CS130: Software Engineering

Lecture 1

Overview
Development Environments
Source Control

Checking In

https://forms.gle/bgteLmPNuP6LvHWm6

One Word: Where y'at today?

One Tweet: What excites you about this class?

One Tweet: What is Software Engineering?

We're full!

If you are **not** enrolled:

The class is full. Total enrollment will not increase beyond ~~80~~ ~~100~~ ~~120~~ ~~150~~ 200.

Enrollment in this class was done by the CS department undergraduate counselor to make sure that those students who need it most urgently are enrolled.

If any students drop, the next person on the waitlist automatically gets in. The number is expected to be low, as this is a required class.

Additional enrollments are up to the CS department.

Introduction

Who are we?

Michael Burns

Philo Juang

Alex Monroe

What is software engineering?

Software engineering is not (just) programming

You need to know how to program and use basic data structures.

```
public class LinkedList<T> {
  private T elem;
  private LinkedList<T> next;

  public LinkedList(T el) {
    elem = el;
  }

  public LinkedList<T> add (T el) {
    next = new LinkedList<T>(el);
    return next;
  }

  public LinkedList<T> next() {
    return next;
  }

  public T get() { return elem; }
}
```

Software engineering is not (just) programming

It's programming that's valuable to other people

Software engineering is not (just) programming

It's programming that's valuable to other people

- The person who paid you to write it
- The person who paid you to use it
- Someone you gave it to
- Future-you, who doesn't have to rewrite it.

A small program

- Uses compiler and a few libraries
- Developed alone
- Run a few times
- No obvious bugs
- I know how it works
- Useful to me

A small program

- Uses compiler and a few libraries
- Developed alone
- Run a few times
- No obvious bugs
- I know how it works
- Useful to me

vs. a big program

- Uses large framework or platform
- Developed by a team
- Run many times
- Well-tested
- Documentation
- Useful to many people

Which means:

- Debugging
- Unit, integration, regression tests
- Bug tracking
- Build process
- Release process
- Monitoring
- Performance testing
- Adding new features
- Updating documentation
- Programmer turnover
- Updates to the framework
- ...

BigTable

Prototype

- ~8 months
- <10 people
- Internal users

~15 years later...

- Still going strong
- More people (team varies in size)
- Commercialized as Cloud Bigtable

- Same basic features
- Almost everything else different

Valuable software projects have risk

Valuable software projects have risk

Software engineering is about managing risk

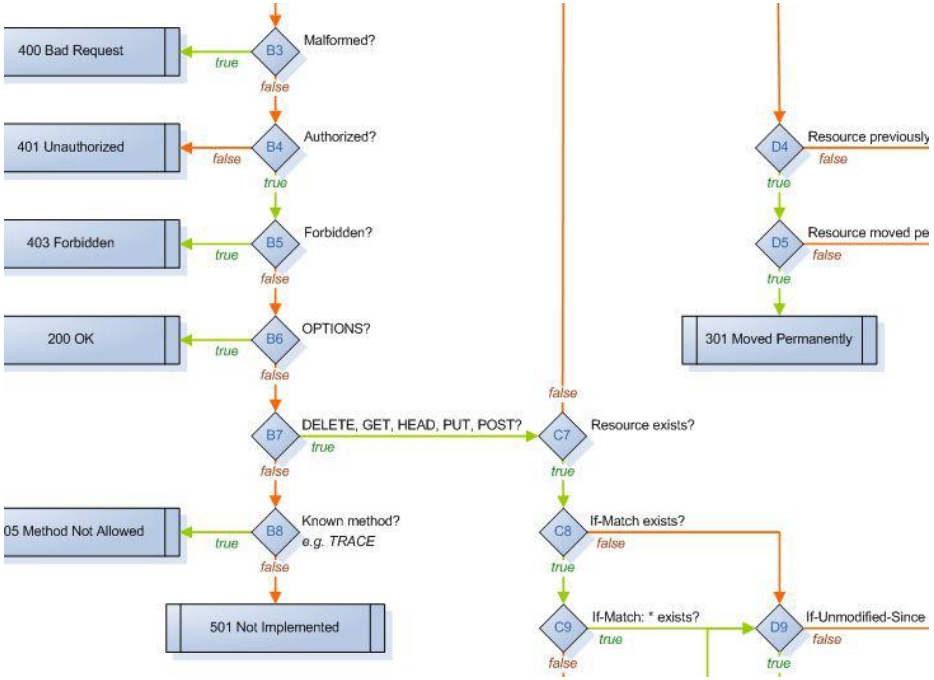
- Revision control
- Code reviews
- Coding style
- Refactoring
- Testable code
- Unit testing
- API design
- Design reviews
- Integration testing
- Build automation

Learning Software Engineering

- Hearing about tools, procedures, and techniques for writing valuable code.
- Putting it into practice
- Watching each other practice
- Hearing from people who have practiced
- Building something valuable!

Project / Assignments

Project: Web server



(From a flowchart by Alan Dean, CC BY license.)

What are the goals?

- Learn how to develop scalable, maintainable, professional-quality software
- Learn how to design robust APIs
- Learn how to test, deploy, and monitor software
- Learn modern technologies
- Learn how to work with other software engineers on a team

What are NOT the goals?

- Learn how to build a web server
- Build something incredibly useful
- Make a project for your resume

Assignments

- Assigned on Tuesday by noon
- Due following Tuesday by noon
- Submit by online form

Late Policy

- Each person starts with 25 hours of late time budget for the quarter
- After exhausting, the budget an assignment receives a 1% penalty per hour
- Each week, the team's tardiness depends on when their TL submits the team form, by default deducting from their TLs personal late budget.
- You may donate hours from your budget to help the team.

Late Policy (Exception)

- Partway through the course, you will be asked to work in another team's codebase
- For this assignment, we will cap the number of late hours you will be able to use so that other teams

Assignment logistics

Each week you should:

- Read the assignment
- Meet as a team, where you:
  - Discuss a plan of action
  - Split the assignment into tasks
  - Assign tasks to individuals
  - Keep notes in a doc
- Write code for your assigned task(s)
- Send code out for review
- Reply to review comments
- Submit code

This is a team endeavor!

- Teams of 4
- Same discussion section
- Don't flake!
- Each week 1 person is the Tech Lead (TL), on a rotating basis

TLs must:

- Lead the weekly meeting
- Keep notes of the weekly meeting
- Review ALL code
- Ensure the quality of the code and health of the project
- NOT write any code (!)

Software Engineering is a team endeavor

- Using and learning from publically available code is encouraged in this class, and will be useful in your career.
  - Always follow license restrictions.
  - In this class, you must cite your source in a comment in the form of a URL.
- Make an honest effort to figure things out, but don't spend hours on some tough problem—look at what someone else did. Most problems have already been solved.
- Goes both ways: share what you've learned with teammates.

Suggestions...

- Learn from each other.
- Start early, get help where needed! Make use of the discussion section and office hours.

Suggestions...

- Learn from each other.
- Start early, get help where needed! Make use of the discussion section and office hours.
- Assignments build on each other!!!

Suggestions...

- Learn from each other.
- Start early, get help where needed! Make use of the discussion section and office hours.
- **Assignments build on each other!!!**

Suggestions...

- Learn from each other.
- Start early, get help where needed! Make use of the discussion section and office hours.
- Assignments build on each other!!!
- Assignment 7 is a doozy – make sure your code is in good shape before then!

Suggestions...

- Learn from each other.
- Start early, get help where needed! Make use of the discussion section and office hours.
- **Assignments build on each other!!!**
- Assignment 7 is a doozy – make sure your code is in good shape before then!
- Pay attention to participation requirements

Don't let this be you!

Technology we assume you know

- C++
- Basic data structures and algorithms
- Linux
- Basic Git, or walk yourself through learngitbranching.js.org
- Basic networking and concurrency

(Or a healthy relationship with stackoverflow.com)

Technology we will help you learn along the way

- Shared code repositories (more advanced Git)
- Containerization (Docker)
- Make / build systems (CMake)
- Remote deployment (Google Cloud Platform)

Piazza – For students

- We have a Piazza class for us to use
- Should be mostly used for knowledge sharing among teams
  - Instructors will not be crawling through Piazza helping to debug your code
- How to ask a good question:
  - Specifications in this course may be left intentionally ambiguous
  - Do not ask what you should do; lay out your options and explain which option you think is best and why

Piazza – For students

- Bad questions lead to bad answers

Student: How should we handle double slashes in URL paths?

Us: How do you think we should handle double slashes?

Piazza – For students

- Bad questions lead to bad answers

Student: How should we handle double slashes in URL paths? Should we fail or should we accept them in a path?

Us: Which do you think makes more sense?

Timeline

~2 weeks

- Assembling teams
- Development environment
- Deployment

~3-4 weeks

- Initial implementation of the project
- API selection
- Midterm

~3 weeks

- Refactoring
- Documentation
- Working with someone else's code

~1-2 weeks

- Feature of your choice
- Demos!

Evaluation (Grading)

Assignments - 70%

- Implementation, tests, code reviews, documentation
- Presentations, written assignments

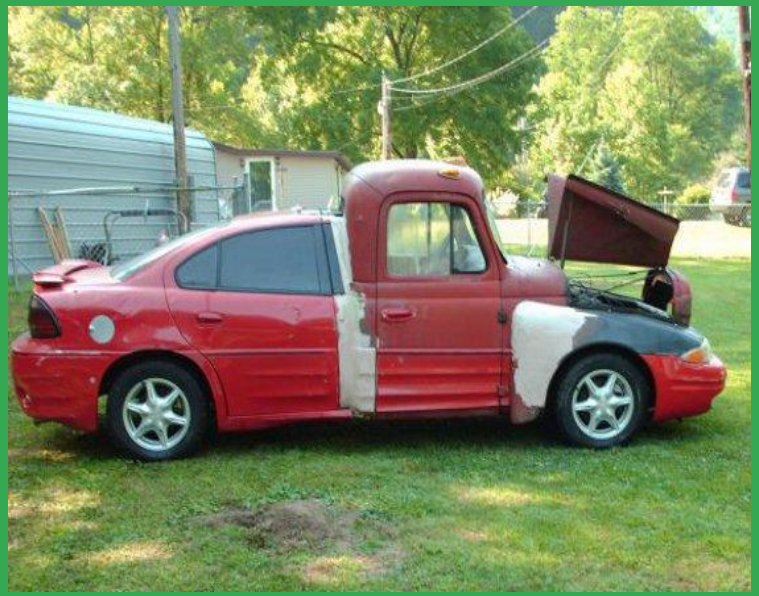
Midterm - 15%

Final - 15%

Standardization: Development Environment

Student Development Environments

- Identity
  - Who knows?
- Platform
  - Linux/MacOS/Windows?
- IDE
  - Visual Studio/XCode/CLion/VS Code?
- Compiler
  - Clang/GCC/ICC/Visual C++?
- File / directory structure
  - Monolithic or separated?
- Code style
  - Opening brace? Indentation?
- Source control system
  - CVS/SVN/Perforce/Git/None?



Corporate Development Environments

Standardized:

- Identity
- Platform
- IDE
- Compiler
- File / directory structure
- Code style
- Source control system

CS 130 Development Environment

- Identity
  - @g.ucla.edu
- Platform
  - Linux
- IDE
  - Recommending VS Code
- Compiler
  - GCC
- File / directory structure
  - Not monolithic
- Code style
  - Make some decisions
- Source control system
  - Git

Identity:

Account is **joebruin@g.ucla.edu**

User is **joebruin**

Demo

- Host is MacBook
  - `clang` compiler
  - missing `gcloud`
  - has `docker`
- Start devel env
  - SSH key
  - `gcc` compiler
  - has `gcloud`
  - has `docker`
  - `git` using cs130.org user

```
cs130 — my_user@my_user_devel_env: /usr/src/projects — docker • bash tools...
Using source directory /Users/mbx/cs130, mapping to /usr/src/projects
Enabling docker in docker
05b13d1fb3954b113043b49dd6e59eaeaf0200859d617aaebd37de29a56ca93b
Starting devel environment
Setting up docker socket...
Setting up git...
Checking SSH key...
Generating SSH key...
Copy the yellow text below to a new SSH key at https://code.cs130.org/settings/#
SSHKeys

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDYvhF7vlG4DsL1sZkhQc8COSMCn97T92+yF8y+Rmc7
FQHnEGDH50e8UYa+kn+k5s2QfgUASPF5S9XbkESzgWGywEtHvWaMH1SwGa2jbxyH2M3OG74yowgijwmL
L1PLqFE8tJXHLfKGqXv4faVuqeh6DzmaMpFlqpm6M4r7biKq8vvu98qM9xT/OtLzHLfylqygVivP+qg8
07mMT8iaQZckJ1MDfO13OD5Sbbd4dwaT5qG68dSEutmoUhQjQgp2C0SgX4lfL4qQ6D10M6DB6XFLxtEJ
PVlc6zQ77aqgI1WmMkn0sg6F5J6BRafvZuOtR0Du7HFcl8vVRyitY1BAyCKuqPP4Kts8KfZaP3oZxzYJ
g3IO6wi/7WKQ6wgZ50djdsmUkXs5wH9RF+DbABBFgfzEqc54uEc8dKtdzV7SxdjrdUkAfyDpEif/qyFd
eUTb77xAWDpJ0a1KGtV3gPqe5jWrAdWXawAM4EQ+pKGW9AW9KUg4DW7F052IuQpbSqaNEl3LdOjbqOKI
6Hz7E1UvFoRP/xwgN32uEkMvUOy+RZRGExgNr4lUxk7wjyLG5jo/8NjmlpbJ2R47oawxSdh+mn8bK/Qy
ARLfH+h2zgg1mfPDcnRTMa6uByt578vnQyv0zDoJHgNyulmFxQ+6Cm8QAsmhTnMpSHSSoqq5Qoh/x+hf
8Q== my_user@my_user_devel_env

Finished initializing development environment
my_user@my_user_devel_env:/usr/src/projects$ ssh key up there
```

Docker what?

- Container: lightweight VM-like environment
- Defined by a series of commands in Dockerfile
- Runs on all major host platforms
- Useful for standardization
- Useful for deployment

Code file/directory structure

Guidelines:

- Use `.cc` file extension
- One file per C++ class implementation
- Separate `_main.cc` file for `main()`
- Separate directories for headers and tests

```
my_project/
- build/
  - bin/
  - [more build files]
- include/
  - server.h
  - session.h
- src/
  - server.cc
  - server_main.cc
  - session.cc
- tests/
  - server_test.cc
  - session_test.cc
```

Source Control

One rule

All valuable source code resides in source control!

Why?

- Reproducible state
- Backing up work
- Documenting progress
- Post-mortems
- Make collaboration possible
- Legal pedigree

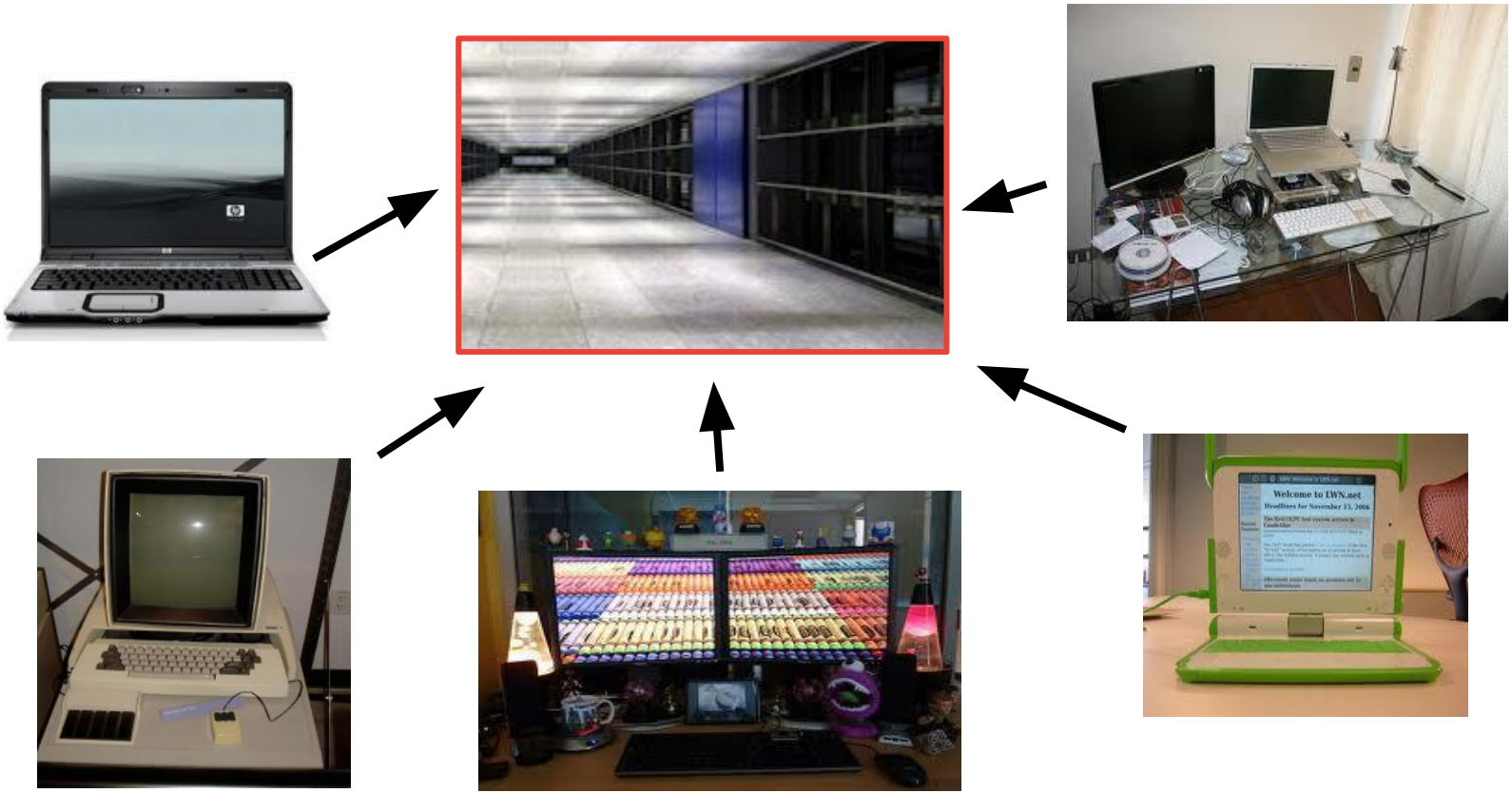
Source control tradeoffs

- Adds complexity
  - can't just edit files willy-nilly
  - have to check them in, check them out
  - have to figure out how to size changelists
  - more thinking
- ... but reduces risk
  - of losing progress
  - of not being able to solve user problems
  - of legal liability
- ... and is more scalable
  - more people can contribute sensibly
  - allows specialization

Source control tools

- Git
- Subversion
- Mercurial
- CVS
- Perforce
- SourceSafe
- BitKeeper

Repository diagram



Changelists and filesets

- Files in source control organized by the logic of the project and build system.
- Different source control systems treat files differently
  - Git treats the entire repository at once, and references the whole fileset by a hash
- Changelists are determined by development logic
  - Chosen to advance the goals of the project
  - Use changelists to make sense internally as a project state transition.
  - Ideally they are:
    - self-contained
    - small
    - single-function
  - Can be part of a larger group of changelists
  - Larger sequences can be arranged as branches

Git source control: Basic idea

What we want:

v1	v2	v3	v4	v5
README.txt — Today is Monday —	README.txt — Today is Tuesday —	README.txt — Today is Wednesday —	README.txt — Today is Thursday —	README.txt — Today is Friday —

Git source control: Basic idea

What we want:

v1	v2	v3	v4	v5
README.txt — Today is Monday —	README.txt — Today is Tuesday —	README.txt — Today is Wednesday —	README.txt — Today is Thursday —	README.txt — Today is Friday —

Anybody see any problems with this?

Git source control: Basic idea

What we want:

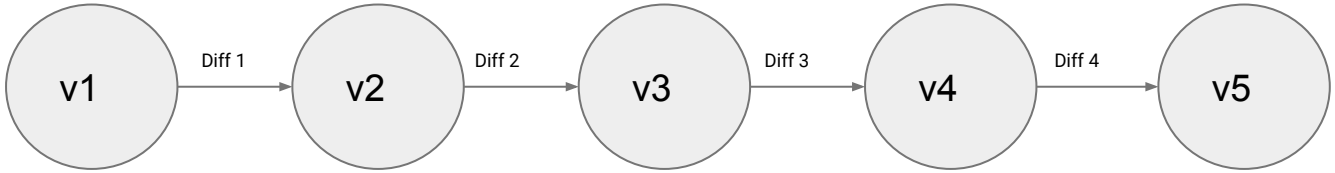
v1	v2	v3	v4	v5
README.txt — Today is Monday —	README.txt — Today is Tuesday —	README.txt — Today is Wednesday —	README.txt — Today is Thursday —	README.txt — Today is Friday —

With this paradigm, need an **entire copy of the whole repository** for every version

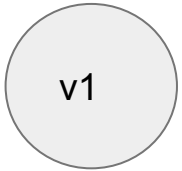
What if your repository contains 1GB? What if you have 100 developers submitting code so there are 100 new versions a day?

Git source control: Diffs

Solution: Instead of storing a copy of the repo at each version, store only a “diff” that tells you how to modify each version to get to the next incremental version



What does git store?



+ Diff 1, Diff 2, Diff 3, Diff 4

Git source control: Diffs

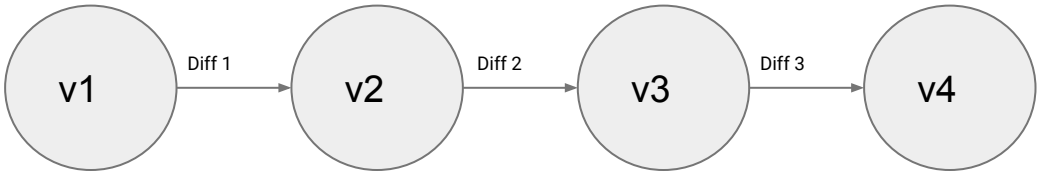
Example:

README.txt v1	Diff 1	README.txt v2	Diff 2	README.txt v3	Diff 3	README.txt v4	Diff 4	README.txt v5
1: Today is 2: Monday	2: Mon -> Tues	1: Today is 2: Tuesday	2: Tues-> Wednes	1: Today is 2: Wednesday	2: Wednes -> Thurs	1: Today is 2: Thursday	2: Thurs-> Friday 3: +TGIF	1: Today is 2: Friday 3: TGIF!

(Git stores the green boxes only)

Git source control: Glossary

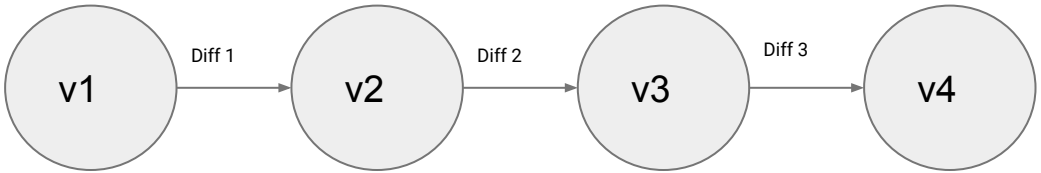
- Commit: A snapshot of the repository at a given point in time (think “node)
    - e.g. v3
- Changelist: A diff that represents how the repository is changed between commits (think “edge”)
    - e.g. Diff 2



Git source control: Making new commits

- In order to commit, edit your files locally, add them to potential commit, and then call `git commit`.
- Git will automatically collapse your changes into a diff and store those changes
- Git will also give your commit a name (hash of fileset)

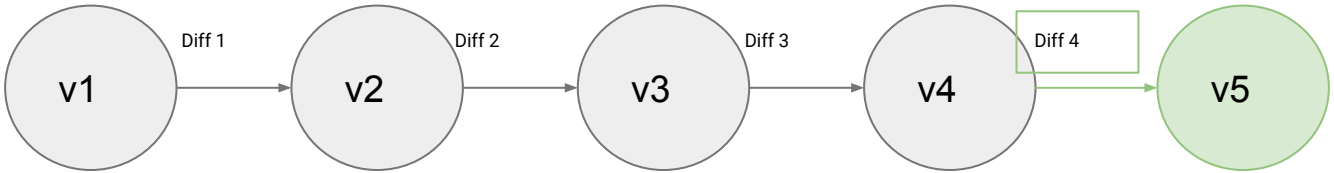
```
$ git add .

$ git commit -m “Adding more docs”
```



Git source control: Glossary

- In order to commit, edit your files locally, add them to potential commit, and then call `git commit`.
- Git will automatically collapse your changes into a diff and store those changes
- Git will also give your commit a name (hash of fileset)

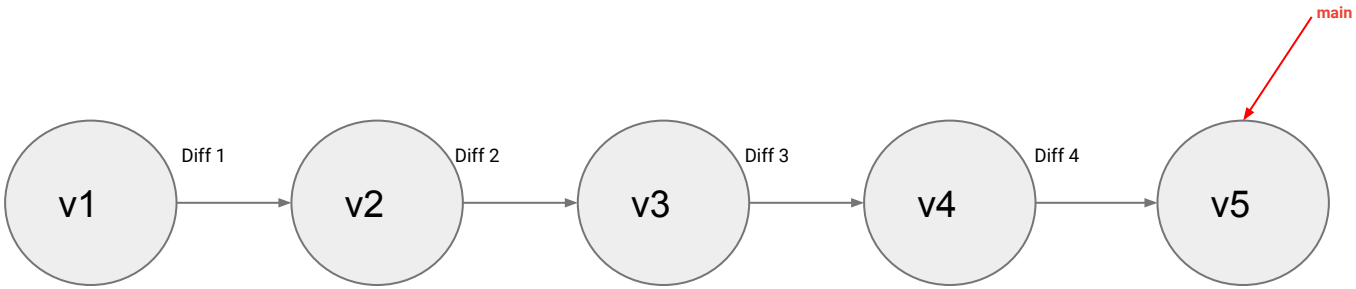
```
$ git add .

$ git commit -m "Adding more docs"
```



Git source control: Glossary

How do I look at versions of the repository?

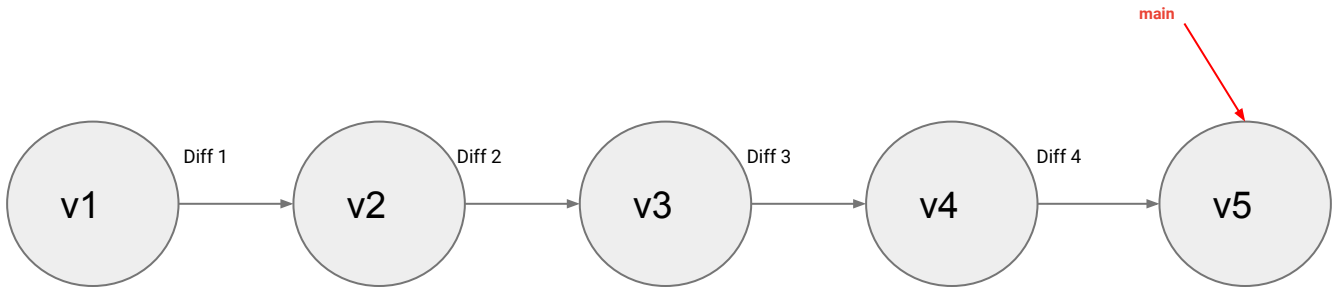
- Branch: A pointer to a specific commit



Git source control: Making a new branch

- “Making a new branch” means “create a new pointer that points at the same place I’m currently pointed at”

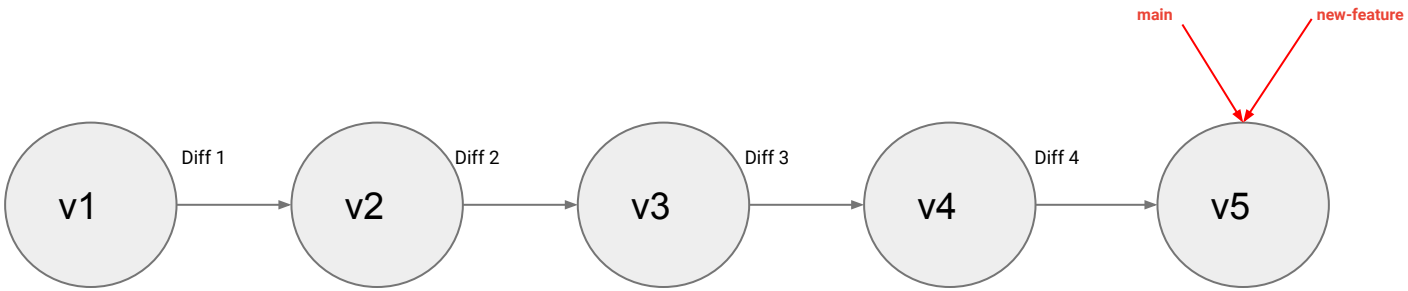
```
$ git checkout -b new-feature
```



Git source control: Making a new branch

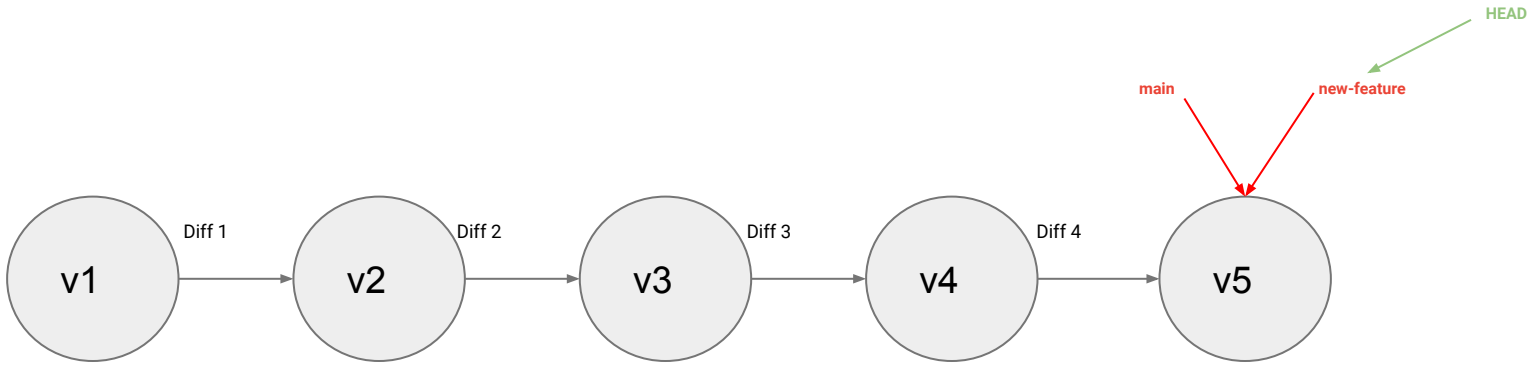
- “Making a new branch” means “create a new pointer that points at the same place I’m currently pointed at”

```
$ git checkout -b new-feature
```



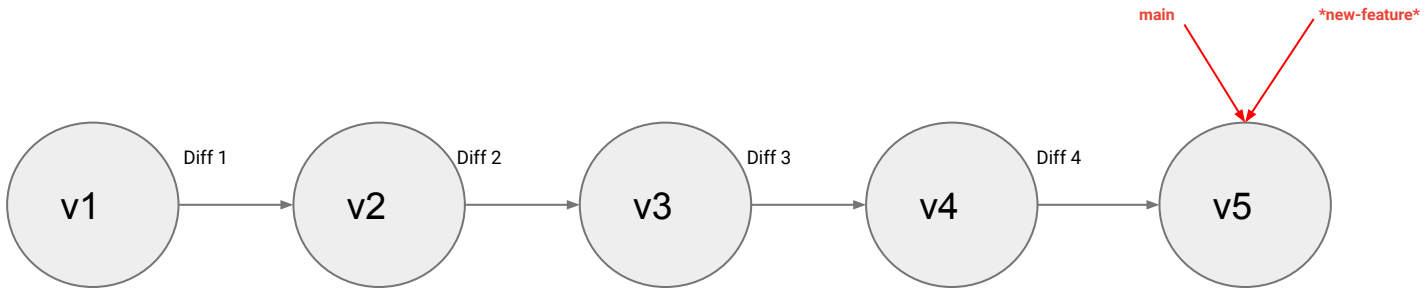
Git source control: Making a new branch

- More terminology: “HEAD” – the commit where **I** am currently working



Git source control: Making a new branch

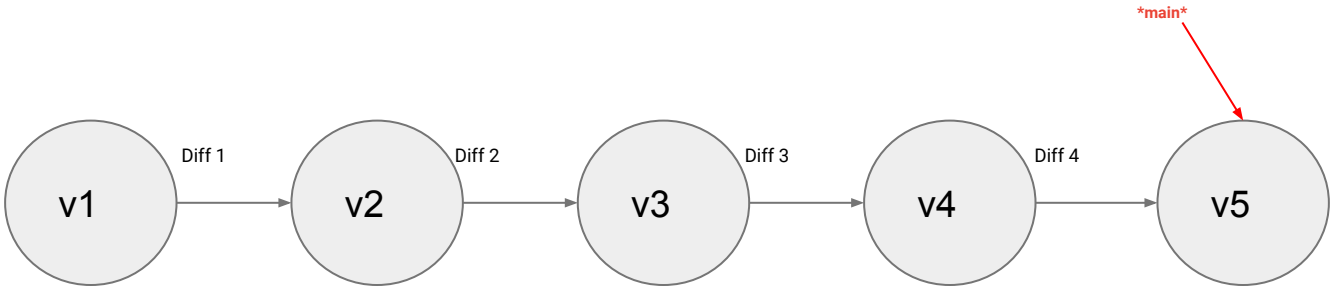
- More terminology: “HEAD” – the commit where **I** am currently working
- Most of the time, HEAD points to a particular branch, so I’ll omit it from the diagrams and instead annotate where HEAD is pointing with asterisks



Git source control: Moving around

- How do I move my pointer around?

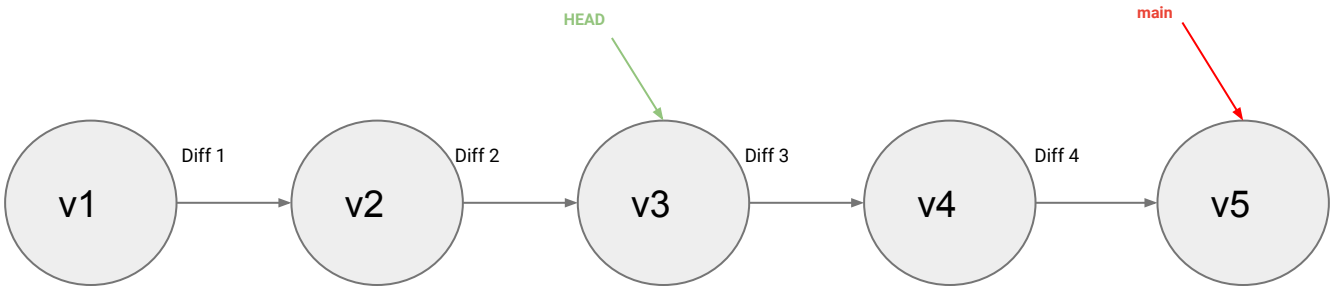
```
$ git checkout v3
```



Git source control: Moving around

- How do I move my pointer around?

```
$ git checkout v3
```

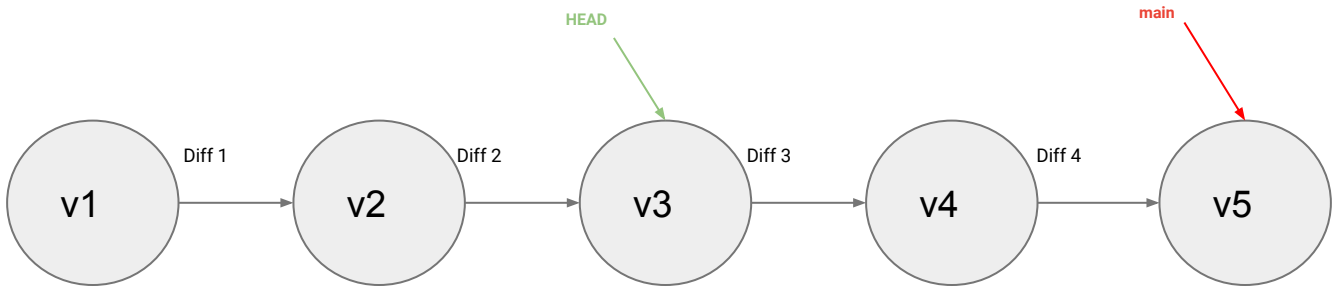


Git source control: Moving around

- How do I move my pointer around?

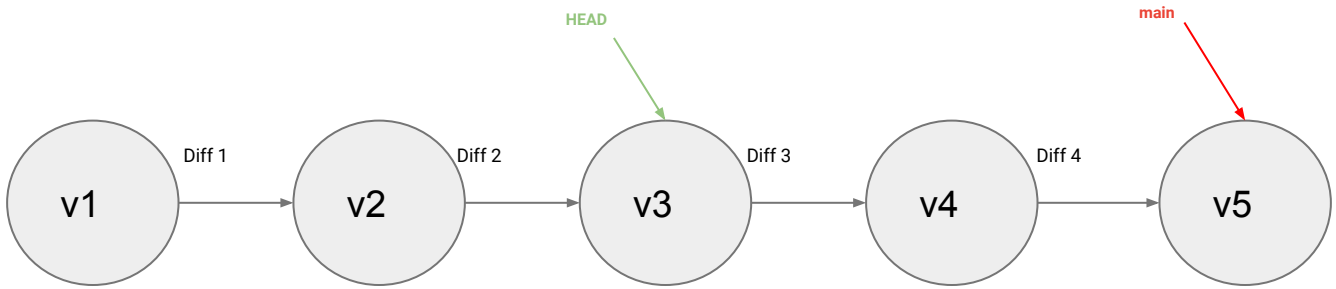
```
$ git checkout v3
```

In reality, more complicated. (commit names, relative refs, go do the tutorial!)



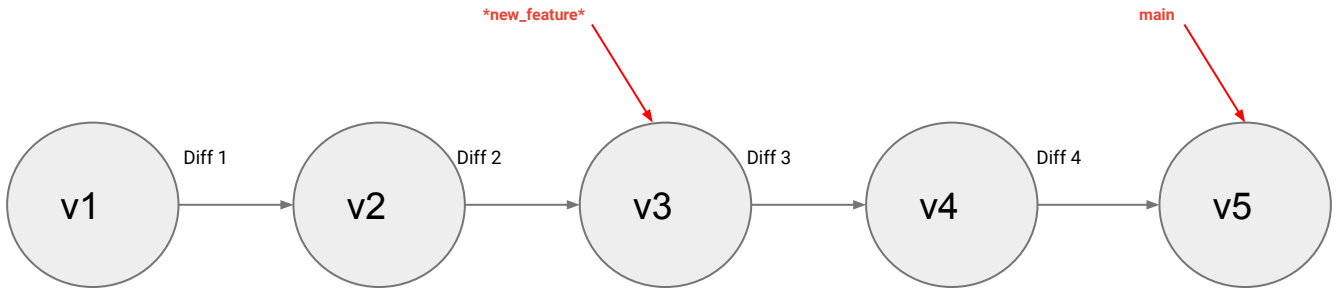
Git source control: List turns into Tree

```
$ git checkout -b new_feature
```



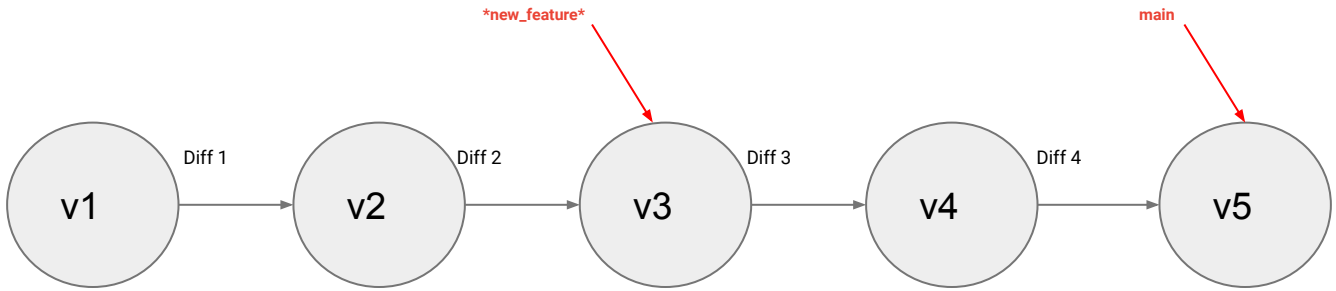
Git source control: List turns into Tree

```
$ git checkout -b new_feature
```



Git source control: List turns into Tree

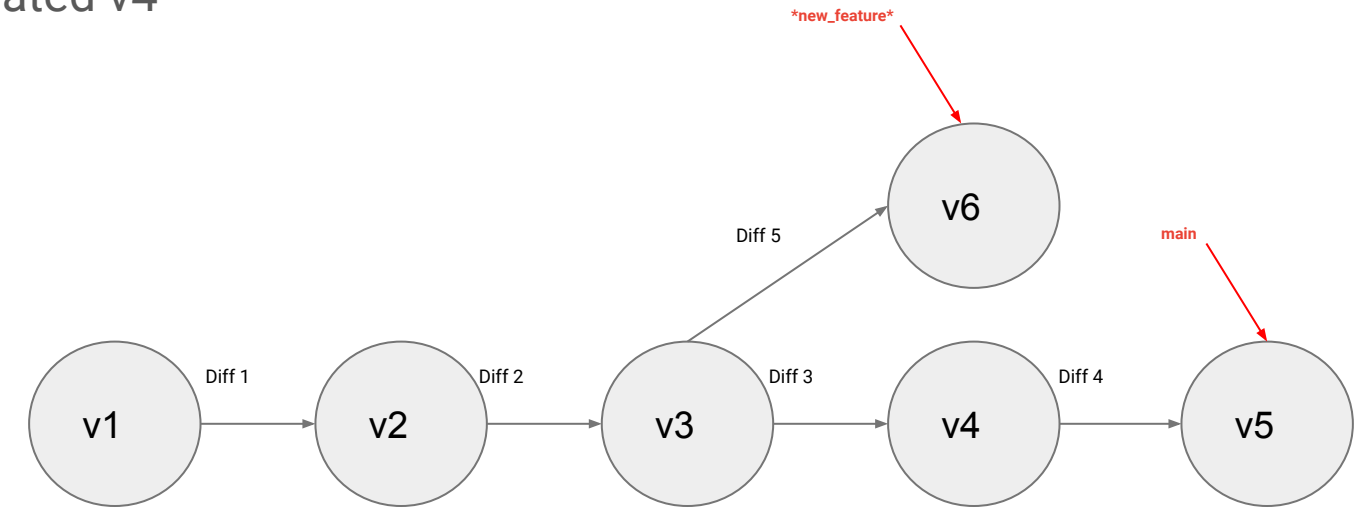
```
$ git commit -m “Adding a new feature”
```



Git source control: List turns into Tree

```
$ git commit -m "Adding a new feature"
```

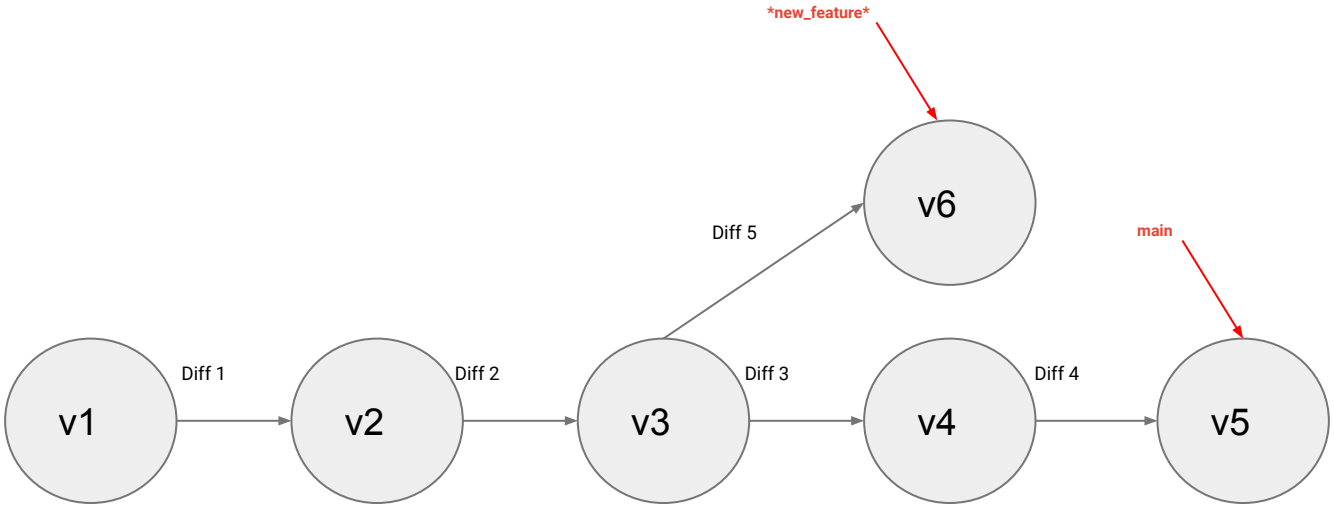
Uh oh. We committed on top of v3 with a diff that wasn't the same as the one that already created v4



Git source control: List turns into Tree

This is not the ideal state of the repository.

How do I know what the most up-to-date version is to start building on?

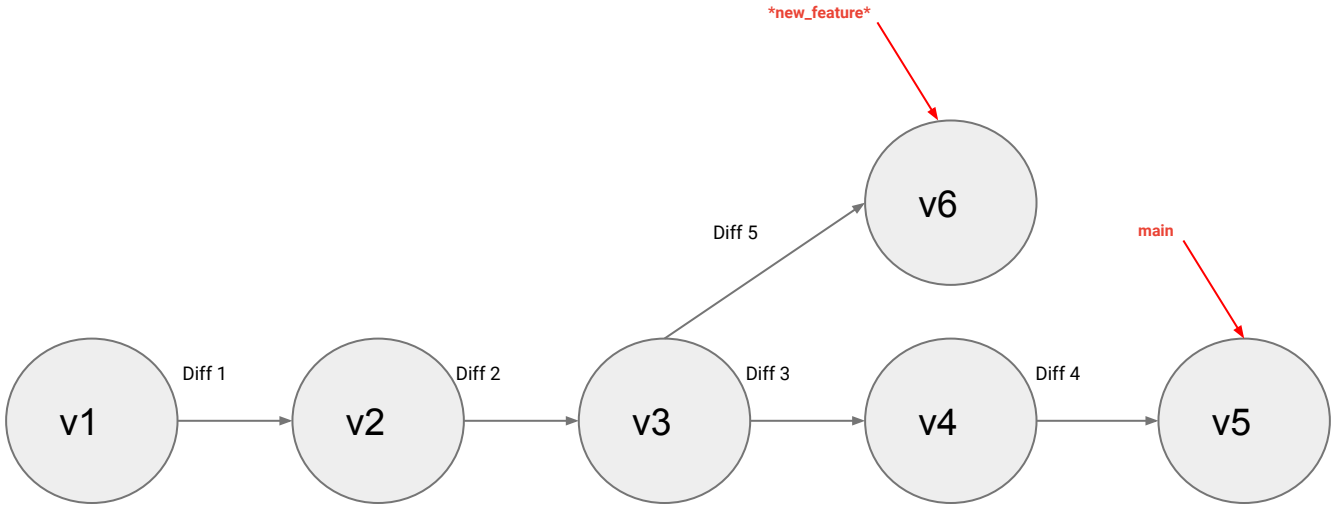


Side note: “branch” still means pointer to a commit! Can be confusing to conflate the branch of the tree a particular git branch is pointing to

Git source control: Tree turns into graph

Terminology: “Merge” create a commit with 2 parents

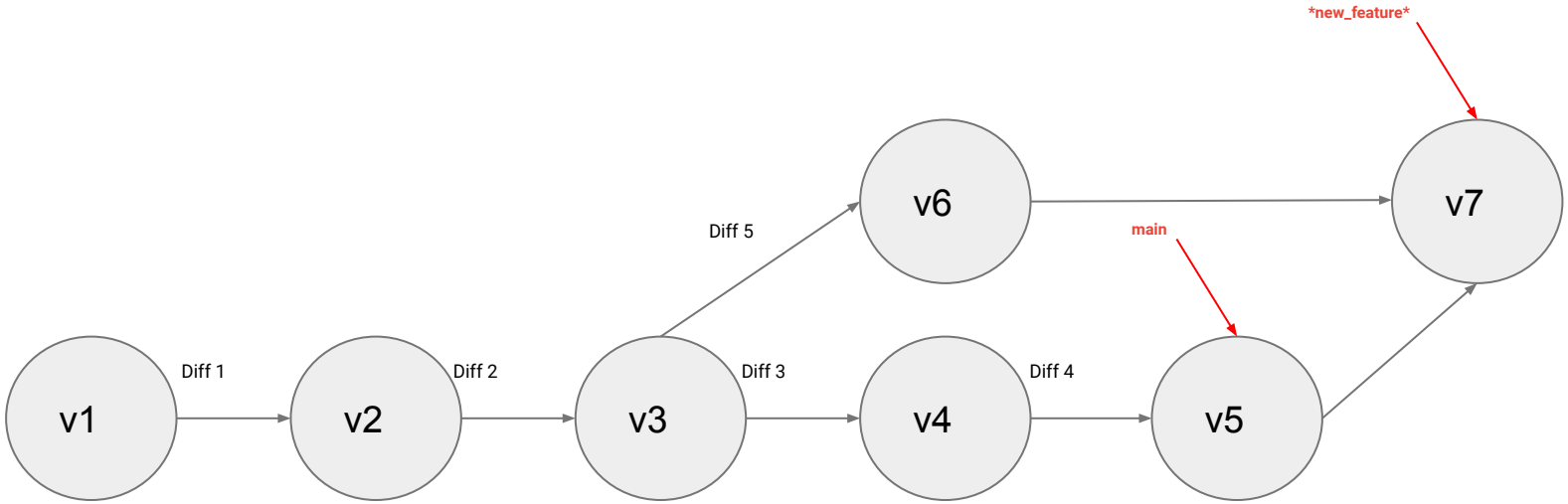
```
$ git merge main
```



Git source control: Tree turns into graph

Terminology: “Merge” create a commit with 2 parents

```
$ git merge main
```



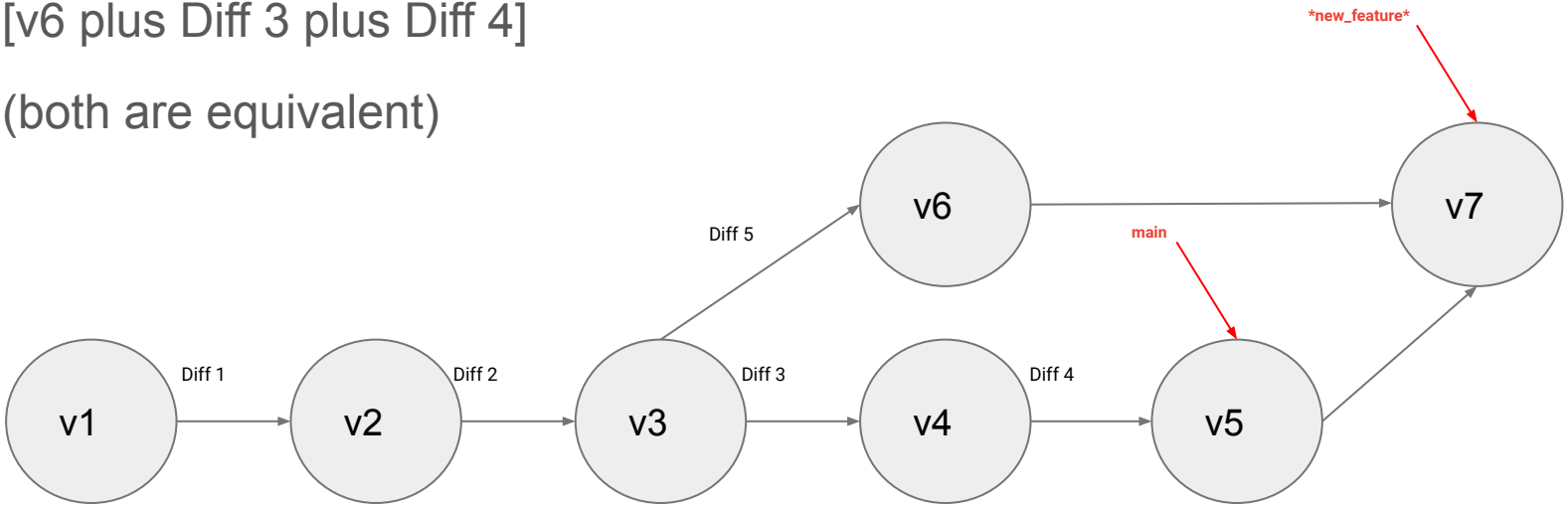
Git source control: Tree turns into graph

What's in v7?

[v5 plus Diff 5] OR

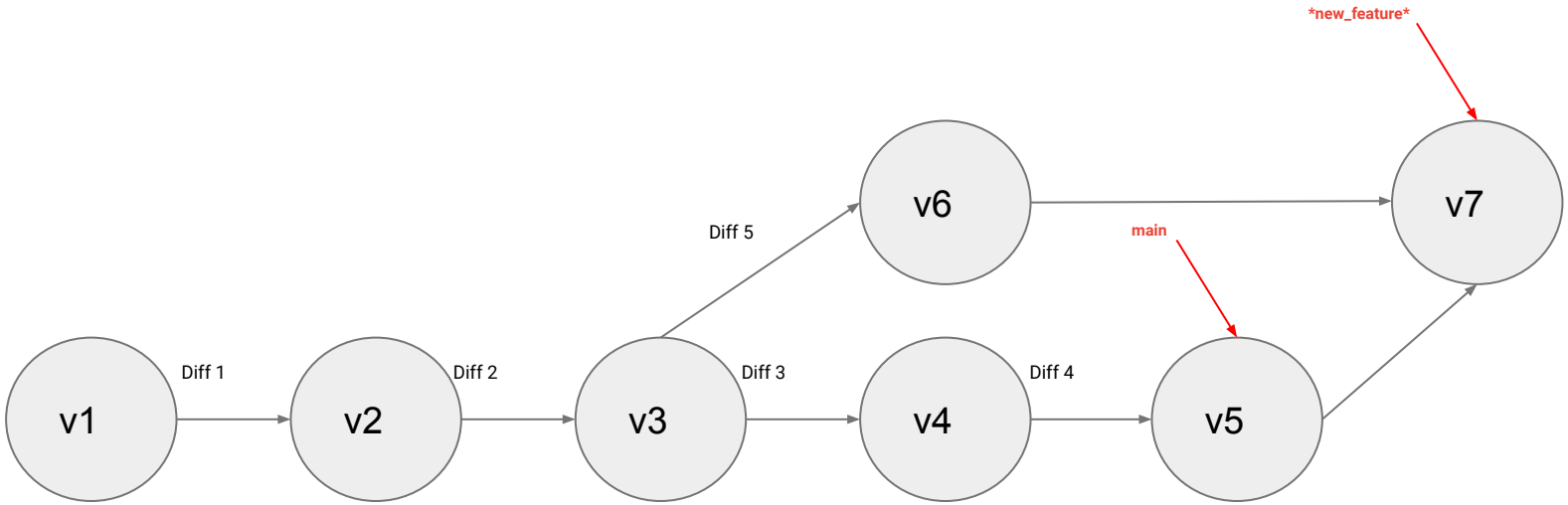
[v6 plus Diff 3 plus Diff 4]

(both are equivalent)



Git source control: Pruning the tree

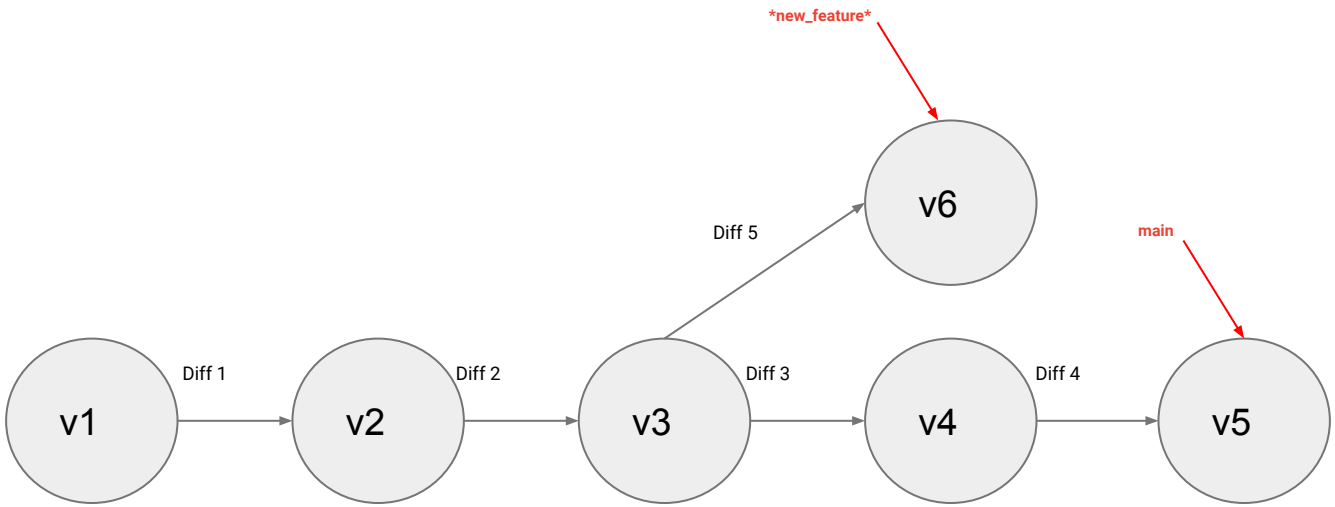
This feels a bit awkward. Can we keep things as having one parent each?



Git source control: Pruning the tree

Yes! This is where rebase comes in

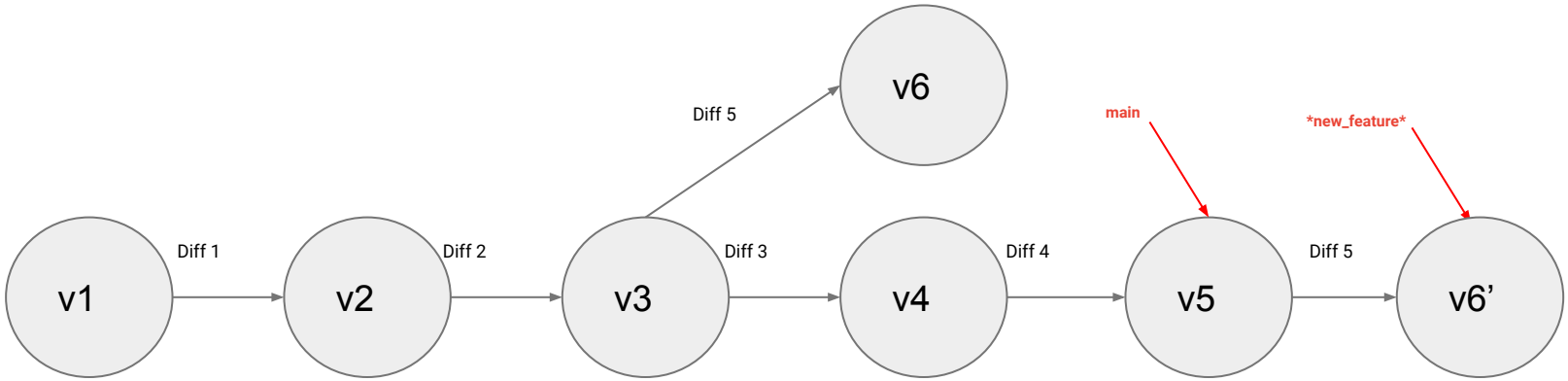
```
$ git rebase main
```



Git source control: Pruning the tree

Yes! This is where rebase comes in

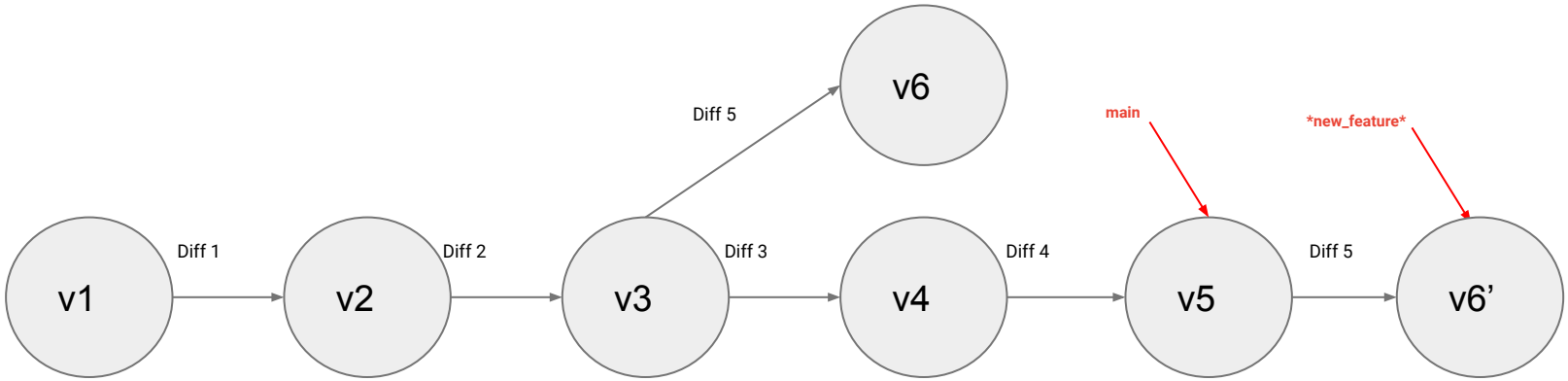
```
$ git rebase main
```



Git source control: Pruning the tree

Copies the diff on top of the commit you are rebasing onto

Leaves an orphan commit (v6)



Recommended tutorial

- https://learngitbranching.js.org/
- We'll show usage and concepts in this lecture, but this tutorial is **highly** recommended if you have not used git on a large project before
- Git can feel very complicated when first starting out
  a. It's easy to get lost in the terminology, and
  b. git does not surface any graph diagrams like the one shown in the prior slides, so it can be tough to figure out exactly what is happening in your live repository

Git source control: Basic usage (local)

`git init`

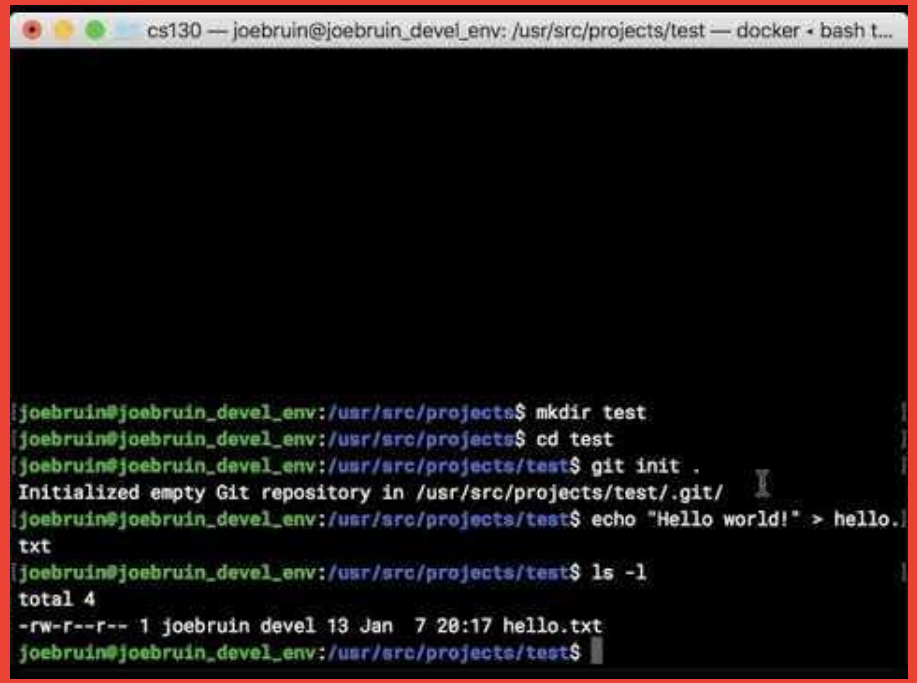
- Creates a blank repository locally, with data in a hidden `.git` directory.

`echo "Hello world!" > hello.txt`

- The files currently in the directory with the .git subdirectory can be edited at any time. The state of these files is the "working directory" state.

`git status`

- Output will show `hello.txt` is "untracked".



Git source control: Basic usage (local)

`git add hello.txt`

- Add a file to the "stage" (or "index") repository state.

`git status`

- Output will show `hello.txt` is staged for commit.

`git commit`

- Create a named historical state of the repository with staged changes.

Note: every commit requires a message that describes its contents (see the -m flag)

```
cs130 — joebruin@joebruin_devel_env: /usr/src/projects/test — docker • bash t...
On branch master

No commits yet

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

        hello.txt

nothing added to commit but untracked files present (use "git add" to track)
joebruin@joebruin_devel_env:/usr/src/projects/test$ git add hello.txt
joebruin@joebruin_devel_env:/usr/src/projects/test$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hello.txt

joebruin@joebruin_devel_env:/usr/src/projects/test$ git commit -m "My first comm
it"
```

Git with Gerrit (remote)

`git clone ssh://code.cs130...`

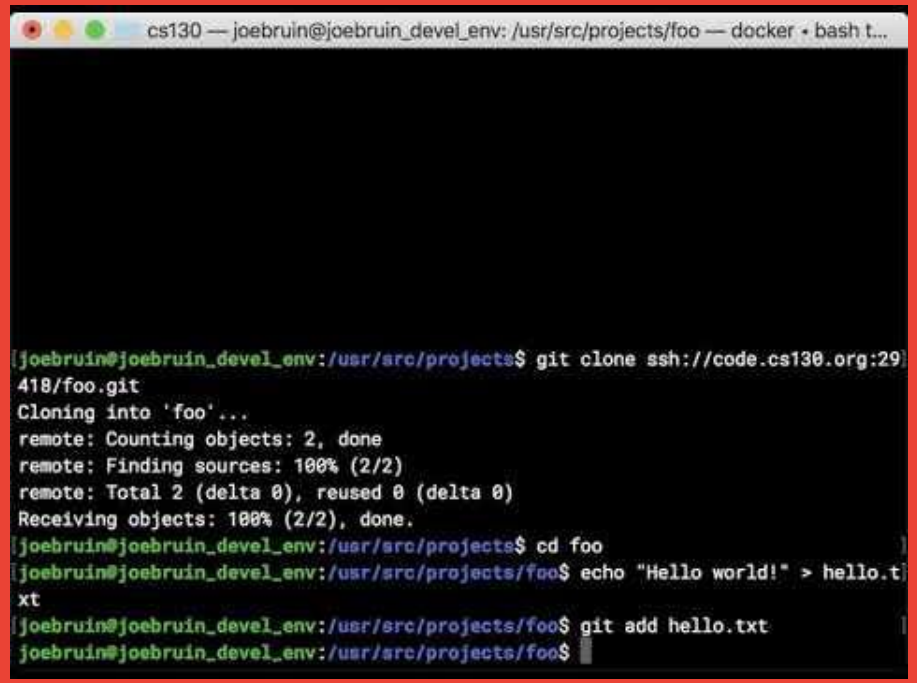
- Downloads a git repository from Gerrit.

`echo "Hello world!" > hello.txt`

- The files currently in the directory with the .git subdirectory can be edited at any time. The state of these files is the "working directory" state.

`git add hello.txt; git commit`

- Add the file. Commit the file.



Git with Gerrit (remote)

`git push`

- This step fails. Code must be reviewed!

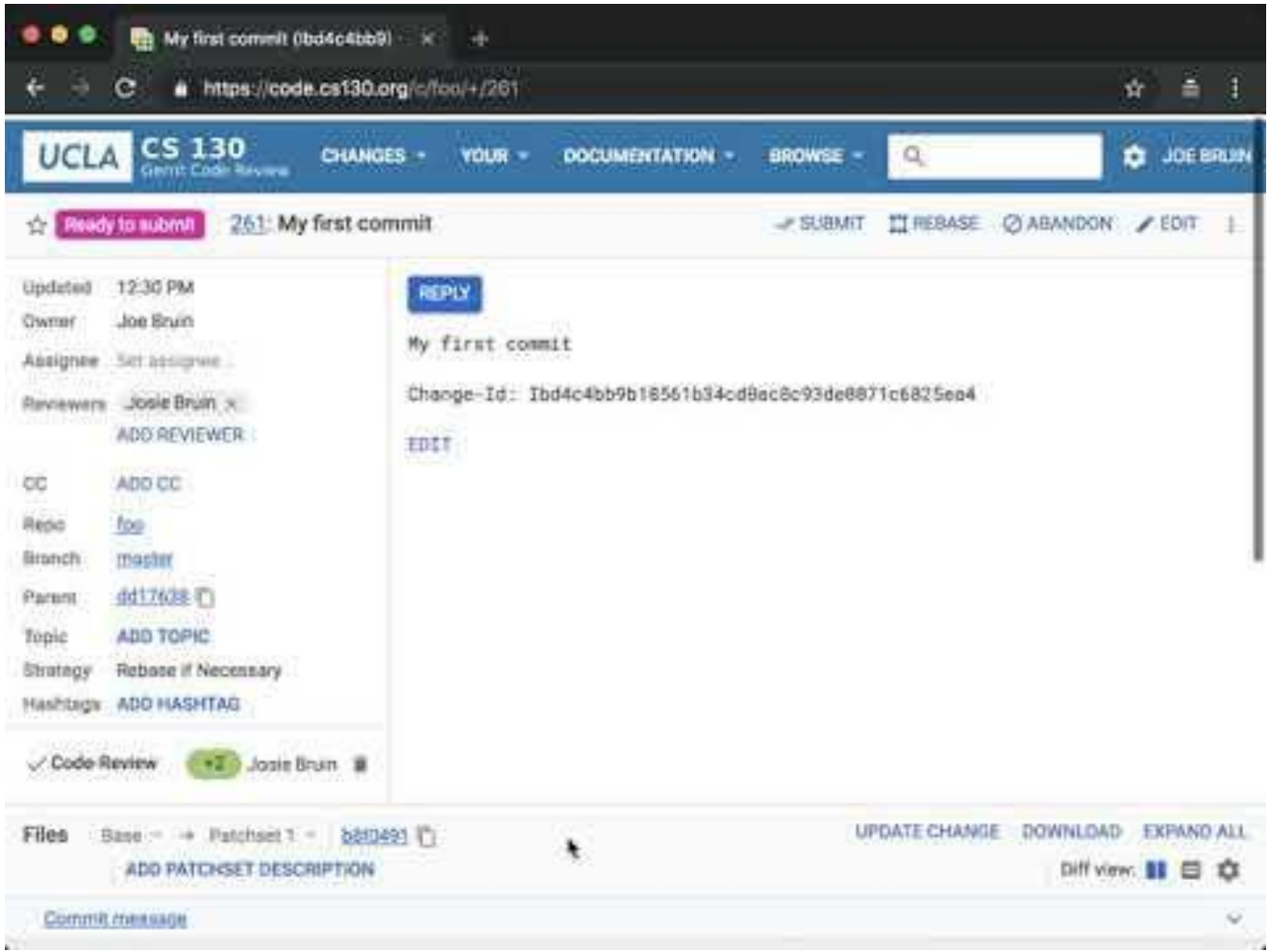
`git review`

- Submit code for review.

`git pull --rebase`

- Pull new changes from the remote repository.

```
cs130 — joebruin@joebruin_devel_env: /usr/src/projects/foo — docker • bash t...
    My first commit

commit dd17638586c8c178426f93508ba033698fe094d3 (origin/master, origin/HEAD)
Author: Joe Bruin <joebruin@cs130.org>
Date:   Mon Jan 7 09:21:38 2019 +0000

    Initial empty repository
joebruin@joebruin_devel_env:/usr/src/projects/foo$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 260 bytes | 18.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: error: branch refs/heads/master:
remote: To push into this reference you need 'Push' rights.
remote: User: joebruin
remote: Contact an administrator to fix the permissions
remote: Processing changes: refs: 1, done
To ssh://code.cs130.org:29418/foo.git
 ! [remote rejected] master -> master (prohibited by Gerrit: not permitted: upda
te)
error: failed to push some refs to 'ssh://code.cs130.org:29418/foo.git'
joebruin@joebruin_devel_env:/usr/src/projects/foo$ git review --reviewers josieb
ruin
```



Submitting in Gerrit

Git with Gerrit (remote)

`git push`

- This step fails. Code must be reviewed!

`git review`

- Submit code for review.

`git pull --rebase`

- Pull new changes from the remote repository.

```
cs130 — joebruin@joebruin_devel_env: /usr/src/projects/foo — docker • bash t...
remote: Pushing to refs/publish/* is deprecated, use refs/for/* instead.
To ssh://code.cs130.org:29418/foo.git
 * [new branch]      HEAD -> refs/publish/master%r=josiebruin
joebruin@joebruin_devel_env:/usr/src/projects/foo$ git log
commit b8f0491ed46e827af796cdcc2dbaf6f553d7c57f (HEAD -> master)
Author: Devel User <joebruin@cs130.org>
Date:   Mon Jan 7 20:23:43 2019 +0000

    My first commit

    Change-Id: Ibd4c4bb9b18561b34cd0ac8c93de0071c6825ea4

commit dd17638586c8c178426f93508ba033698fe094d3 (origin/master, origin/HEAD)
Author: Joe Bruin <joebruin@cs130.org>
Date:   Mon Jan 7 09:21:38 2019 +0000

    Initial empty repository
joebruin@joebruin_devel_env:/usr/src/projects/foo$ git pull --rebase
From ssh://code.cs130.org:29418/foo
   dd17638..b8f0491  master     -> origin/master
Already up to date.
Current branch master is up to date.
joebruin@joebruin_devel_env:/usr/src/projects/foo$
```

Git source control: Advice

- Keep branch structure simple
- Develop in branches (don't touch `main`)
- Locally, use branches to do incremental development
- Get familiar with merge and rebase. Use them to manage how your commits to the `main` branch look.
  - Hint, prefer *rebase*
- For class: Always send code for review, and submit in Gerrit web UI

Coming up

Assignment 1

Assigned today, due next Monday

Next lecture

Testing

Checking Out

https://forms.gle/skuXEv1fuea5gJxu7

One word: How do you feel now?

A Tweet: What excites you about this class?