to high-performance machine for running.

	1. Get a server or Github account	
	2. Initialize a remote repo	git init –bare (Not used on Github)
Repeat	3. Develop code locally	Development workflow
	4. Push the master branch	git push origin master
	5. Pull to High-Performance Machine (HPM)	git pull
	6. Fix errors that need testing on HPM	
	7. Push any changes made on HPM	git push
	8. Pull to local development machine	git pull

Github vs. Personal Server

Github or Personal Server?

For ease of use and virtually guaranteed data integrity, use Github and follow this page. To keep code private, use a personal server.

For a **personal server**, get a desktop from Dan Bradley and install Ubuntu Server on it.

Overview of Presentation

PART I

- `git init`
- `git status`
- `git add`
- `git commit`
- `git log`
- `git diff`
- `git restore`
- `Linear Workflow`

PART II

- `git branch`
- `git switch`
- `git merge`
- `git rebase`
- `Development Workflow`

PART III

- `git init --bare`
- `git remote`
- `git fetch`
- `git push`
- `Self-Collaboration Workflow`

> The **best way to learn git** is probably to first only **do very basic things** and not even look at some of the things you can do until you are confident. – Linus Torvalds

Installing Git

Operating System	Instructions
Linux (RedHat)	sudo dnf install git-all
Linux (Debian)	sudo apt install git-all
MacOS	git --version
Windows	Download Git for Windows

If that doesn't work, look here:
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Resources and Tips

```
> git <command> --help
```

Command to bring up documentation.
Can be difficult to read if inexperienced.

☆ Git SCM: https://git-scm.com/book/en/v2

Full explanations of git commands in a readable format.

10 Years of Git: An Interview with Git Creator Linus Torvalds

Great interview providing historical background and motivation for git.
Can answer a lot of “Why?” questions you may have.

Learn Git Branching

Web application offering hands-on practice with git branching.

TIP: Run git status often

TIP: Present tense is considered professional in commit messages

TIP: Write your commit messages like it’s for someone else.

TIP: Run git diff to catch any unwanted changes you made (e.g. extra whitespace, print statements)

TIP: Switch to the branch where you want the code before merging.

Questions?