Hands-on

# Introduction to Git

Robben E. Migacz

March 7, 2019

Center for
HIGH PERFORMANCE COMPUTING
THE UNIVERSITY OF UTAH

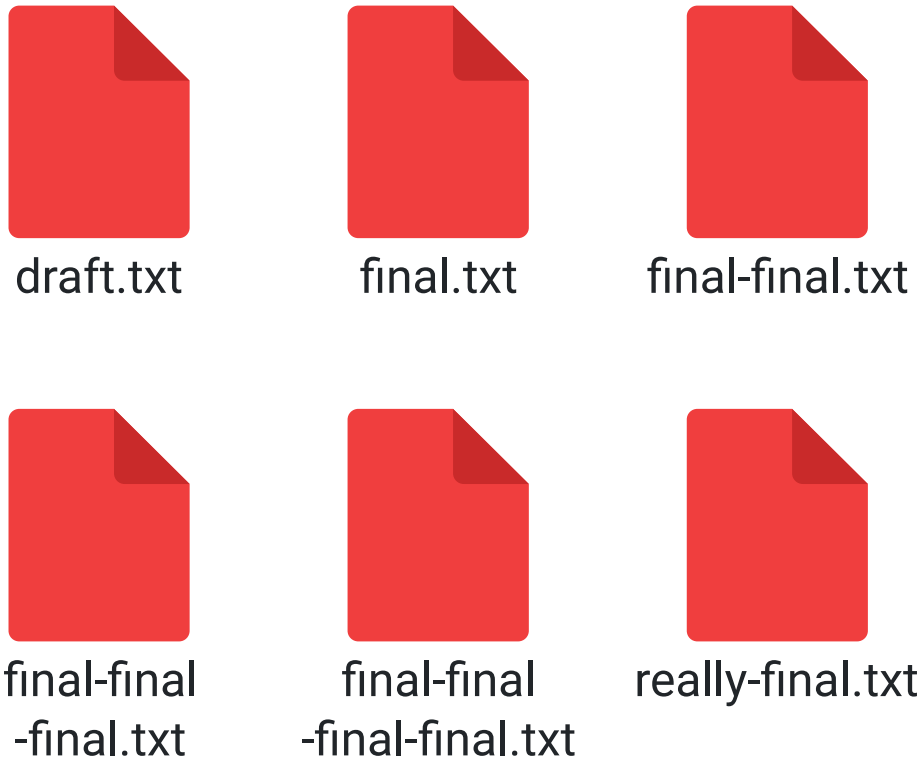
# Agenda

- Introduction to version control
- Overview of concepts and terms
- Tutorial section
- Additional topics
- Time for questions

# Introduction

# Version control

Version control software is used to keep track of **changes** to files over **time**.



# Everyone can use it!

- Writers
- Instructors
- Managers
- Scientists and engineers
  - Digital object identifiers (DOI)

# Why use version control?

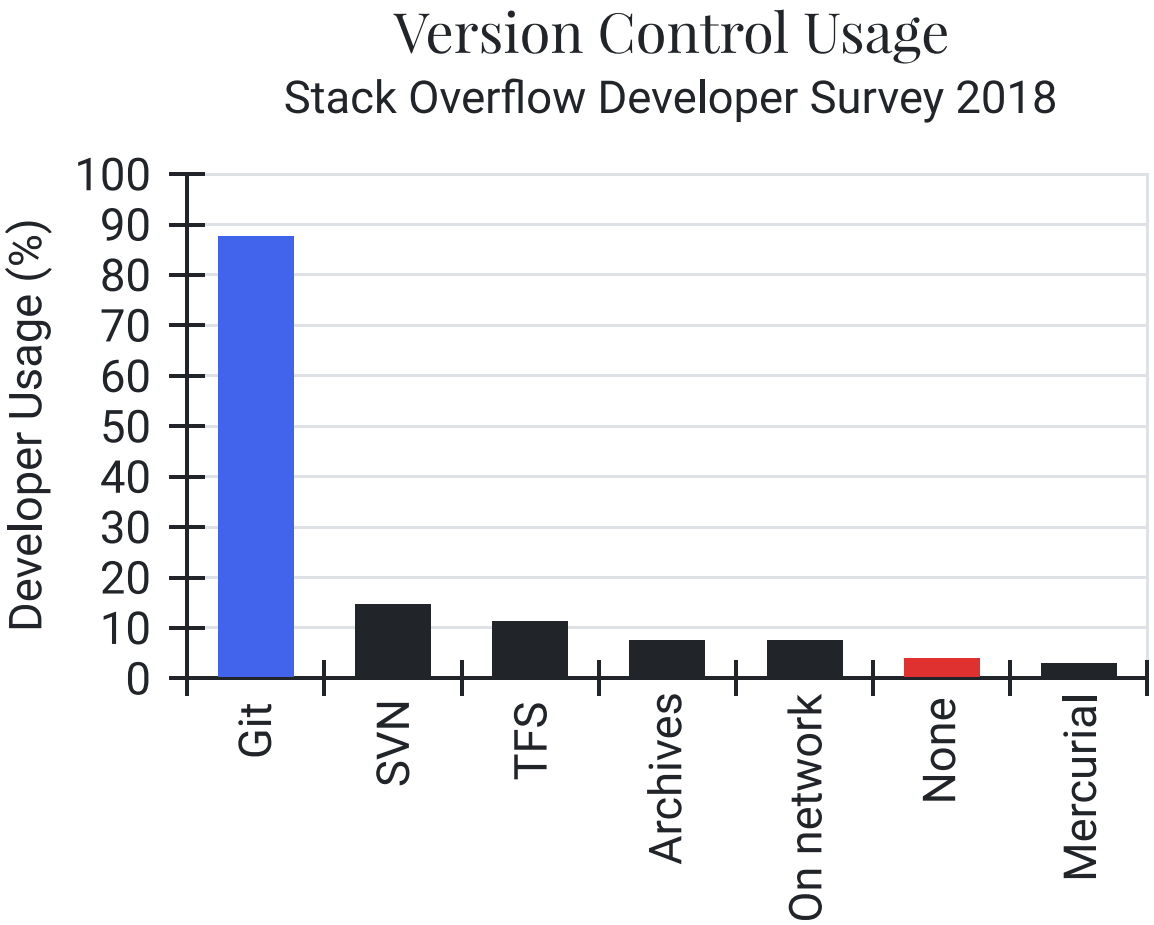
- Collaborate on projects
- Keep historical versions
- Keep copies on remote servers
- Hold editors accountable for changes

Version control software should *not* be used for large backups!

# Available software

- Git
- Subversion
- Mercurial
- GNU RCS
- Commercial offerings

Git is the most common by far.


Version Control Usage
Stack Overflow Developer Survey 2018
Developer Usage (%)
100
90
80
70
60
50
40
30
20
10
0
Git
SVN
TFS
Archives
On network
None
Mercurial

# The Fundamentals

```
git help command
```

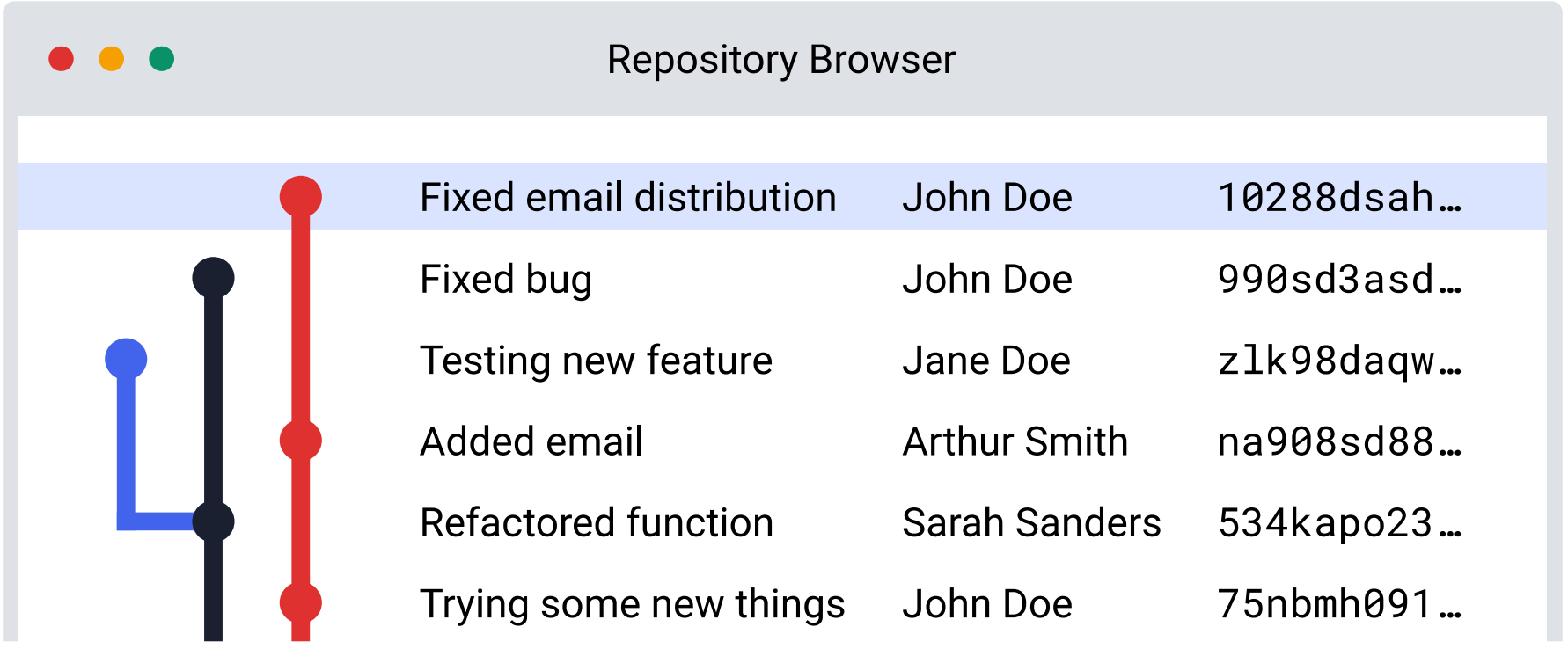
# Git packages

- Linux
  - Install with package managers
- macOS
  - Included with Xcode tools
  - Homebrew
- Windows
  - Git Bash

*Git is not the same thing as GitHub.*

# Graphical tools

Graphical software offers much of Git's functionality without the need to learn commands.



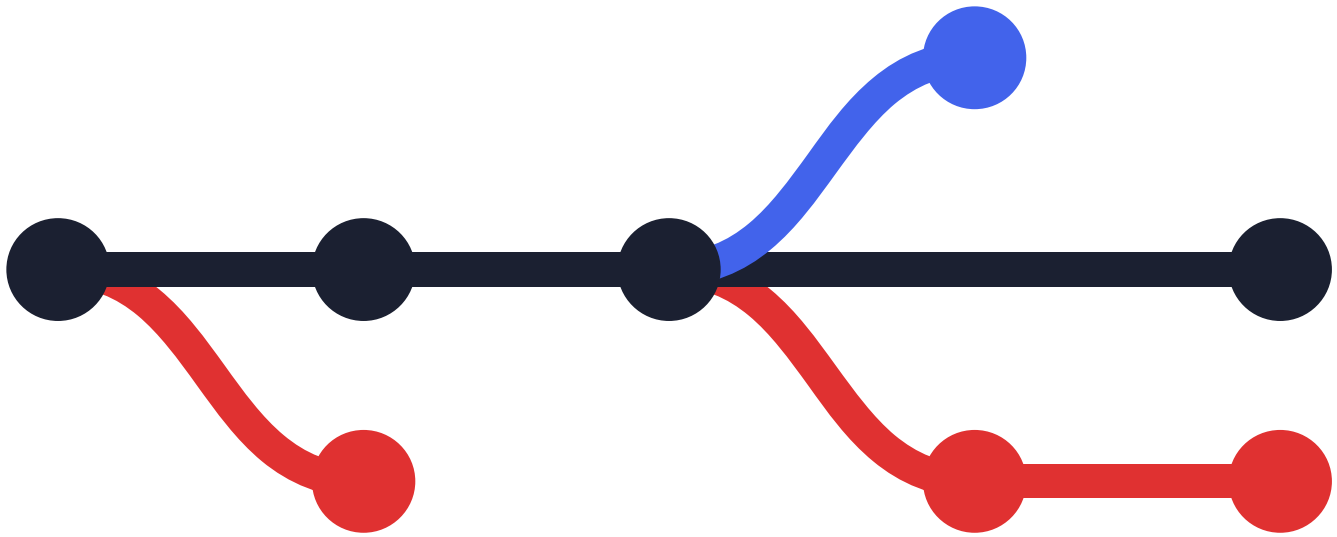
# Concepts and Terms

# The repository

All project files are stored in the repository.

You will need to make or otherwise acquire a repository to work with Git.

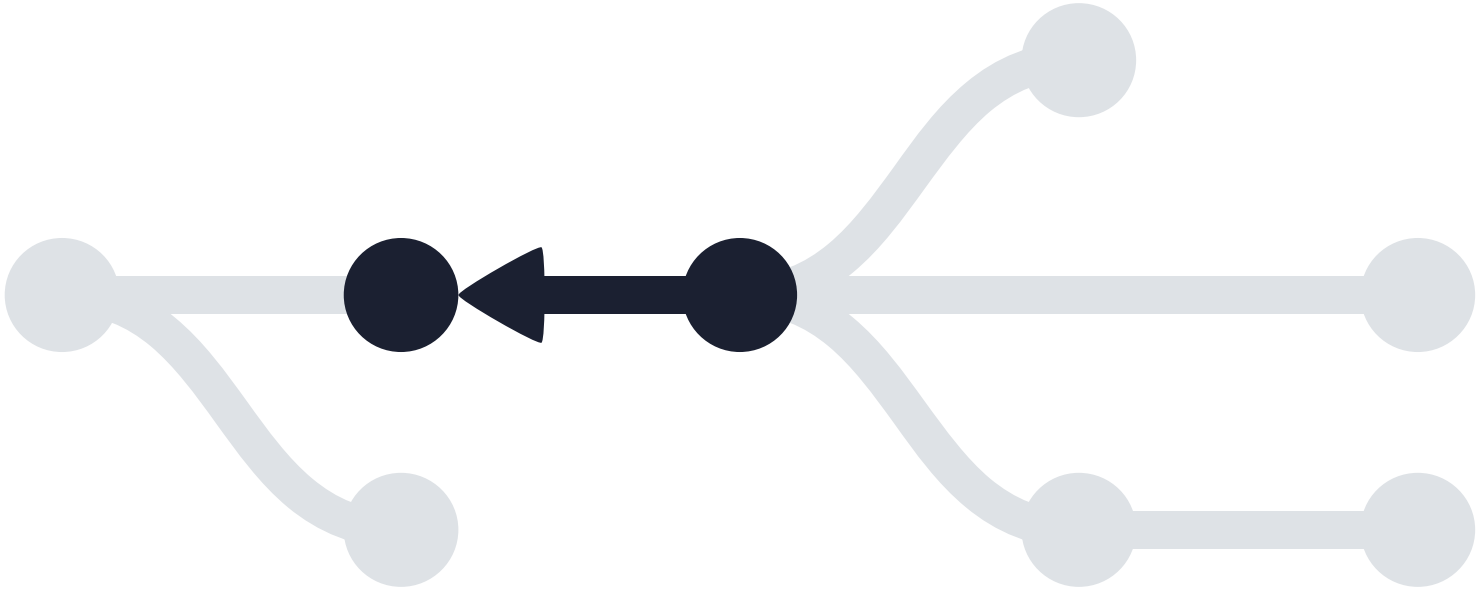
# The graph



# The commit

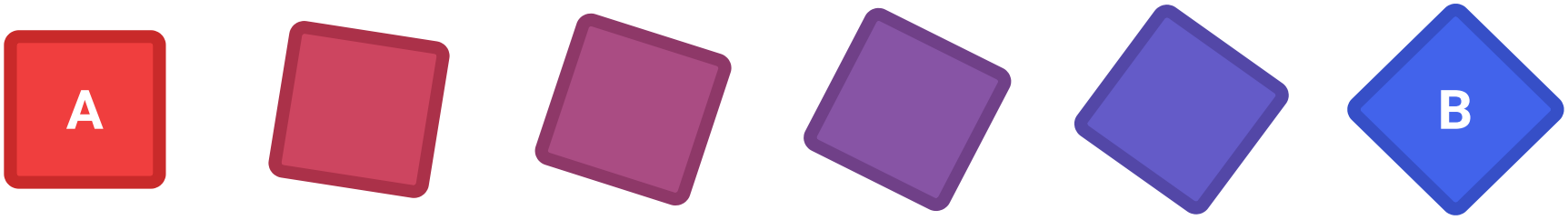

Committer
Commit date
Commit message
Author
Author date
Tree (files)

# Parent commits



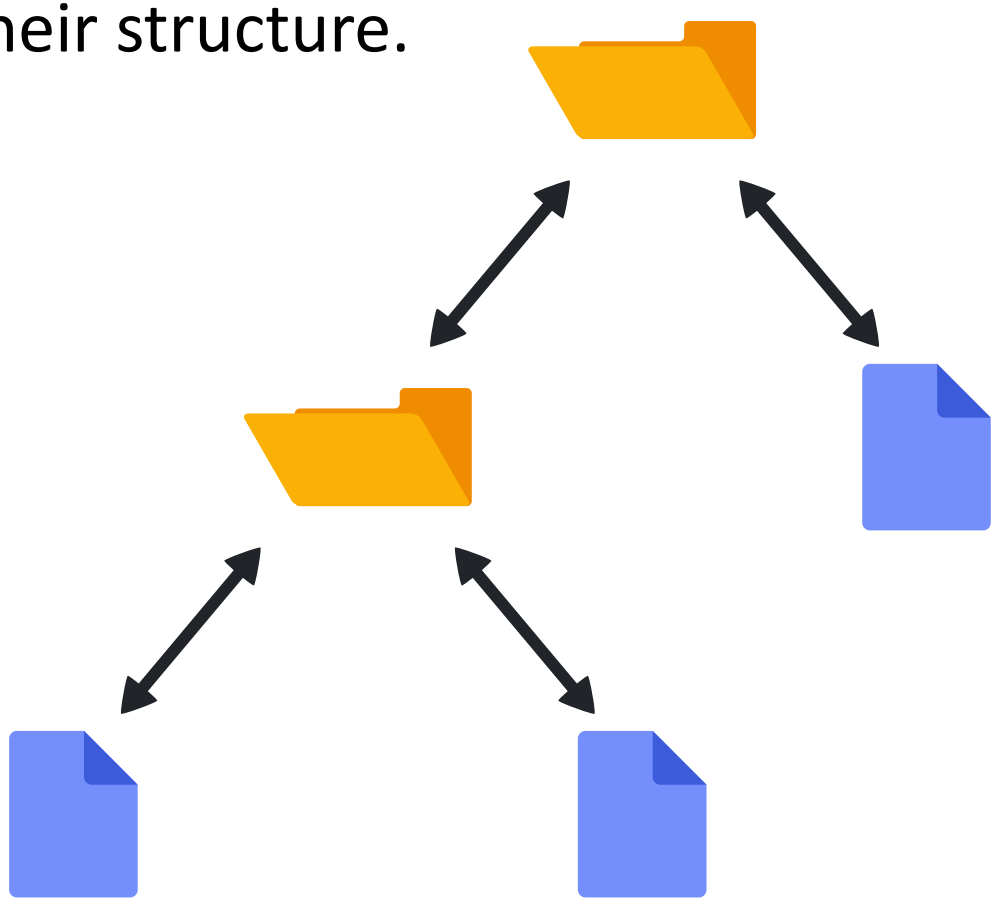
# Snapshot storage

Git stores a snapshot of the whole tree on each commit—not just the changes between commits—to make operations faster.



# The tree

The tree is the hierarchy of files. The term refers to the project files and their structure.

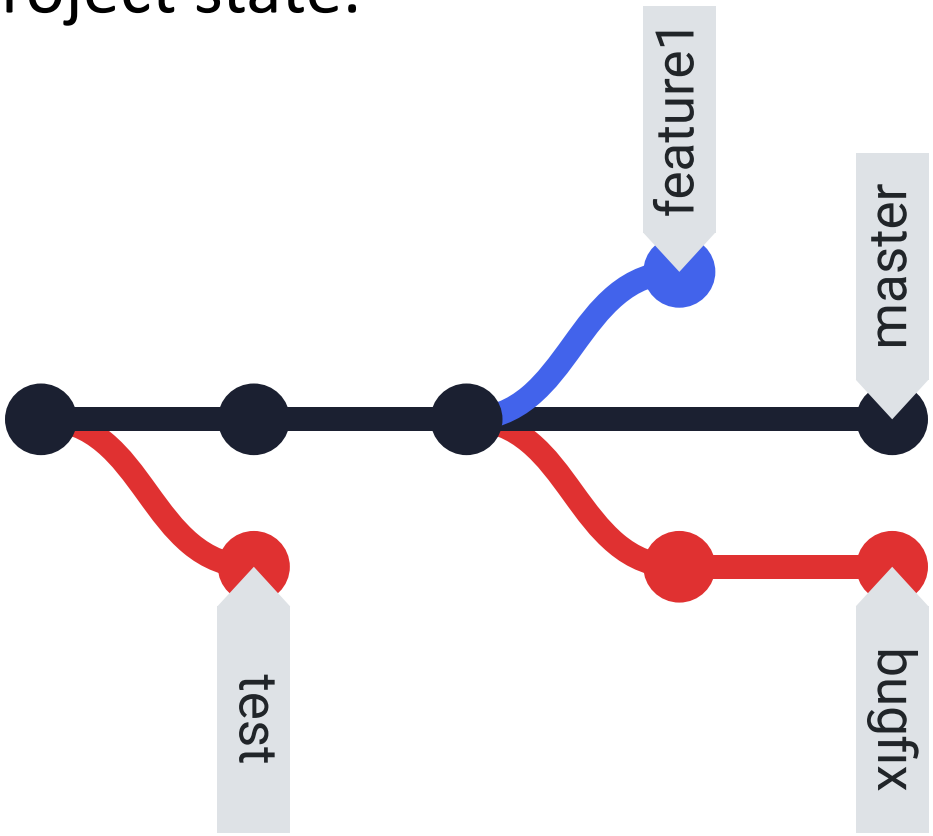


# The blob

Files are stored as “blobs,” or “binary large objects.”

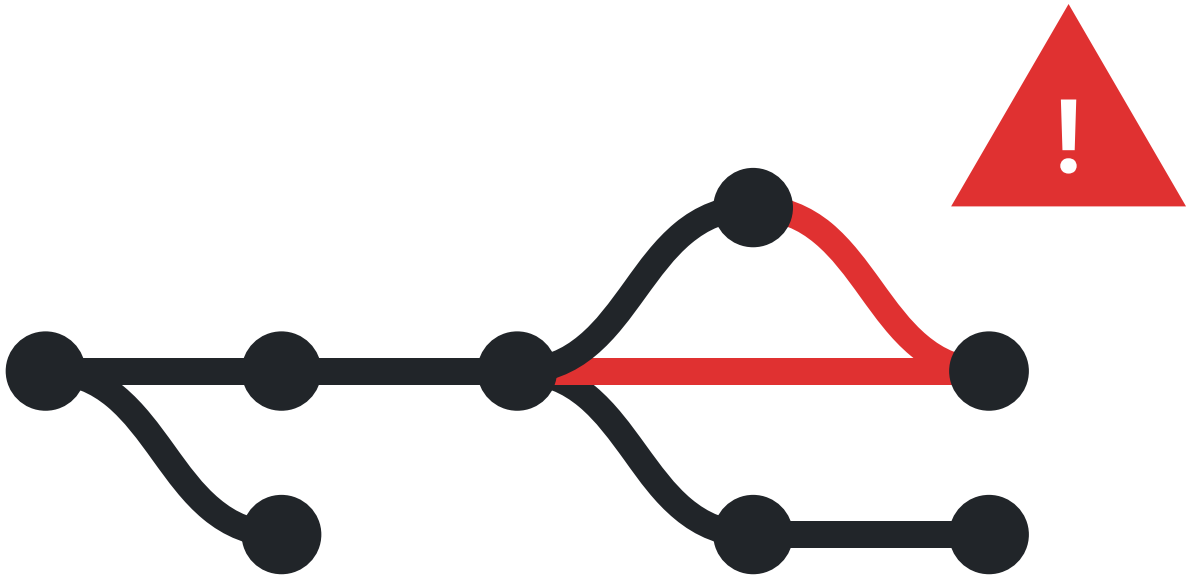
# The branch

A branch is a collection of commits that describe a particular project state.



# Branching and conflicts

What happens if you try to merge (combine) conflicting branches?



# Using Git

# Hands-on: Prepare

Connect to a Linux server.

Get a recent version of Git.

- `module load git`
- `apt install git`
- `yum install git`

Check that it works with `git --version`.

# Repositories

The repository contains the project information.

- Created in a directory with `git init`
- Cloned from an existing source with `git clone source [destination]`

See usage on handout.

# Configuring Git

You should always configure Git in a new repository. Add `--global` to change everywhere.

- `git config user.name "Your Name"`
- `git config user.email "your.name@utah.edu"`
- `git config core.editor editor`
- `git config commit.template path`
- `git config user.signingkey gpg_key`

# Hands-on: Make a repository

Create a new repository or clone one from an existing source.

Configure your name and email address (at a minimum) in the new repository.

# Editing files

You can edit files in any editor.

Text files work best with Git.

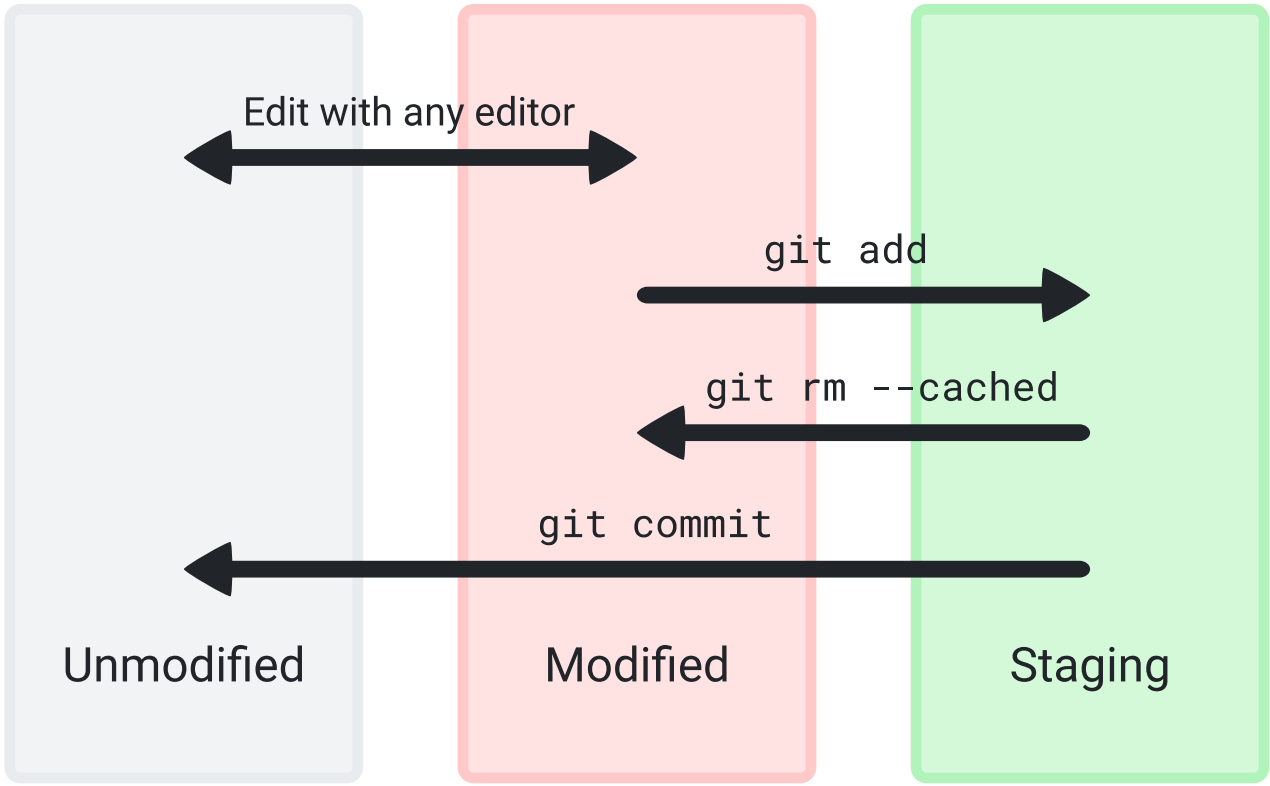
- Consider using text formats for writing
- Binary files like images and word processor documents will not work with comparison tools

# Staging files

Git won't “track” *all* your changes.

- By design
- `git add` files to the “index” (“staging area”) before commits to identify desired modifications

# Effects of Selected Commands



# Commits

Commits only include content from staged files.
(Your files must be in the index.)

- `git commit`
- `git commit -m "Message"`

# Commit identification

Commits are hashed with SHA-1. A long string is used to refer to a particular commit.

The string can be shortened where you need to use it; “5203b1d979f05bcd88c28257950f467e1c2396f9” is (probably) the same as “5203b.”

# Hands-on: Commits

Modify files with any editor.

Add your changes to the index.

Commit changes (be sure to add a message). Make several commits if you have time!

# Logs and differences

View the project history with `git log`.

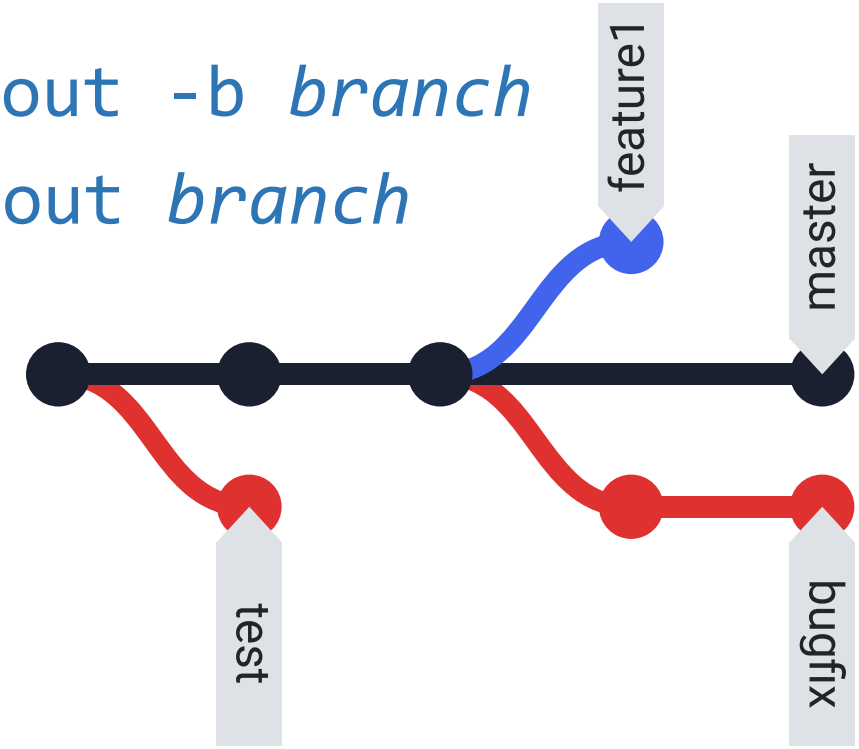
View new changes with `git diff`.

- Add commit identifications to the command
- Without specific commits, this compares the current state to the previous commit

# Branches

Branches allow you to have multiple versions of your project simultaneously.

- List with `git branch`
- Create with `git checkout -b branch`
- Switch with `git checkout branch`



# Hands-on: Branches

Create a new branch.

Modify files on the new branch and make a commit.

# Merging branches

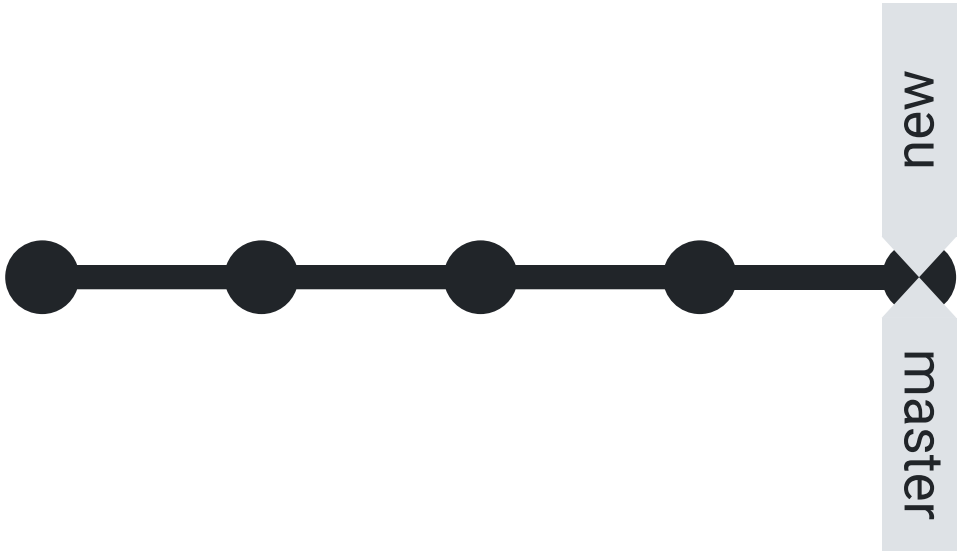
- Switch to the branch you would like to merge *into*
- `git merge source_branch`

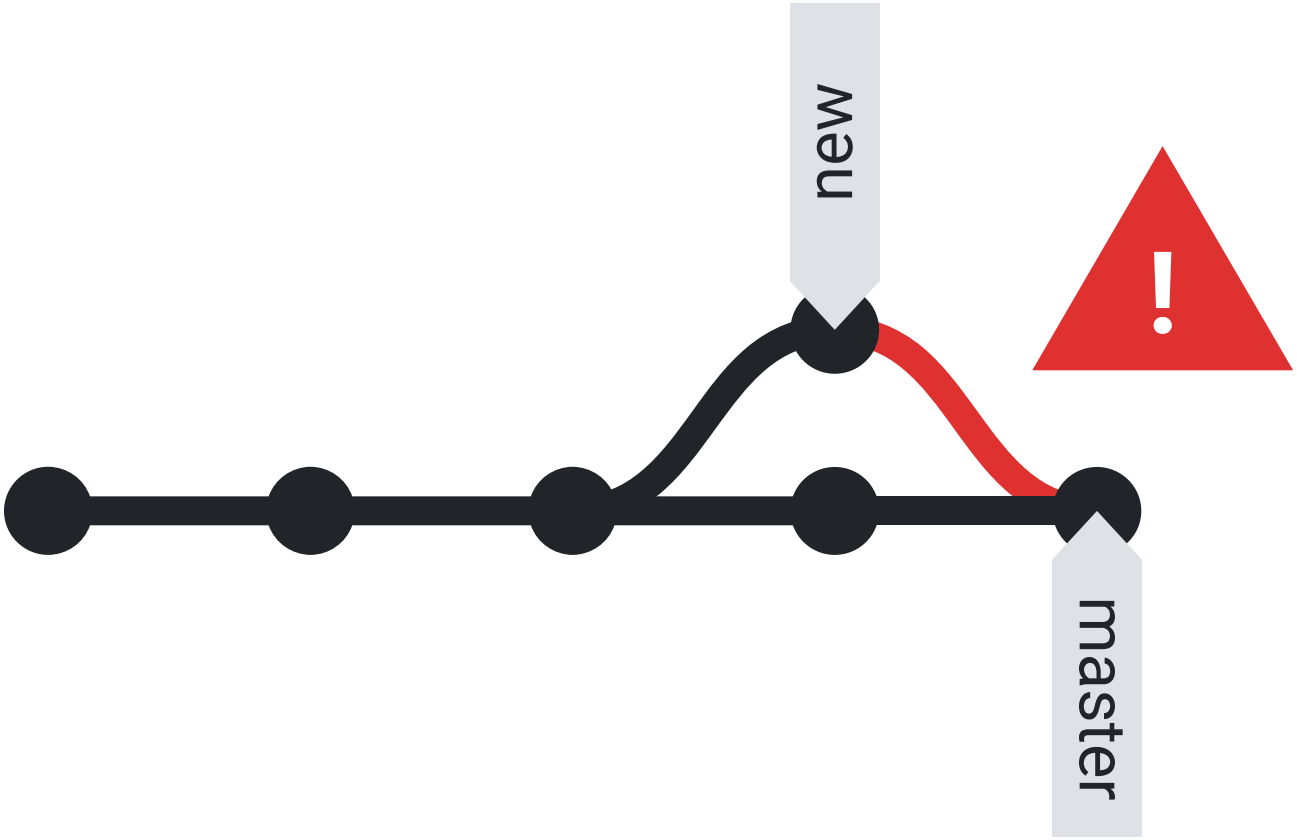
If there are conflicting commits, issues will be identified within files. (More on this later.)

# Hands-on: Merging

Switch to the “master” branch.

Merge the changes from the previous exercise (the new branch).




new
!
master

# Fixing problems

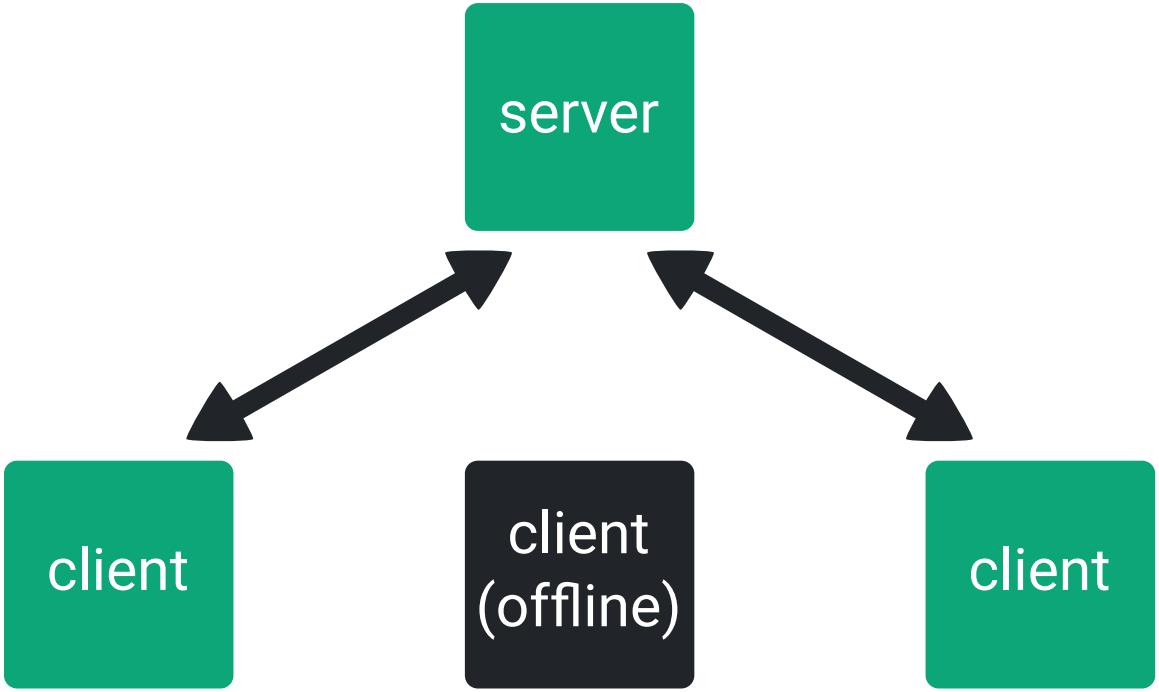
- Fix files with problems
- Create a new commit

```
<<<<<<< HEAD
This is an example of the first version of a file.
=======
This is the second version!
>>>>>>> 57a4c537d0cc429794dfed77d02e5a1bfca9d91b
```

# Remote repositories

- Store projects on highly available resources
- Good option for collaborative projects

- The “origin” remote is configured automatically when using `git clone`
  - The primary remote is typically called “origin”
- `git remote add name url`


server
client
client
(offline)
client

*Never store sensitive information on a remote server unless you are certain it is permissible.*

# Interacting with remotes

Interaction generally consists of uploading and downloading newer versions of the project.

- git push *remote branch*
- git pull *remote branch*

# Working with other projects

- *Forks* are copies of a project owned by another user
  - Helps manage project permissions
  - Protects important content
- *Pull* or *merge requests* are used to ask the original project owner to include your changes
  - Generally done on the remote repository host's website

# Conflicts with remotes

- Similar to merge conflicts
- Generally happen when trying to `git push`

1. Pull the current version with `git pull`
2. Resolve issues in files
3. Create a new commit
4. Try to `git push` again

# GitLab at CHPC

gitlab.chpc.utah.edu

- Accessible with University credentials
- Create projects that cannot be accessed publicly
- *Not* for sensitive information

# Hands-on: Remotes

Create a new project on a remote host (like GitHub or GitLab).

Add the remote to your local repository.

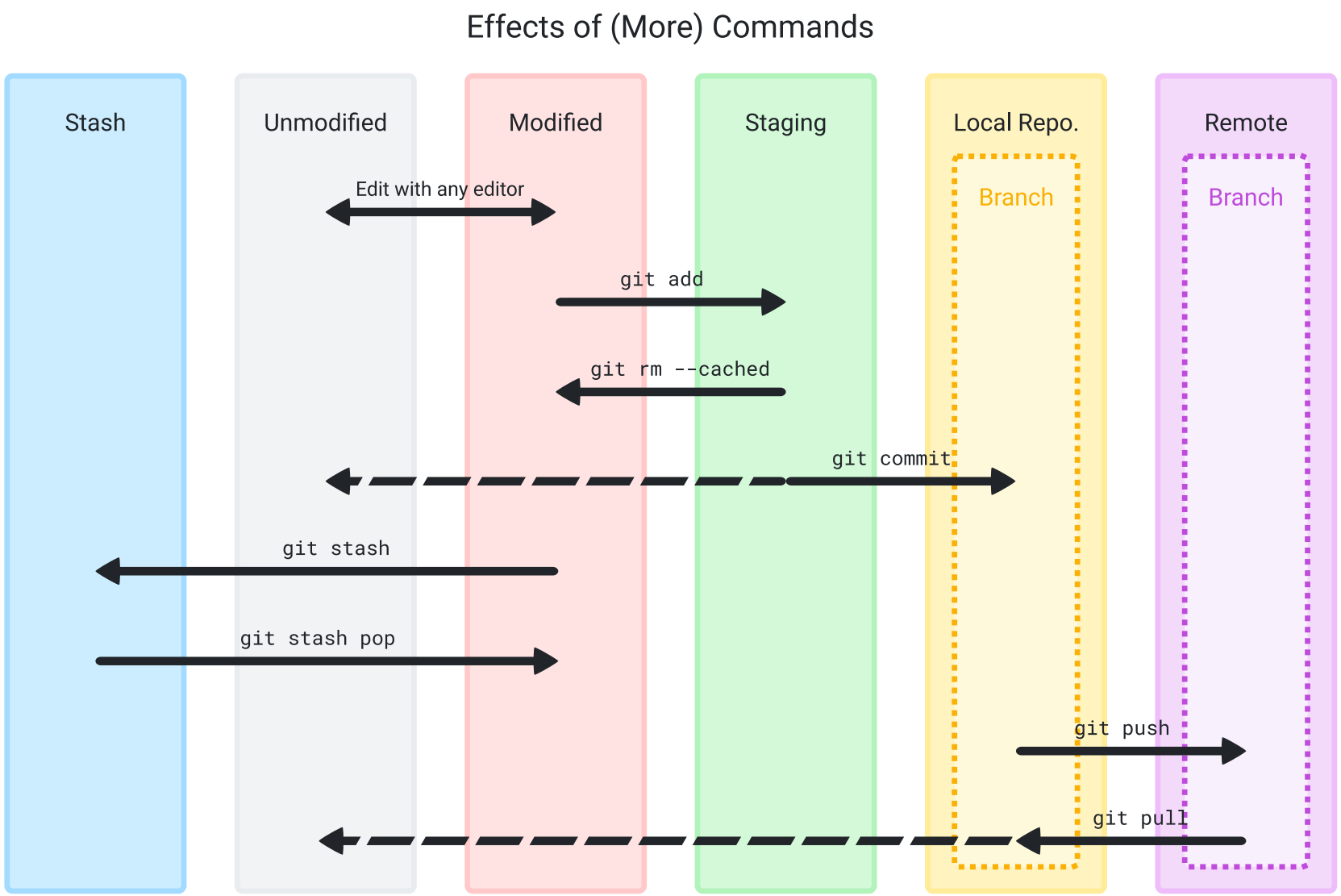
`git push` your project (use `--all` to push tags and all branches).

# The stash

The stash is used to save the project state without creating a commit.

- Helpful if changing state (e.g. testing another user's commits) with unfinished changes
- Returns to a clean working directory

- `git stash push`
- `git stash pop`


Effects of (More) Commands
Stash
Unmodified
Modified
Staging
Local Repo.
Remote
Branch
Branch
Edit with any editor
git add
git rm --cached
git commit
git stash
git stash pop
git push
git pull

# Reverting changes

- `git checkout` a previous commit and create a new branch at that point
  - Works best from an unimportant branch
  - Leaves unwanted commits untouched
- `git revert` to create a new commit that returns the project to a different state
  - Keeps unwanted commits in history
- `git reset` to remove commits entirely
  - *Not a good option for shared repositories*
  - May be acceptable if all changes are local

# Additional Information

# Rewriting history

- Frowned upon; others probably won't like it if you modify anything public
- Can be done with most commands by appending the `-f` flag
- Be *very careful*!

# Continuous integration and delivery

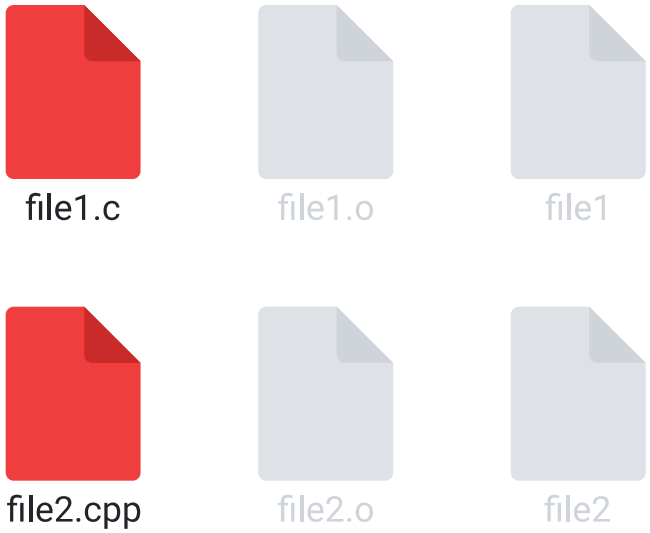
- CI: Merge to primary branch often, complete automated testing
- CD: Similar to CI, but also automates build process

- In theory, release functional software from primary branch at any time
- In the case of “continuous deployment,” successful modifications go directly to end users

# .gitignore

Selectively ignore files (with pattern matching) from most commands.

- Makes operations like `git add *` safer
- Helps avoid clutter from compiled binaries and output files



# .gitattributes

Attributes of files in repository (improves behavior).

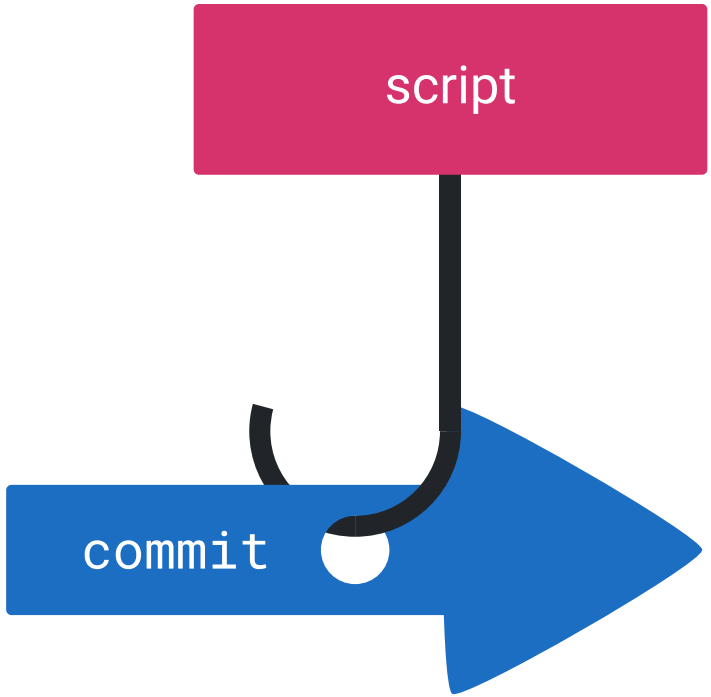
- Identify line endings (Windows, Unix)
- Customize command behavior for certain files
- Mark binary files so they do not appear in `diff` output (most recognized automatically)

# Repository information

- The README file is the source of general information on the project
  - Often Markdown
- The LICENSE file contains the license of repository contents
- The CITATION file provides information on citing the project
  - Most common in academic projects

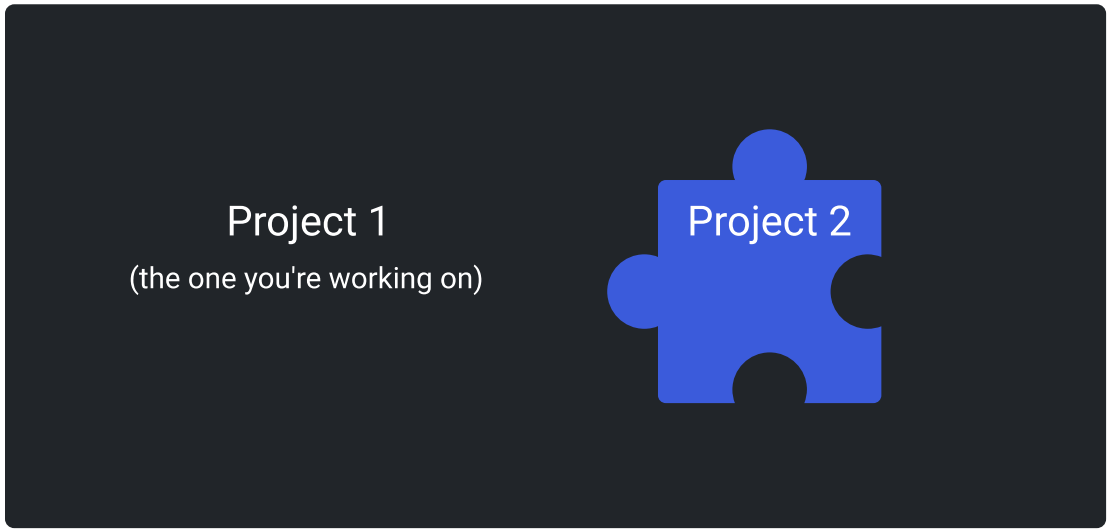
# Hooks

- Found in .git/hooks
  - Examples in *.sample
- Run a script conditionally, such as when you run a command
- Interrupts normal workflow



# Submodules

- Git repositories inside of Git repositories
- Help simplify project structure
- Reduce redundancy and complexity



Questions or comments?

Thank you for your participation!

Robben Migacz
robben.migacz@utah.edu

Center for High Performance Computing
helpdesk@chpc.utah.edu