git
An Introduction to Time Travel (and version control)

What is Version Control?

- It's like a time machine for your code
  - With fewer grandfather paradoxes and more merge conflicts
- Version control software can…
  - Maintain the history of all your changes
  - Travel between branches for different features
  - Compare and combine different versions of your code

The Goal:

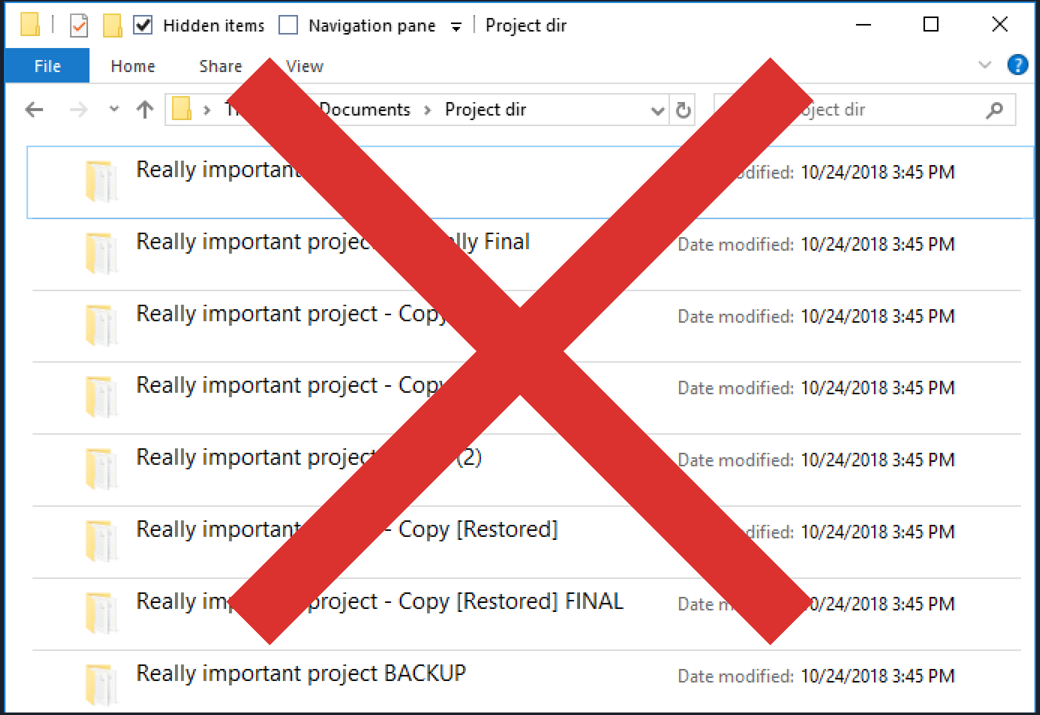
Never write the same code twice!

The Goal:

Never write the same code twice!

Because writing code is hard.

Version Control Alternatives



Really importan
Date modified: 10/24/2018 3:45 PM

Really important projec lly Final
Date modified: 10/24/2018 3:45 PM

Really important project - Copy
Date modified: 10/24/2018 3:45 PM

Really important project - Copy
Date modified: 10/24/2018 3:45 PM

Really important project (2)
Date modified: 10/24/2018 3:45 PM

Really important - Copy [Restored]
Date modified: 10/24/2018 3:45 PM

Really im project - Copy [Restored] FINAL
Date modified: 10/24/2018 3:45 PM

Really important project BACKUP
Date modified: 10/24/2018 3:45 PM

Version control options

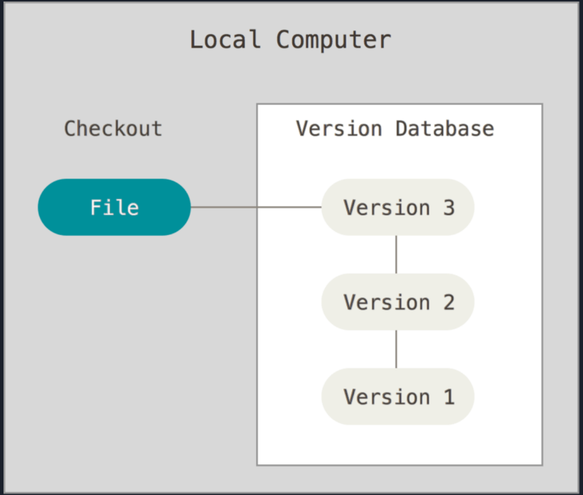
git

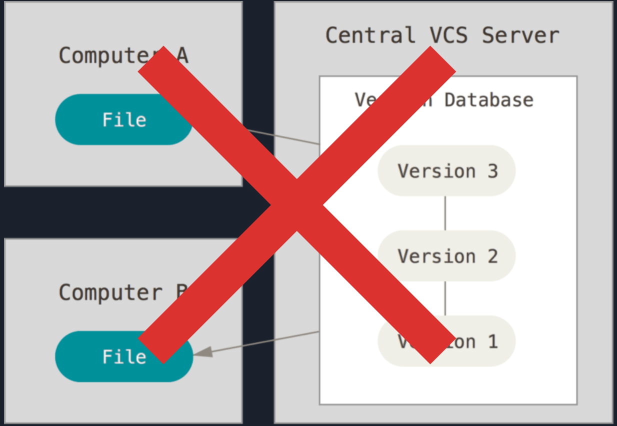
mercurial

SUBVERSION

What is Git?

- Form of **distributed** version control
- Stores the entire project (repository) **locally**





What is GitHub?

- GitHub is a website for sharing and collaborating on Git repositories
- You don't need to use GitHub to use Git
- Git runs locally

git ≠ GitHub

Downloading Git

- Download git (right now!) from:
  - https://git-scm.com/
- Git can be used from the terminal
  - Use the "Git Bash" application on Windows
  - All commands start with "git"
  - Ex: `git status`
  - Get help with: `git help [command name]`

Demo: init and status

git init

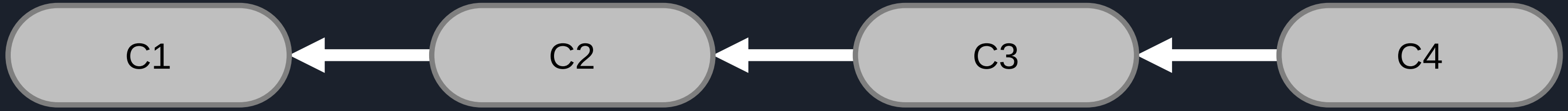
- Run `git init` to create new repository
- Creates a .git folder
    - Used to store the entire project history
    - Stores additional state (like what part of the timeline you are looking at)

git status

- `git status` displays a bunch of information about git's current state
- Lists which files have been modified and your current branch (more on that later)

Commits

- Snapshots of your working directory
- Record the state of all "tracked files"
- Commits are referred to by their "hash"
  - A deterministically generated unique identifier
- Each commit points to the commit(s) preceding it
- Commits are almost never edited/deleted



Demo: git commit, git diff

Making commits

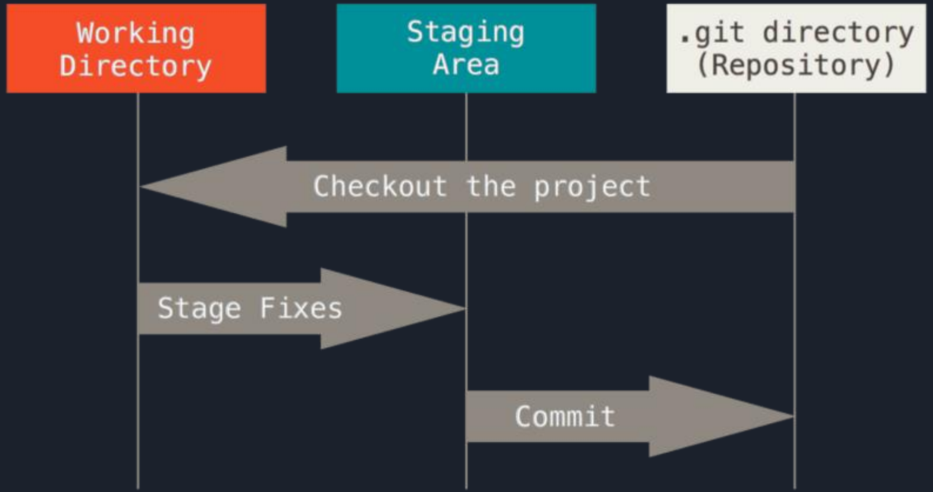
1. Edit/create some files
2. "Stage" the changes with `git add path/to/file`
3. Commit the changes with `git commit`
4. Enter a commit message and save
5. Repeat

Shortcuts

- `git commit -a` (Stage all changes and commit)
- `git commit -m "Commit Message"`
- `git commit --amend` (Edit previous commit)
- `git add –A` (Add all files)

The Staging Area

- A temporary space for preparing a commit
- Changes are staged with `git add`

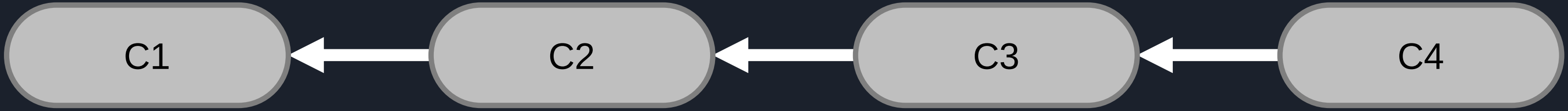


git diff

- `git diff` shows which lines have been modified since your last commit
- `git diff path/to/file` compares a specific file

Summary so far

- We can now create a linear timeline for our project
- But how do we access this data?



Demo: git log, git checkout

git log

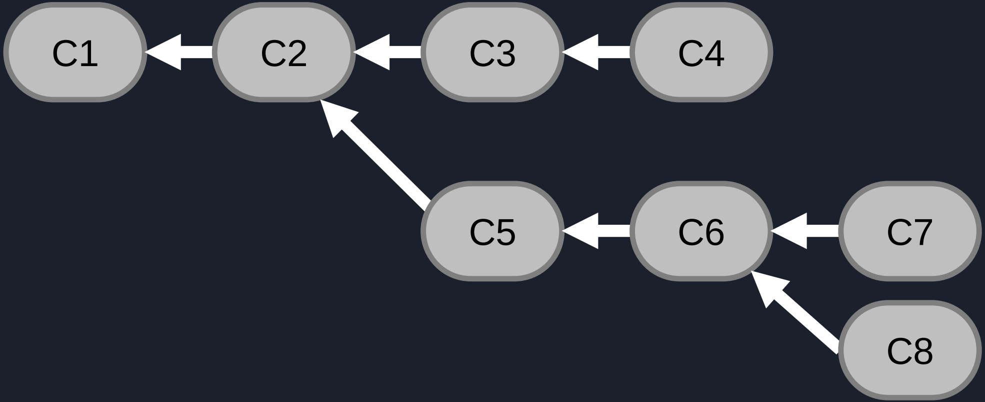
- git log shows a list of actions git has performed
- It's like a timeline for your timeline
- Lists the hashes and descriptions of each commit

git checkout

- `git checkout [commit hash] /path/to/file`
- Is used to copy a file into your working directory
- Will overwrite the existing file
- You can use a prefix of the file hash
- But how do we checkout an entire commit?

Branches

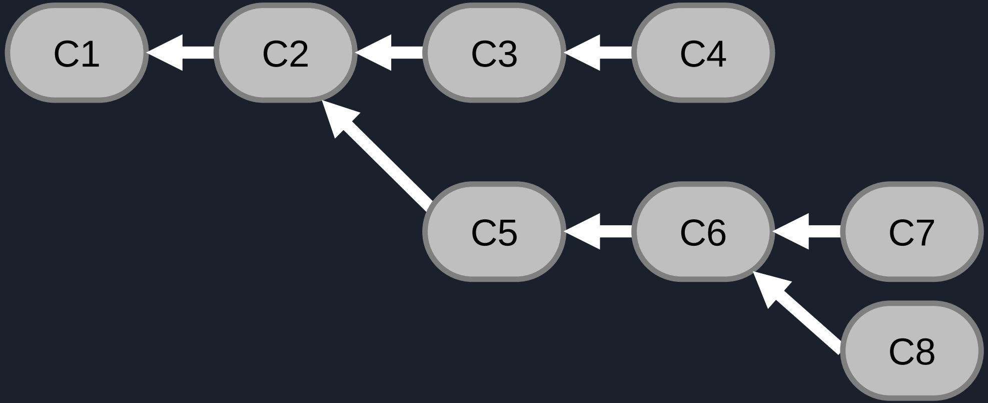
- Like parallel universes



Directed
Acyclic Graph

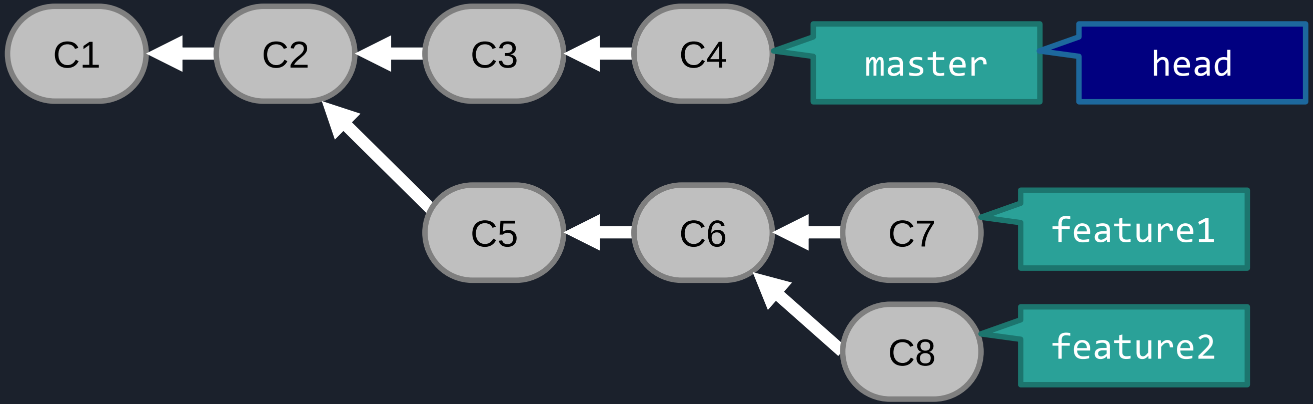
Branches

- How many branches are in this picture?
  - None!
- A branch is a pointer to a commit



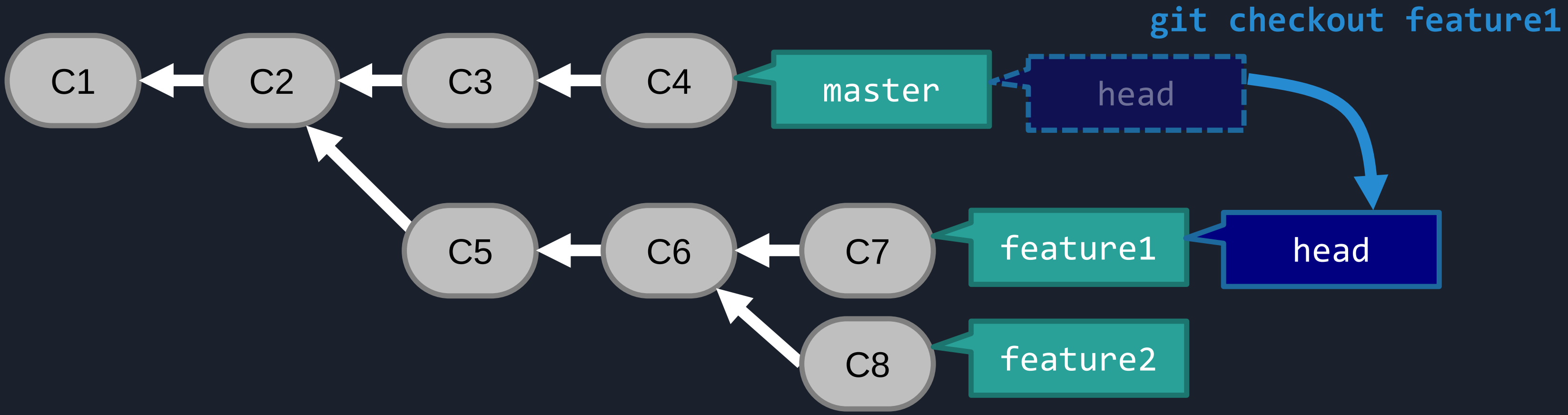
Branches

- How many branches are in this picture?
  - None!
- A branch is a pointer to a commit



Branches

- Use `git branch [branch name]` to make a new branch
- Use `git checkout [branch name]` to switch between branches
- "head" points to your current branch



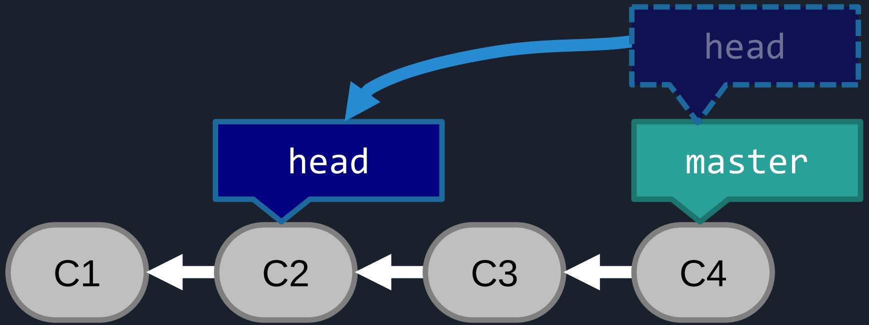
Demo: git branch

Branches

- `git branch [branch name]` (Create a branch)
- `git branch` (List all branches)
- `git branch -d [branch name]` (Delete a branch)
- `git checkout [branch name]` (Switch branches)
- `git checkout -b [branch name]` (Create and switch to a branch)

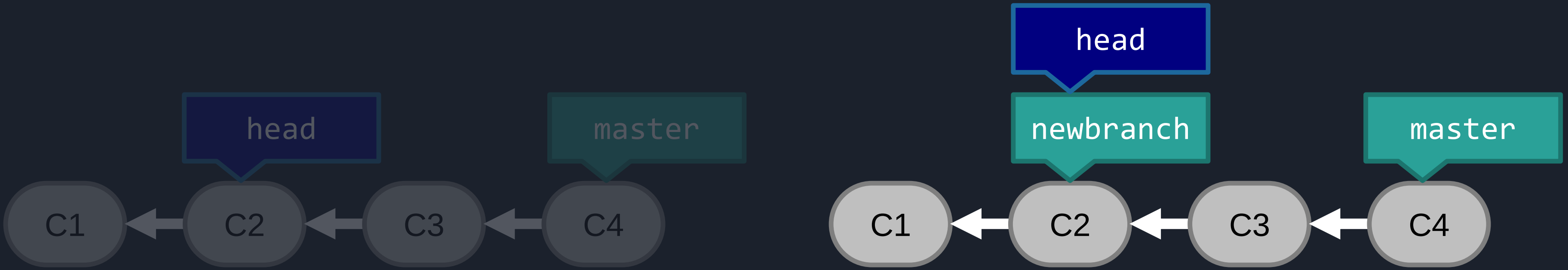
Branching off of a commit

- You can checkout an entire commit
  - `git checkout [commit hash]`
- But there is a problem: you aren't on a branch anymore!



Detached HEAD!

- A detached head error occurs when you are not on any branch (head points to a commit)
- Simple solution: `git checkout -b [new branch name]`
- Now you can make commits to the new branch

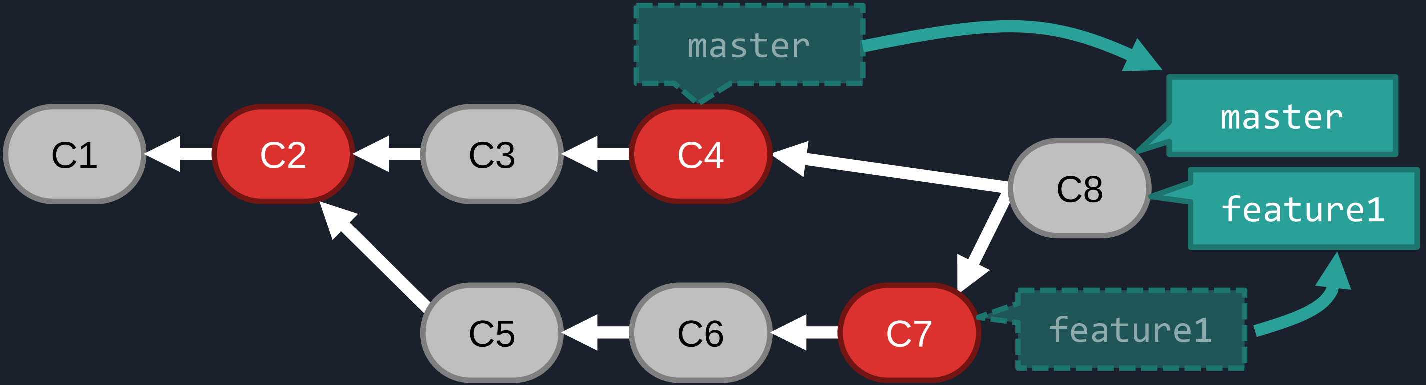


Demo: detached head

Demo: detached head

git merge

- `git merge [branch name]`
  - Merges changes from branch/commit to current branch
- Uses least common ancestor to determine the changes introduced in the commits being merged



Merging steps

1. Checkout the branch you want to merge into
2. Run `git merge [branch name]` for the branch you want to merge from
3. Run `git status` to see conflicts
4. Edit files with conflicts and use `git add` to stage them
5. Run `git commit`

git revert

- `git revert [commit hash]` undoes the effect of a commit

Summary: init, status, diff, log

- `git init` creates an empty git repo in the current directory
- `git status` shows the status of each file
- `git log` shows commits chronologically
- `git diff` shows lines that were changed since last commit

Summary: making commits

- `git commit -a` (Stage all changes and commit)
- `git commit -m "Commit Message"`
- `git commit --amend` (Edit previous commit)
- `git add –A` (Add all files)

Summary: editing process

1. Edit/create some files
2. "Stage" the changes with `git add path/to/file`
3. Commit the changes with `git commit`
4. Enter a commit message and save
5. Repeat

Summary: checkout

- `git checkout [commit hash] path/to/file`
  - Copy file from commit to working directory
- `git checkout path/to/file`
  - Copy file from most recent commit to working directory
- `git checkout [branch name]`
  - Switch to branch
- `git checkout [commit hash]`
  - Copy commit to working directory. Enters detached head state

Summary: branch

- `git branch [branch name]` (Create a branch)
- `git branch` (List all branches)
- `git branch -d [branch name]` (Delete a branch)
- `git checkout [branch name]` (Switch branches)
- `git checkout -b [branch name]` (Create and switch to a branch)

Summary: merge

1. Checkout the branch you want to merge into
2. Run `git merge [branch name]` for the branch you want to merge from
3. Run `git status` to see conflicts
4. Edit files with conflicts and use `git add` to stage them
5. Run `git commit`