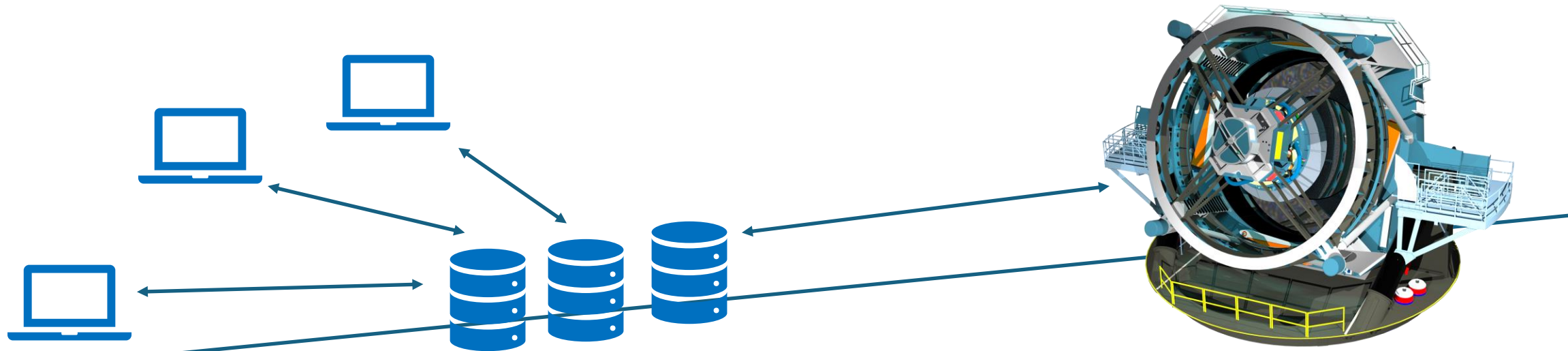


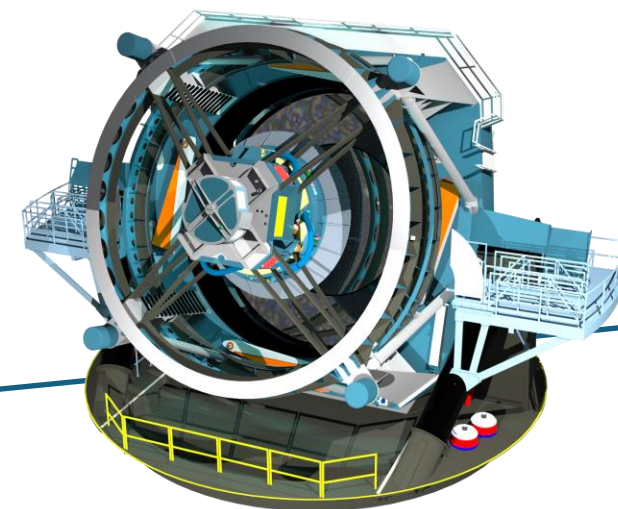
Introduction to Software Repositories

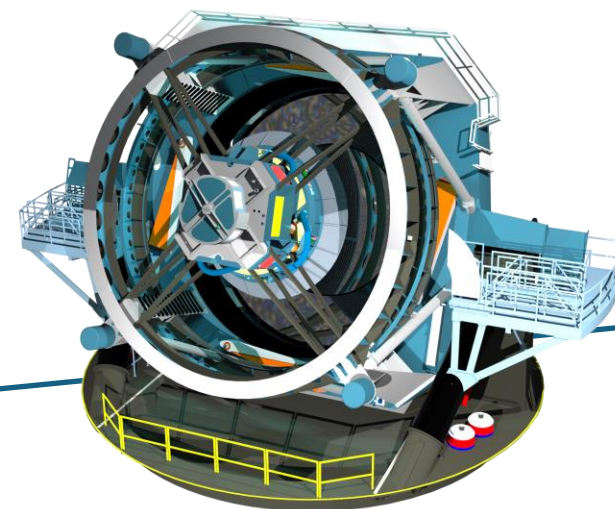
Bryan Scott

LSST-DA Data Science Fellowship Program Session 21

University of Illinois, Urbana-Champaign



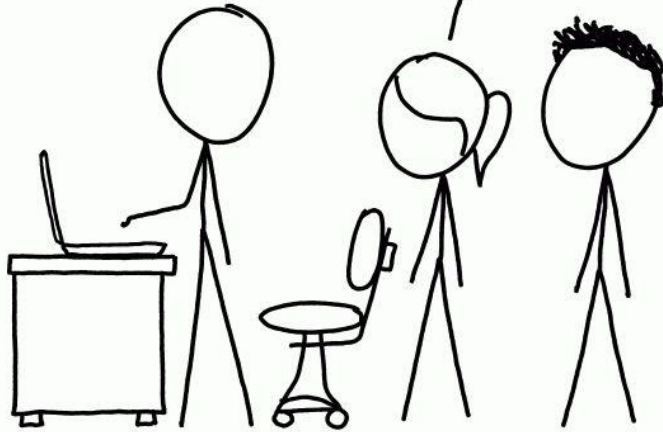




THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



Goals for Today:

Understand Distributed Version Control
Systems

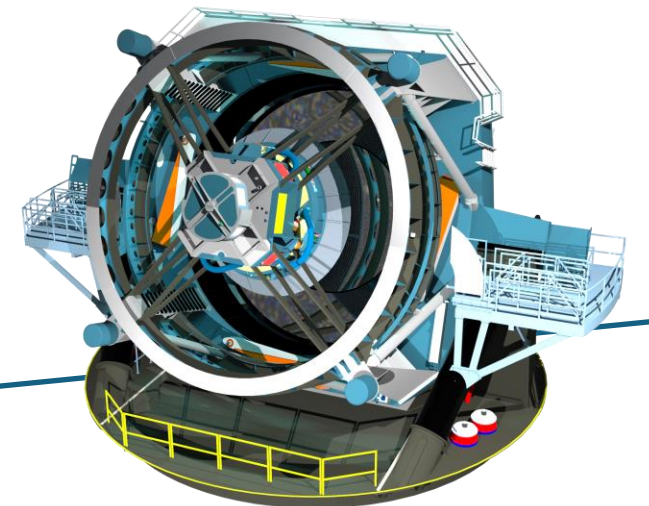
How (and why) to use git

How git works:

"Directed Acyclic Graphs"

Role in Software
Development Workflows

Mechanics of using git
(understanding *why* each
command is used)



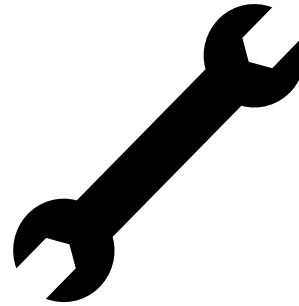
Let's be real - development workflows...



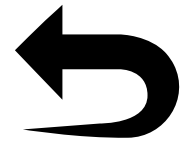
Brilliant
Idea!



Prototyping



Fixes



Oh no!

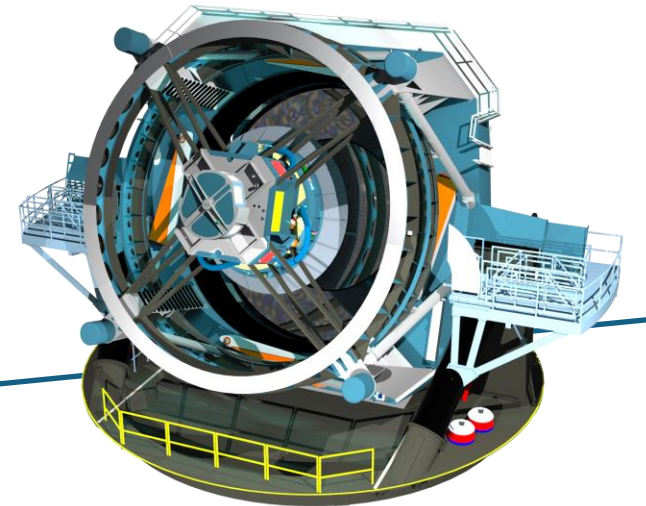
Problems in typical workflows

- ❏ MG_IM_models_revised_2.py
- ❏ MG_IM_models_revised_3.py
- ❏ MG_IM_models_revised_4.py
- ❏ MG_IM_models_revised_5.py
- ❏ MG_IM_models_revised_6.py
- ❏ MG_IM_models_revised_7_test.py
- ❏ MG_IM_models_revised_7.py
- ❏ MG IM models revised.py

Paper_draft_final.tex
Paper_draft_final_revised.tex
Paper_draft_final_revised_final.tex
Paper_draft_final_revised_final_submission.tex
Paper_draft_final_revised_final_submission_with_comments.tex
Paper_draft_final_revised_final_submission_with_comments_and_responses_draft.tex
(and so on....)

Version control is a *system* for creating
a *reproducible* record of changes to a
Project.

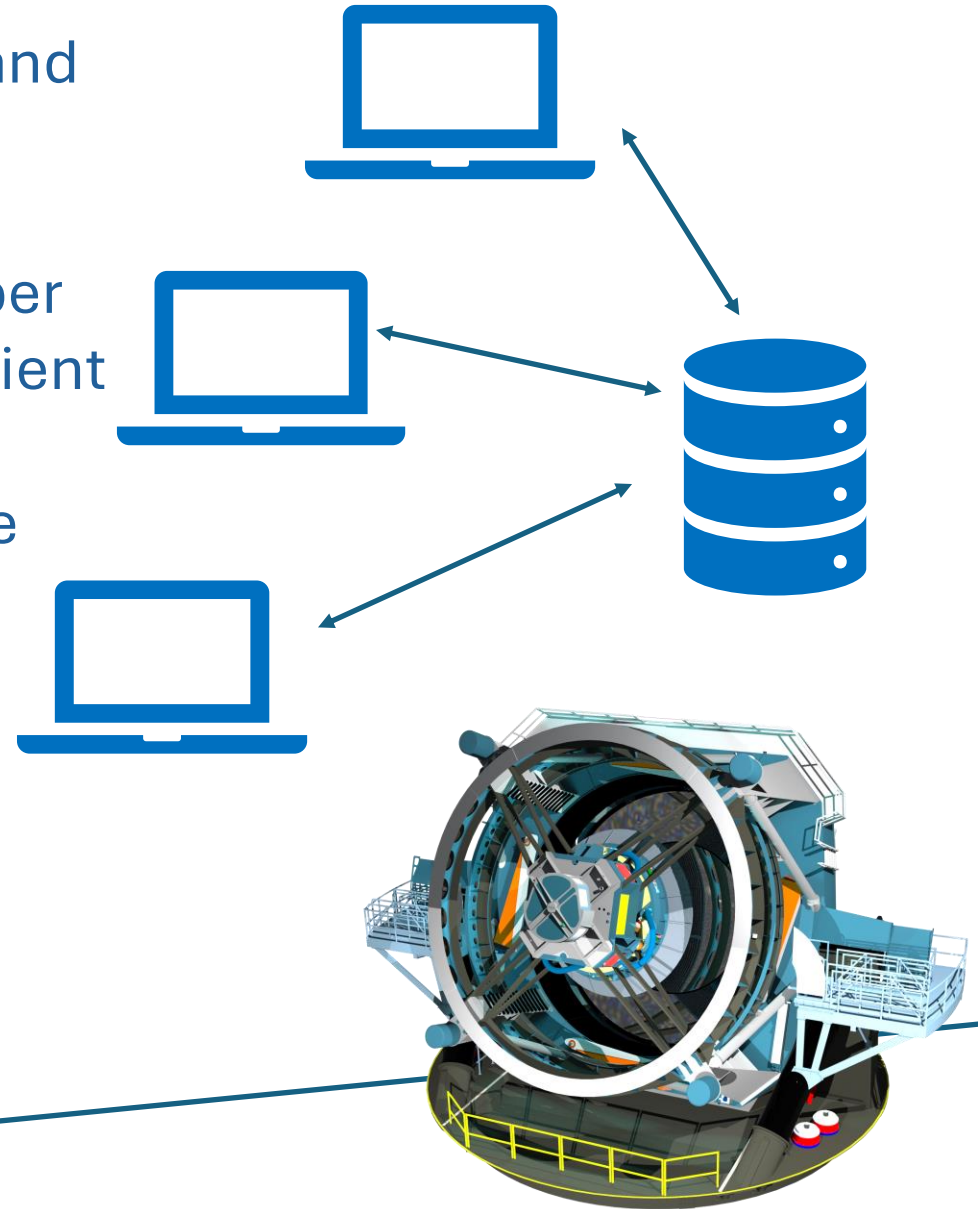
The BIG idea is that there is ONE project.



A software repository is a place for storing software and metadata about it.

In a distributed version control system, each developer has a copy of the repository – while it may be convenient to choose a "central" copy as a way of aiding collaboration – there is no authoritative version of the repository.

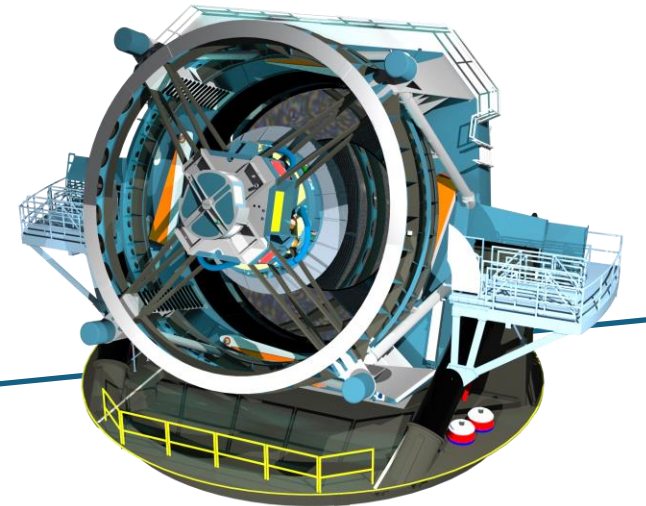
Git is an example of a distributed version control system.

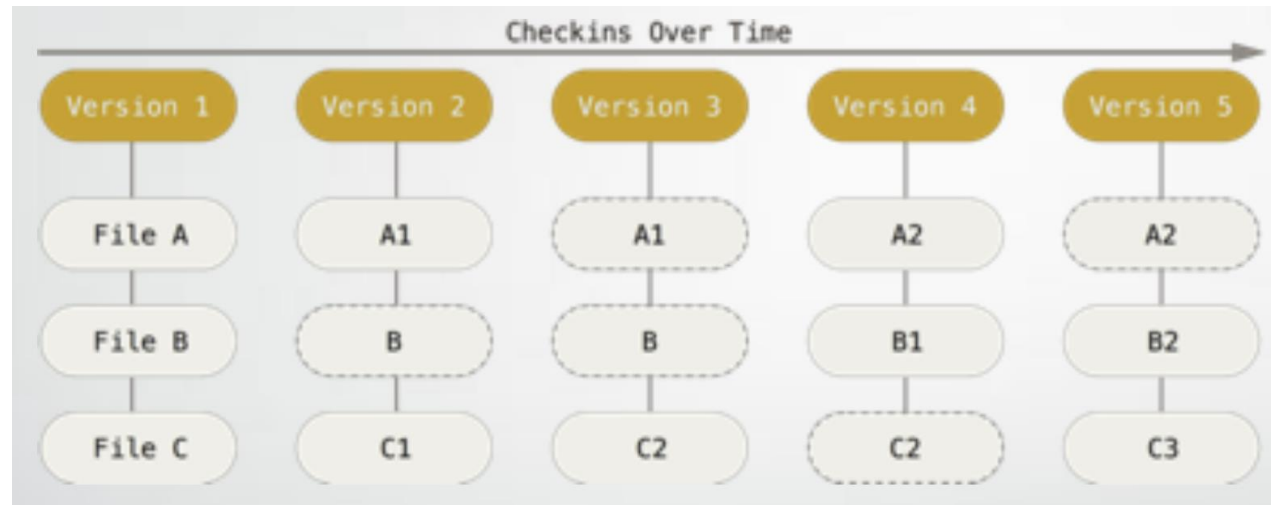


When and why are software repositories helpful...

For "small projects", version control makes your work more reproducible and gives you a "global undo".

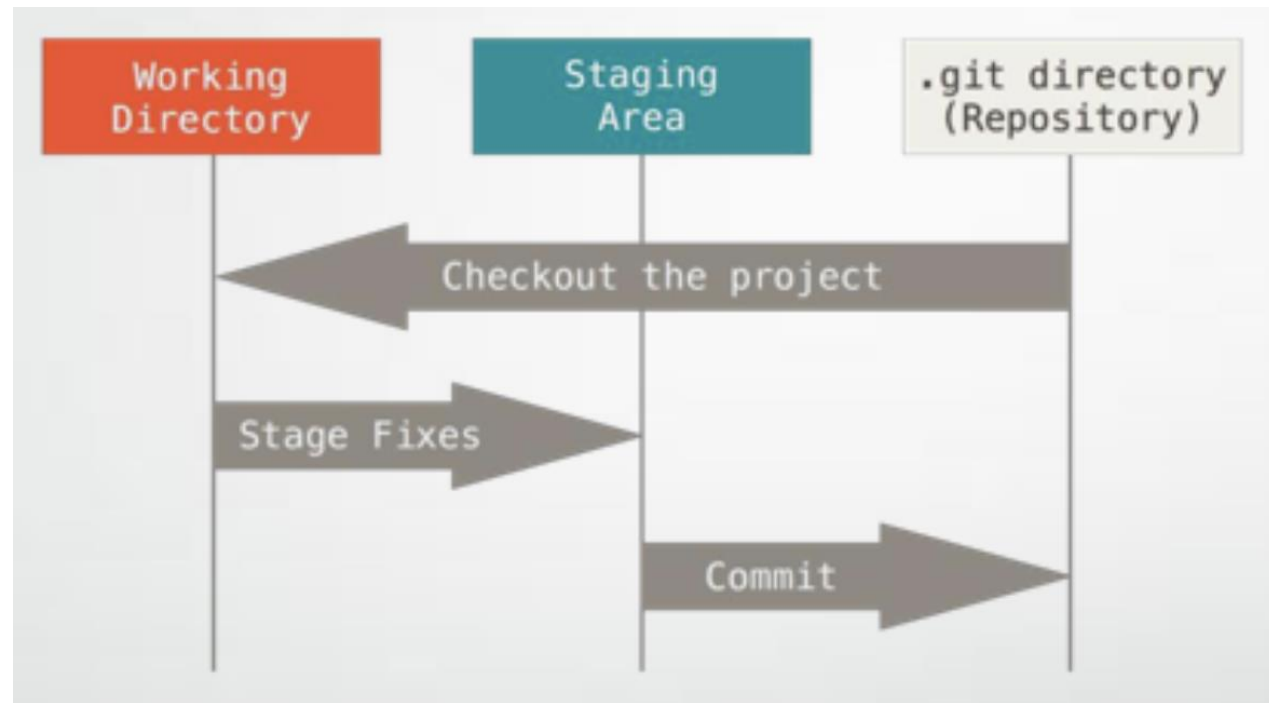
For "large projects", a distributed VCS can allow multiple developers to work independently and synchronously.





Git stores snapshots of the project at different times. If a new file is created or a change is made, git will add that change to the next snapshot of the project. If a file is unchanged, then git will simply link to the version of the file in the previous snapshot.





Files can be in one of three states. *Modified*, *staged*, and *committed*. Files move between the three stages as you work. After editing a file it is *modified*. You then *stage* the file which tells git to include it in the next snapshot. You then *commit* the changes (save a snapshot of the project).

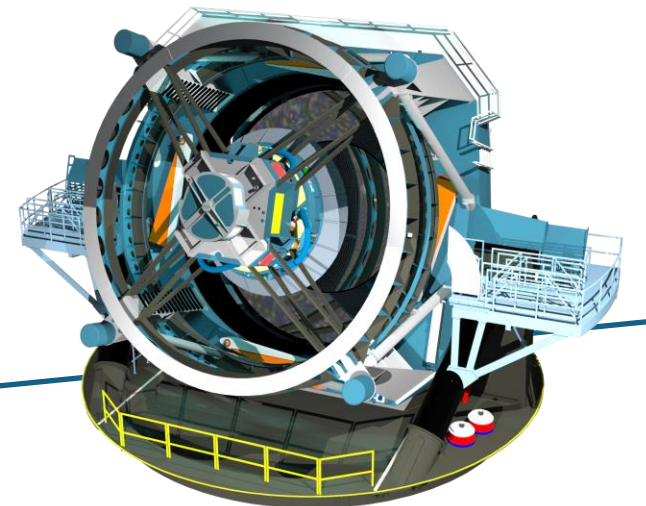


Let's practice this...

Create a directory that will become a git repo. Now, open a terminal/command prompt and type:

```
git init
```

Then create a text file `readme.md` and add some text to it. Do this using your favorite text editor.

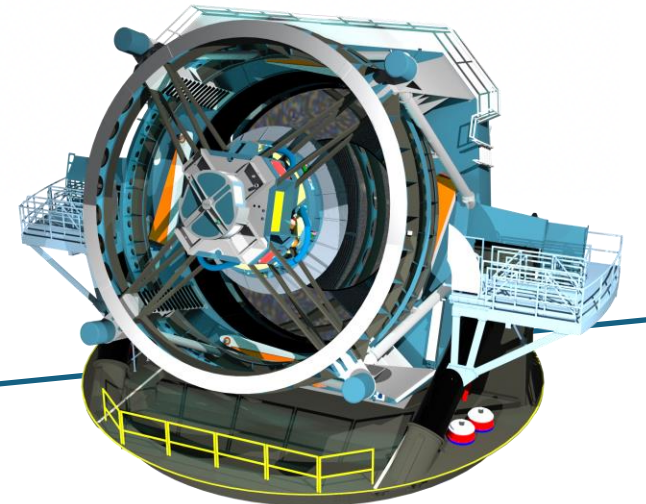


The git *init* command

```
Last login: Sat Jun  1 13:43:12 on ttys002
[(base) bryan@Bryans-MacBook-Pro ~ % cd /Users/bryan/Documents/DSFP/Session21-github
[(base) bryan@Bryans-MacBook-Pro Session21-github % git init
Initialized empty Git repository in /Users/bryan/Documents/DSFP/Session21-github/.git/
[(base) bryan@Bryans-MacBook-Pro Session21-github % git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
(base) bryan@Bryans-MacBook-Pro Session21-github % █
```



The *git status* command

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % touch readme.md
```

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git status
```

```
On branch main
```

```
No commits yet
```

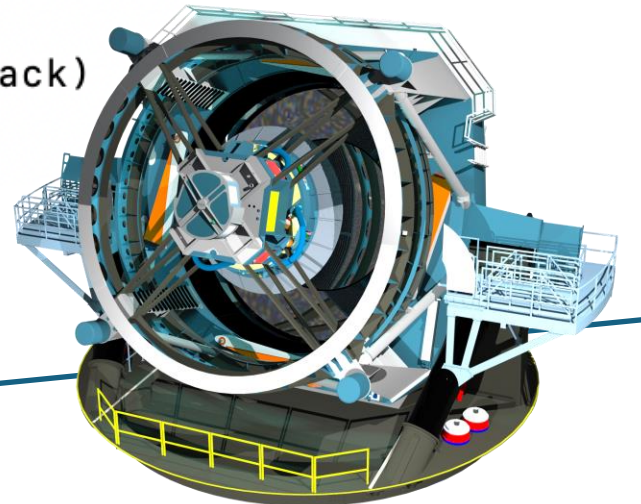
```
Untracked files:
```

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

```
    readme.md
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

```
(base) bryan@Bryans-MacBook-Pro Session21-github % █
```



Let's practice this...

Now we tell git to track this file. We do this with

```
git add readme.md
```

This stages readme.md so that git will include it in the next commit. Now we add them to the repository with:

```
git commit -m "adding readme to our repository"
```



The git *commit* and *log* commands

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % touch readme.md
[(base) bryan@Bryans-MacBook-Pro Session21-github % git status
On branch main

No commits yet

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

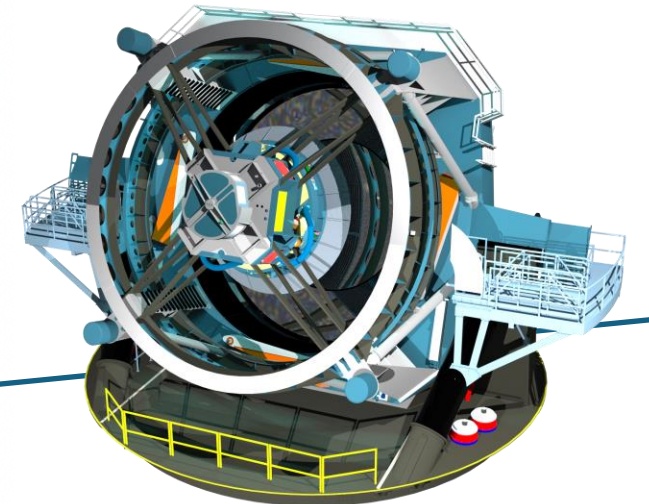
nothing added to commit but untracked files present (use "git add" to track)
[(base) bryan@Bryans-MacBook-Pro Session21-github % vim readme.md
[(base) bryan@Bryans-MacBook-Pro Session21-github % git add readme.md
[(base) bryan@Bryans-MacBook-Pro Session21-github % git status
On branch main

No commits yet

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

[(base) bryan@Bryans-MacBook-Pro Session21-github % git commit -m "adding readme"
[main (root-commit) 2cf4d5d] adding readme
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
[(base) bryan@Bryans-MacBook-Pro Session21-github % git log
commit 2cf4d5d4b361dc0c54045e8c232b343153593230 (HEAD -> main)
Author: Bryan Scott <36455167+bscot@users.noreply.github.com>
Date:   Sat Jun 1 14:02:57 2024 -0500

    adding readme
(base) bryan@Bryans-MacBook-Pro Session21-github %
```



Commit-ing modified files

Suppose we change a previously commit-ed file – what does git status looks like.

```
adding README
((base) bryan@Bryans-MacBook-Pro Session21-github % vim readme.md
((base) bryan@Bryans-MacBook-Pro Session21-github % git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.md

no changes added to commit (use "git add" and/or "git commit -a")
((base) bryan@Bryans-MacBook-Pro Session21-github % █
```

A word of caution: you need to *git add* every time you make a change to a file.

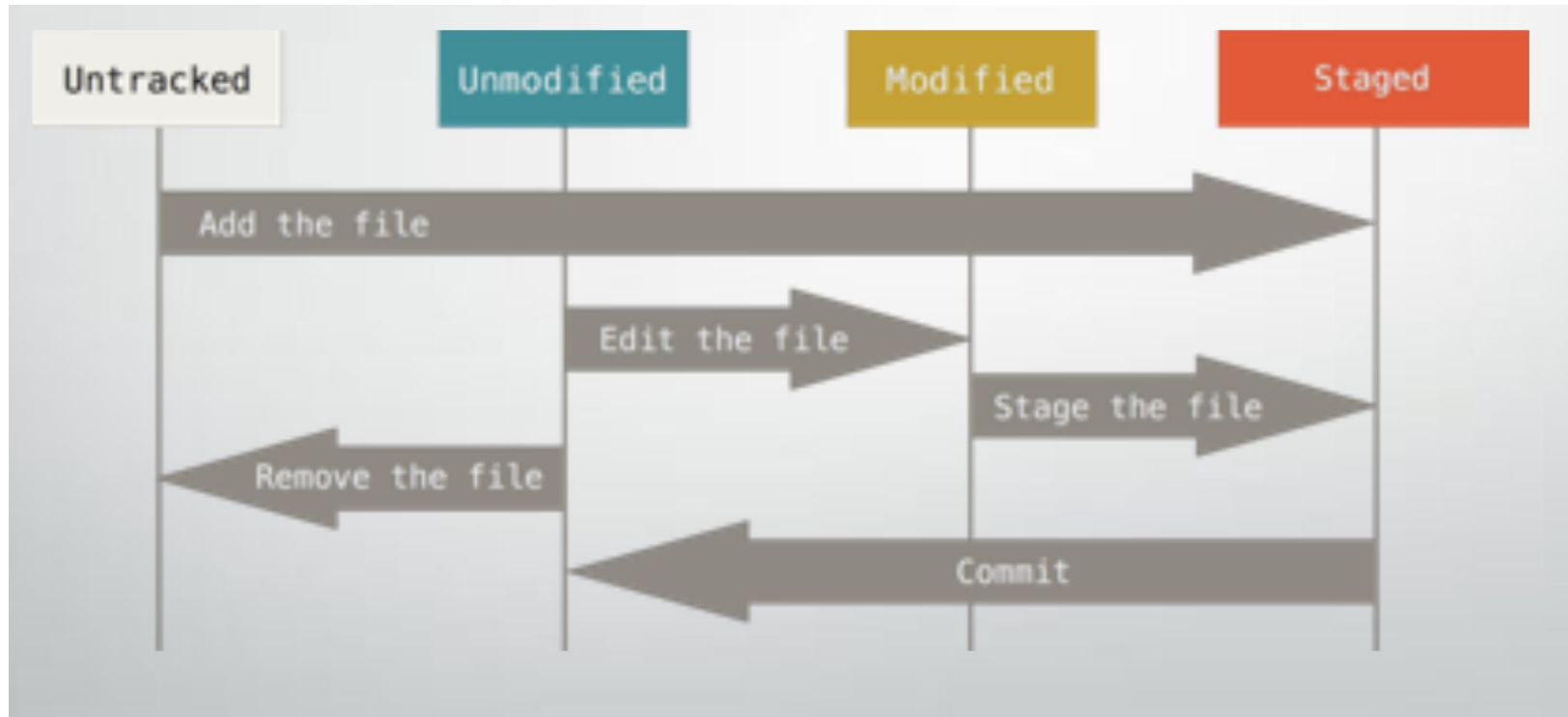


Files in your project can be in one of 2 states – *tracked* or *untracked*

Git tracks all files that were in the last snapshot (*commit*) or that have been created and staged with the git *add* command.

To add changed files to the repository, you need to stage them with *add* and then snapshot with *commit*





Git diff

Tells us exactly what is changed between the last time files were changed and what is in the working directory.

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git diff
diff --git a/readme.md b/readme.md
index 75a1b66..de2e673 100644
--- a/readme.md
+++ b/readme.md
@@ -1,1 @@
-This is a new repository for a git lesson
+This is a new repository for a git lesson (and its very awesome)
(base) bryan@Bryans-MacBook-Pro Session21-github %
```

Caution: git diff doesn't show all changes, only those that have **not** been staged!

.gitignore files tell git not to track files


```
# ignore all .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in any directory named build
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

More on Commits

```
git commit
```

Without a flag will open the text editor for you to add a message describing this commit in the git log.

Some useful flags are `-v`, `-m`, and `-a`. `-v` outputs the diff to the command line so you can see what changes are included. `-m` allows you to add a commit message without opening the editor. `-a` stages and commits at the same time.



Removing files

```
git rm [file]
```

This is the opposite of `git add` – it stages a file to be removed from the next snapshot.

If a file has already been staged or modified, you must force the operation with `-f`. This prevents accidental deletion of a file such that it can't be recovered from a previous commit.

The `-cached` flag removes a file from tracking without deleting it locally.

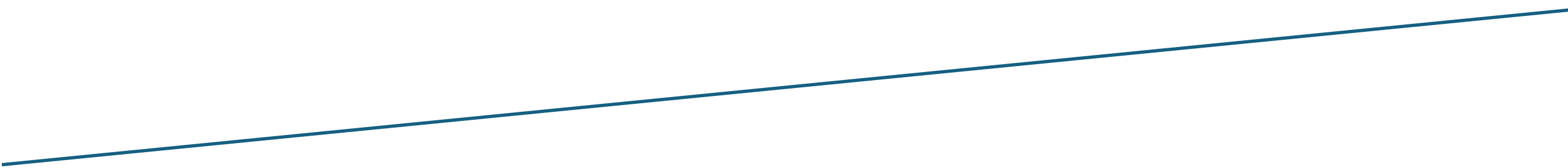


Commit History

At the cost of a more complex workflow, we now have a complete record – a directed acyclic graph - of changes to our project. To view this:

```
git log --oneline --graph
```

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git log --oneline --graph
* e74f775 (HEAD -> main, new_branch) added some more text to the readme on this new branch
* 2cf4d5d adding readme
(base) bryan@Bryans-MacBook-Pro Session21-github %
```



Git as a distributed version control system

Up until now, we've only considered a local copy of the software repository on our machine.

But there is nothing special about our local copy. We can make n copies to share across many developers.

We'll need to figure out how to manage n copies, but the power to collaborate is obvious! This is where github or gitlab come in.



Create a remote repository on github

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

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *



bscot ▾

Repository name *

Session21-github-lesson

✔ Session21-github-lesson is available.

Great repository names are short and memorable. Need inspiration? How about [glowing-rotary-phone](#) ?

Description (optional)



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:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

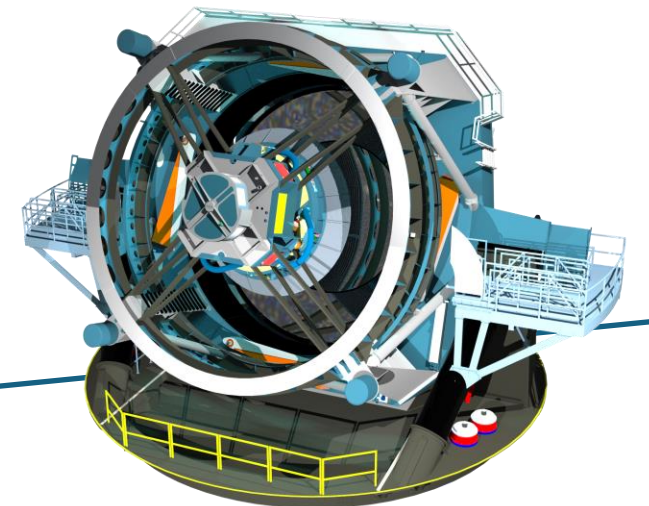
Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

You are creating a public repository in your personal account.

Create repository



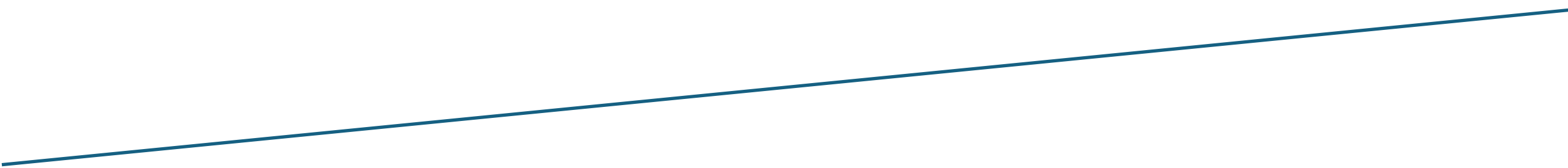
Link the local and remote repositories

To link your local and remote repositories:

```
git remote add origin [link to your github repo]
```

Then we can send our (staged) changes to the remote with the famous command:

```
git push origin main
```



Notes on naming conventions

By convention, main is the name of the working *branch* – or version – you push from on your local machine. We will talk about branching in a minute. *The word "master" was commonly used for this until recently and you may find references to it in some out of date references.*

Origin is the name of your remote branch.

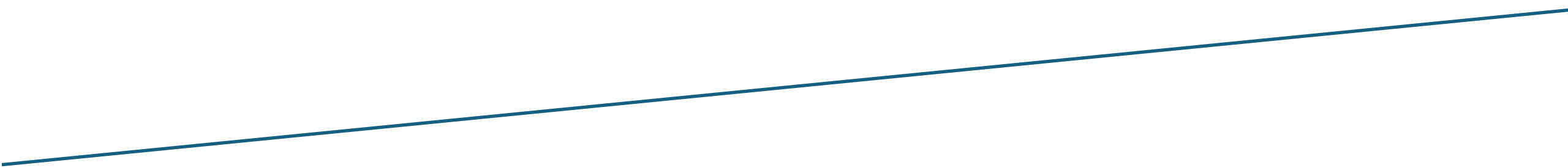
Upstream is used for a repository shared among multiple developers.



"To branch or not to branch, that is the question."

Let's say we want to add a new feature to our code. We don't want to impact our previous version with bugs that the new feature introduces.

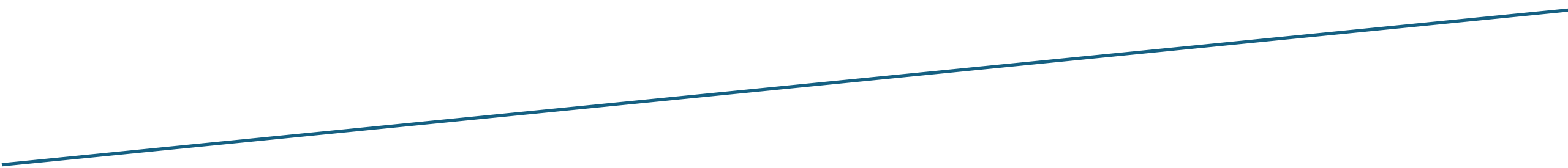
We could copy the project and work on a whole new copy. Maybe `project_revised.py` would be a new file in the copy. Or we could just branch the project – recognizing that we're still working on the same project, just adding something new.



What is a branch? (really)

What git stores is a snapshot of the project with references to files in previous commits and copies of changed files in the current commit.

A branch is just a reference to a specific commit – the parent. Each commit has at least one parent. As you make commits, the reference to the parent moves along the branch. The HEAD variable points to the name of the branch we are currently on. Any commits you make will have the HEAD reference as the parent.



Git Branches...

```
git branch [branch name]
```

```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git log  
commit e74f7754aed8ea010ce9f9cd57387e21f02bdd64 (HEAD -> main, new_feature, new_branch)  
Author: Bryan Scott <36455167+bscot@users.noreply.github.com>  
Date: Sat Jun 1 14:13:04 2024 -0500
```

```
    added some more text to the readme on this new branch
```

```
commit 2cf4d5d4b361dc0c54045e8c232b343153593230  
Author: Bryan Scott <36455167+bscot@users.noreply.github.com>  
Date: Sat Jun 1 14:02:57 2024 -0500
```

```
    adding readme
```

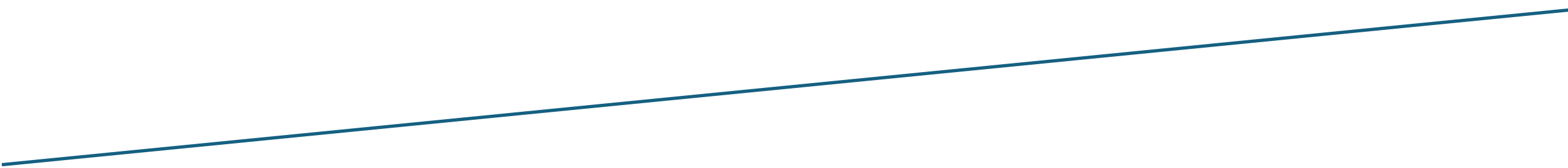
```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git checkout new_feature  
Switched to branch 'new_feature'  
[(base) bryan@Bryans-MacBook-Pro Session21-github % git status  
On branch new_feature  
nothing to commit, working tree clean  
[(base) bryan@Bryans-MacBook-Pro Session21-github % █
```

Switching to a branch

```
git checkout [branch name]
```

Now that we've created a branch, we need to switch to it in order to add our new feature. We change branches with the git checkout command.

This is very powerful – it allows us to make changes to part of the project without impacting other parts. Or revert back if we make a catastrophic change.

A thin blue diagonal line starts from the bottom left corner and extends towards the top right corner, spanning the width of the slide.

Git checkout:



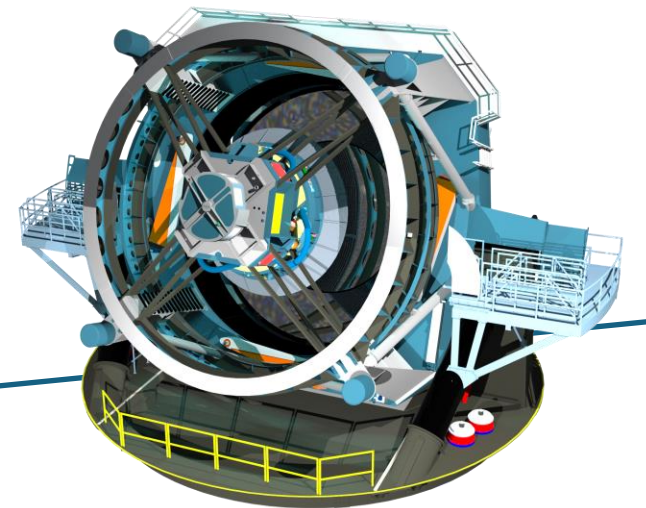
BACK
TO **FUTURE**
THE



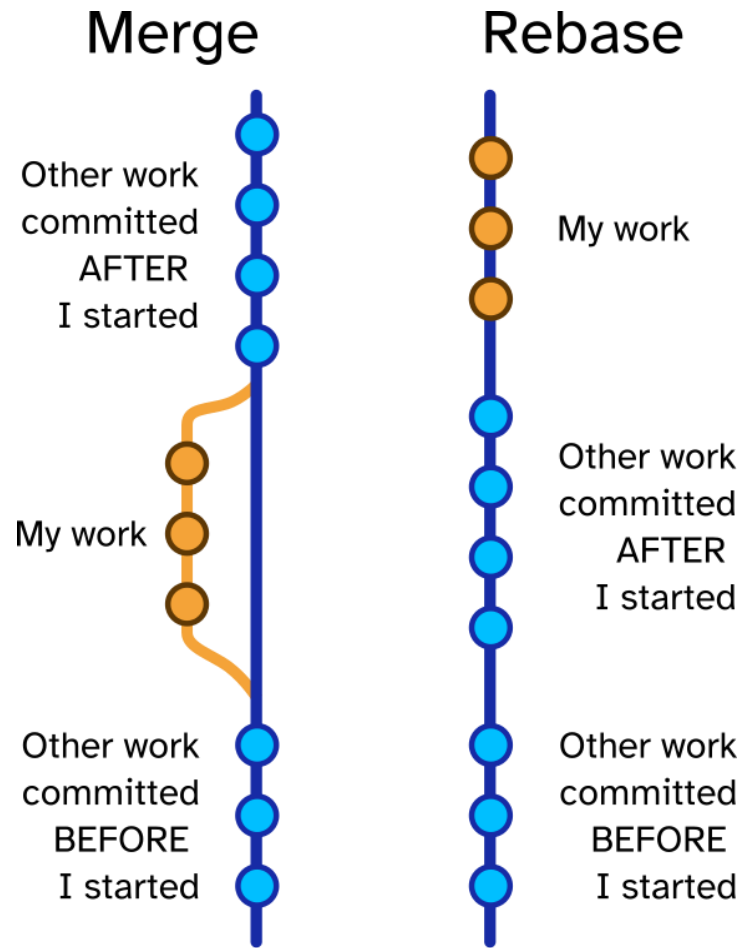
Merges

```
git merge [name of branch to be merged]
```

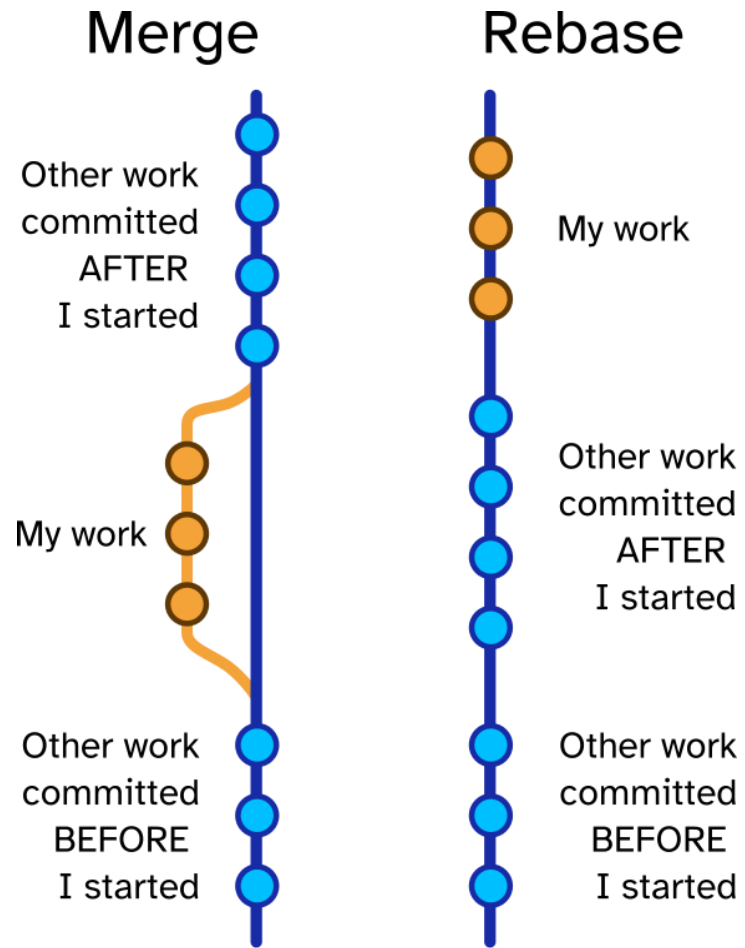
We can merge two branches together. Merging reproduces all of the commits on one branch in another.



Git rebase:



Git rebase:



Charles L Flatt, <https://www.softwaremeadows.com>

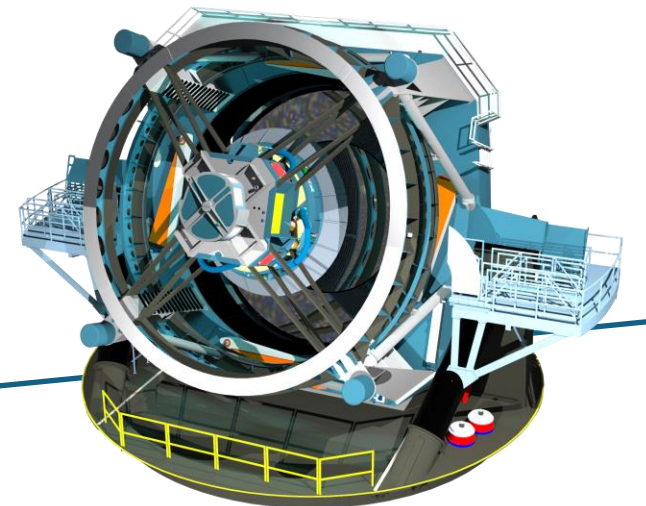


Remote Merges – Pull Requests

The output of pushing a branch to remote will be either:

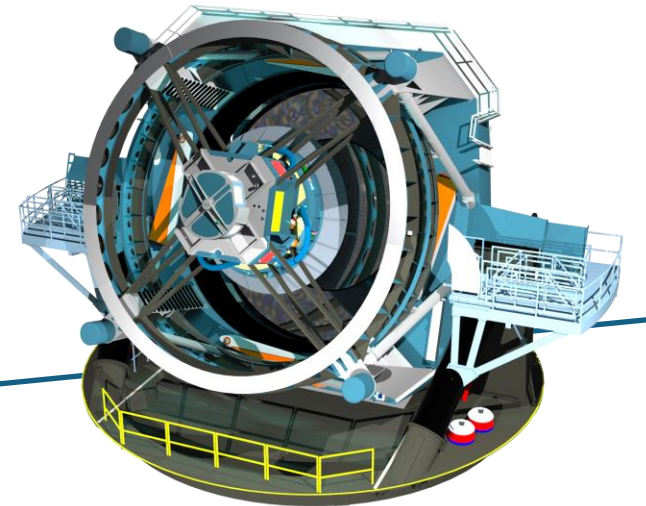
1. An automatic merge – if there are no conflicts.
2. Opening of a pull request, where you will manually resolve any conflicts.

Conflicts can be resolved either through the github GUI or by editing locally and pushing again.



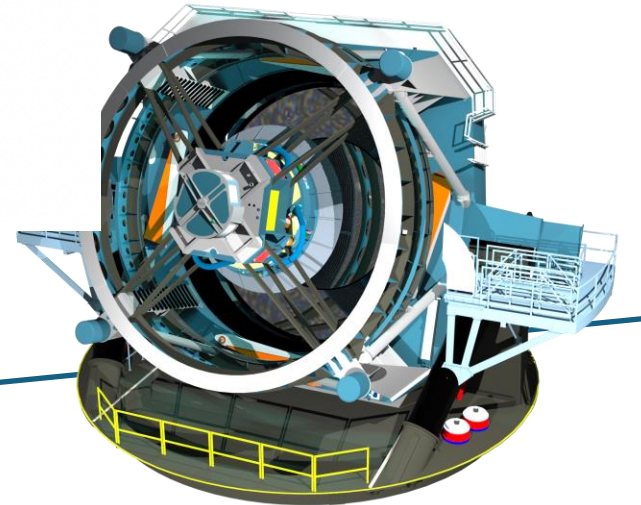
A note on jargon

The terminology here is a little weird and confused me for a long time. Push and pull are just opposite actions based on my perspective – am I copying into my branch or copying out of my branch. Remember: git is distributed, so relationships between repos are symmetric.



Github pull requests

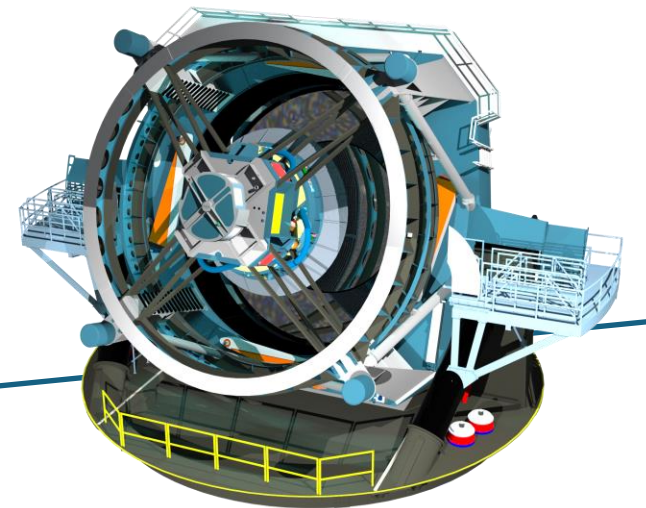
```
[(base) bryan@Bryans-MacBook-Pro Session21-github % git checkout new_feature
Switched to branch 'new_feature'
[(base) bryan@Bryans-MacBook-Pro Session21-github % vim readme.md
[(base) bryan@Bryans-MacBook-Pro Session21-github % git commit -a
[new_feature c29ac90] New merge conflict text
 1 file changed, 1 insertion(+), 1 deletion(-)
[(base) bryan@Bryans-MacBook-Pro Session21-github % git push origin new_feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 384 bytes | 384.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'new_feature' on GitHub by visiting:
remote:   https://github.com/bscot/Session21-github-lesson/pull/new/new_feature
remote:
To https://github.com/bscot/Session21-github-lesson.git
 * [new branch]      new_feature -> new_feature _
```



Pulling branches from remote

```
git pull origin main
```

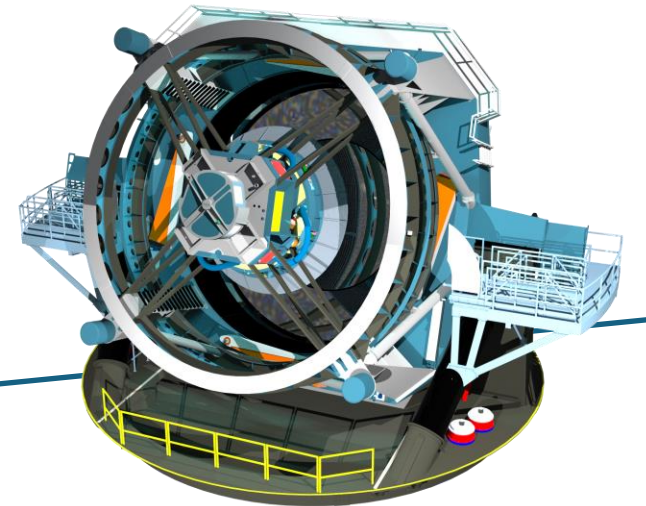
This works in reverse too. If I want to copy changes in the remote repo to my local repo, I git pull.



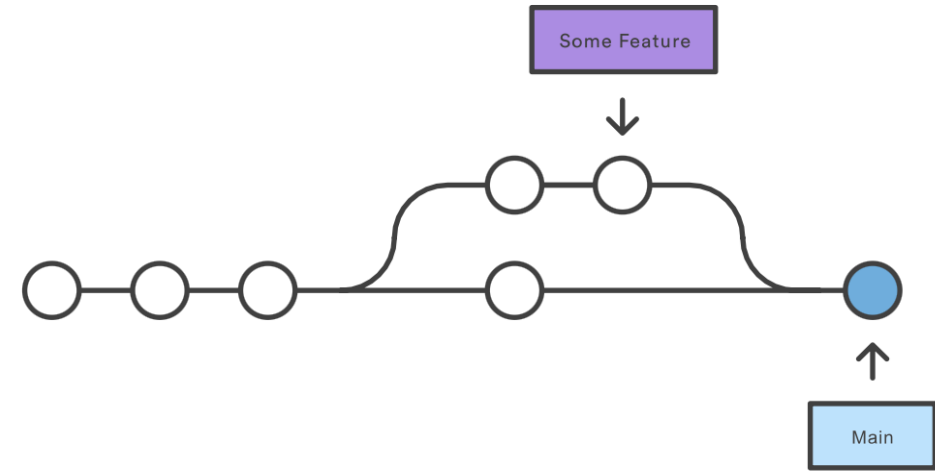
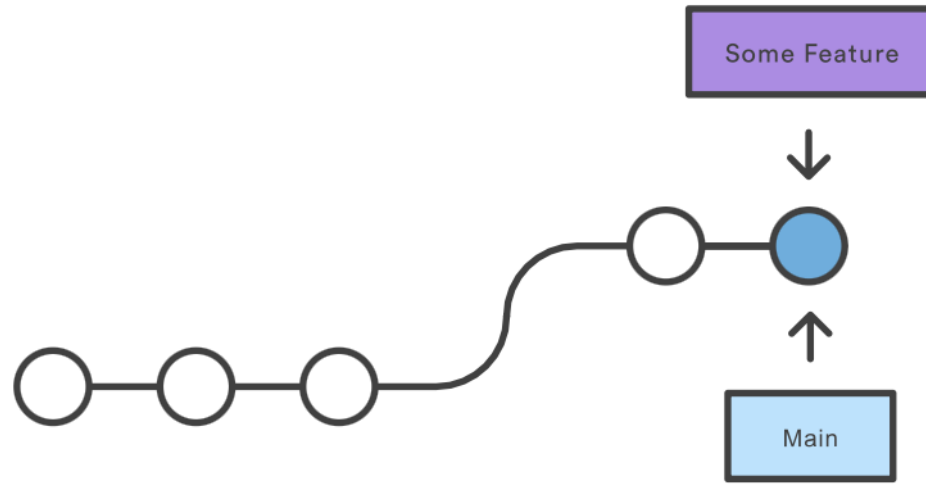
Types of merges

A fast-forward merge brings the local repo up to date with changes made in the remote repo. This kind of merge occurs if the local and remote repos are on the same "timeline", that is, if the remote repo is the child of the local repo.

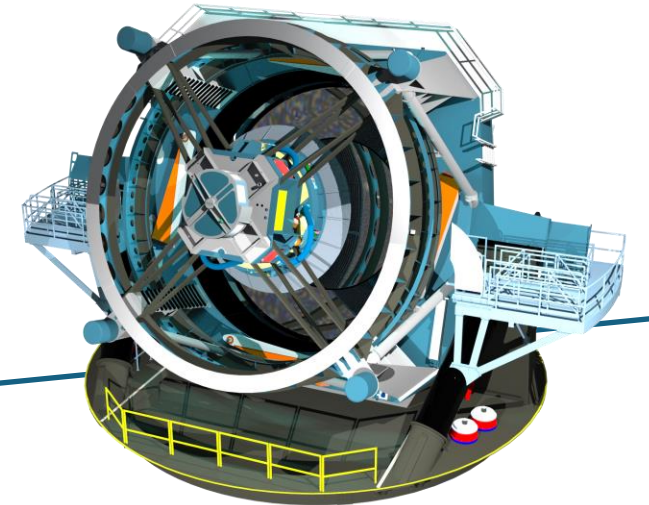
A 3-way merge occurs when there is no simple parent child relationship that can be used to generate the merge. This happens if we change the same file across different branches. Git handles this with a two-step commit.



Types of merges



(graphics from Atlassian git tutorial)



Merge Conflicts


In a 3 way merge, git first performs a *fetch* action where the branch to be merged is copied into the repository we're working in. A git reference called FETCH_HEAD is created that points to this copy.



Git then attempts to make the merge, which produces a conflict. The repository is now in the merging state, where we need to resolve the conflicts to complete the merge.


We manually fix the conflicts and commit the changes.






Merge Conflicts on Github


 **Session21-github-lesson** Public


 Pin  Unwatch 1


 **new_feature** had recent pushes 1 minute ago [Compare & pull request](#)

 main 2 Branches 0 Tags

 Go to file  Add file [Code](#)

 **bscot** setting up a merge conflict


 readme.md setting up a merge conflict





 **README**


This is a new repository for a git lesson (and its super awesome)

I can even add some text to a version of this file in a new branch

New merge conflict text #1


 Open **bscot** wants to merge 1 commit into **main** from **new_feature**


 Conversation 0  Commits 1  Checks 0  Files changed 1 +1 -1




Changes from all commits File filter Conversations Jump to 

[Review in codespace](#) [Review changes](#)

New merge conflict text

 **new_feature** (#1)

 **bscot** committed 4 minutes ago commit c29ac900ad5148047eea6146f10b331362840872

 2  readme.md 

@@ -1,3 +1,3 @@

1 - This is a new repository for a git lesson (and its super awesome)

1 + This is a new repository for a git lesson (its still cool even with merge conflicts)

2 2

3 3 I can even add some text to a version of this file in a new branch

Putting this into practice

When you review and merge
your own Pull Request in
your own repository

