

What do you think?



Work with a partner(s):
Given:

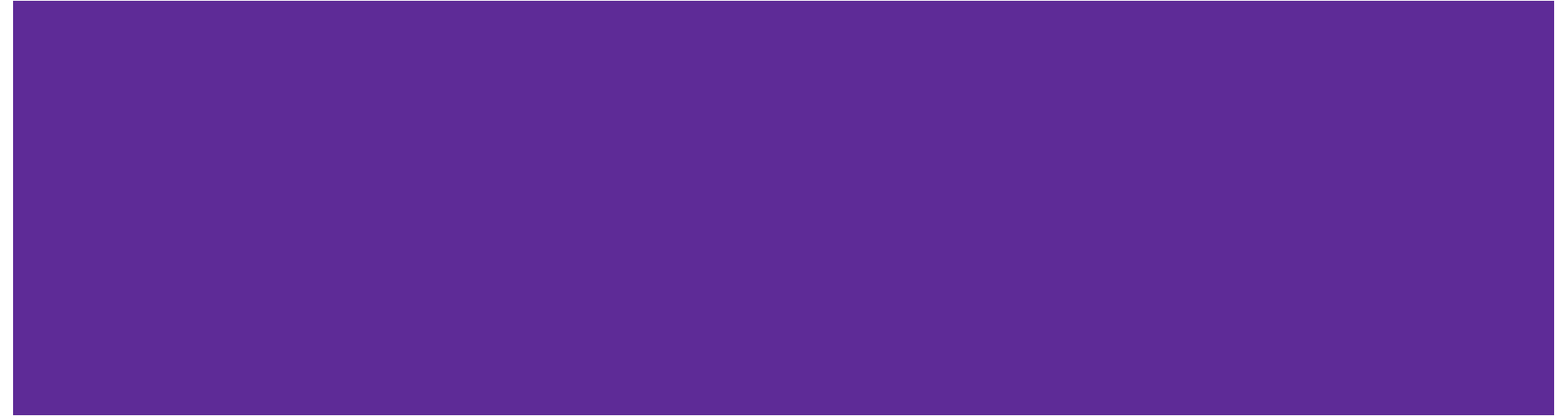
**What do these
commands do?**

1. first line
2. second line
3. third line
4. fourth line
5. fifth line
6. sixth line
7. seventh line
8. eighth line
9. ninth line
10. tenth line

```
sed '3y_0123456789_ABCDEFGHIJ_' sedaddresses  
sed '3~2y/0123456789/ABCDEFGHIJ/' sedaddresses  
sed '1,3y/0123456789/ABCDEFGHIJ/' sedaddresses  
sed '4,$y/0123456789/ABCDEFGHIJ/' sedaddresses  
sed '/f.*th/y/0123456789/ABCDEFGHIJ/' sedaddresses
```

CSE 374 Lecture GIT

Version control and Git



Review: Regex

```
/[a-zA-Z_\-]+@((([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4})/
```

regular expression ("regex"): describes a pattern of text

- can test whether a string matches the expr's pattern
- can use a regex to search/replace characters in a string
- very powerful, but tough to read

regular expressions occur in many places:

- text editors (Sublime, Vim, etc.): allow regexes in search/replace
- languages: JavaScript; Java Scanner, String split
- Unix/Linux/Mac shell commands (`grep`, `sed`, `find`, etc.)
- The site [regexr](#) is useful for testing a regex

What is version control?

Subversion, perforce, mercurial,
cvs, sourcesafe, **git**

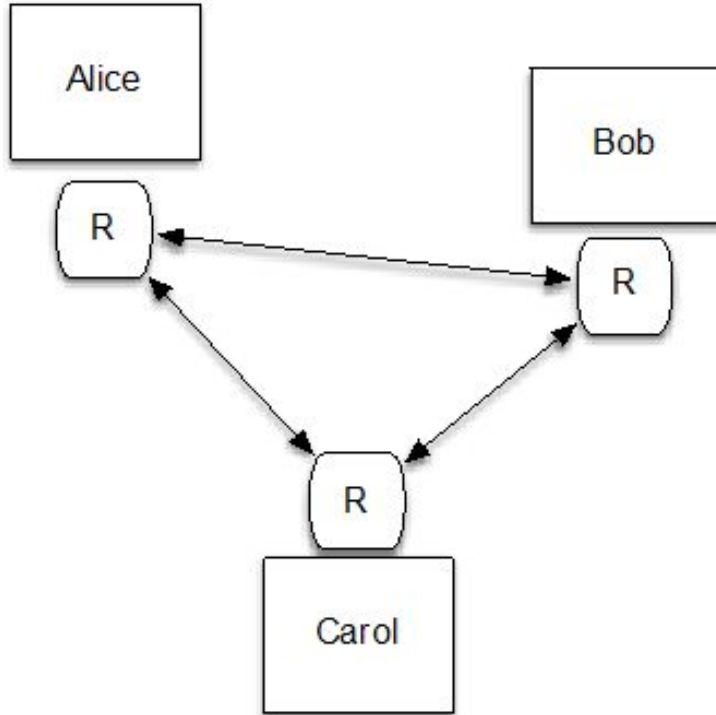
Software system that keeps records of files, changes-to-files, and manages sharing them between collaborators.

Why is version control?

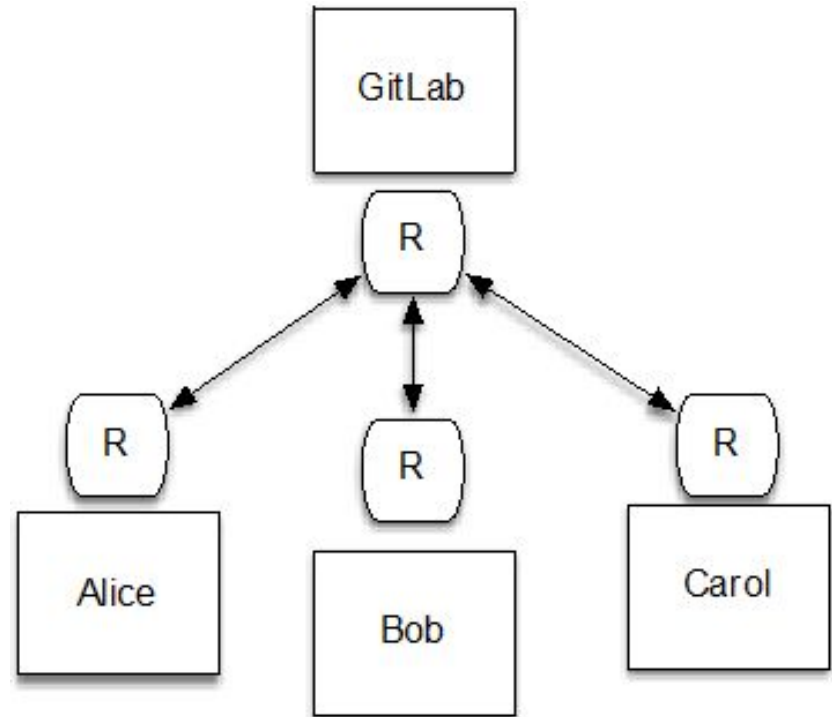
1. **Backups.** Archives a project to keep a safe copy.
2. **Collaboration.** Keeps a shared copy of project that all collaborators can access and update. Manages concurrent and maybe conflicting changes.
3. **Version log.** Keeps copies of previous versions so collaborators can revert if necessary.

Notes: Not language or coding specific; version control is used for all types of documents.

Alternate Models



Distributed System



Centralized System

Linus Torvalds

The creator of both Linux and Git!

- Linus → Linux

Git was originally created in 2005 for the sole purpose of continuing development of Linux.

- Linux was originally written in 1991.
- The old version control system revoked its free license for Linux

Git is the most popular version control system, with approximately ~95% of developers using it.



Repository

A **repository**, commonly referred to as a **repo** is a location that stores a copy of all files.

Synonymous to the root directory in a file system (i.e. '/').



A screenshot of a web browser showing a GitHub repository page. The browser's address bar displays 'https://github.com/uu-os-2018/example-repository'. The page header includes navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The repository name 'uu-os-2018 / example-repository' is shown with statistics: 1 Unwatch, 0 Stars, and 0 Forks. Below the header, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The main content area shows 'A small example GitHub repository' with an 'Edit' button. It lists '1 commit', '1 branch', '0 releases', and '1 contributor'. A 'Branch: master' dropdown and a 'New pull request' button are visible. A list of files is shown, including 'sub', 'README.md', 'one.txt', and 'two.txt', all marked as 'Initial commit' and '13 minutes ago'. The 'README.md' file is expanded, showing the text: 'Example GitHub repository' and 'A small example GitHub repository with a few files and one sub directory.'

Repository Do's and Don'ts

What is stored inside of a repository?

- Source code files (i.e. .c files, .java files, etc)
- Build files (Makefiles, build.xml)
- Images, general resources files

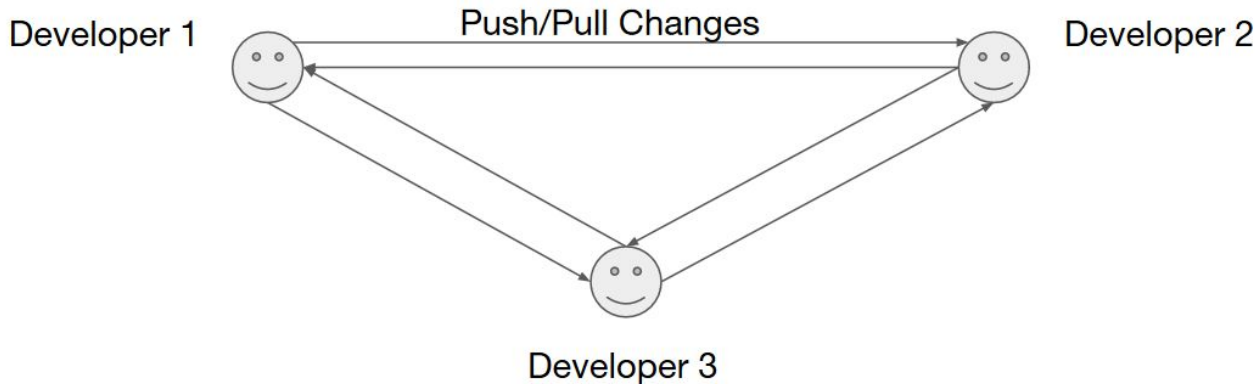
What should NOT be stored inside of a repository (generally)

- Object files (i.e. .class files, .o files)
- Executable binary files (executable scripts are OK!)

More on these types of files when we start with C next week!

Sharing Changes

- With git, everyone working on the project has a copy of the repository and its history
 - Everyone has a local copy of the repository, which is what we use to make our own changes.
 - We share changes by *pushing* and *pulling*



Central Git Repository

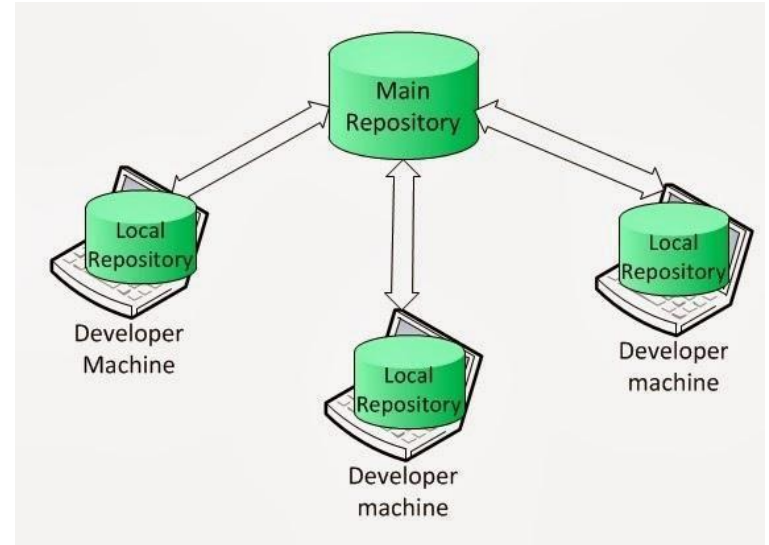
Keep an "origin" copy of the repo on a Git server

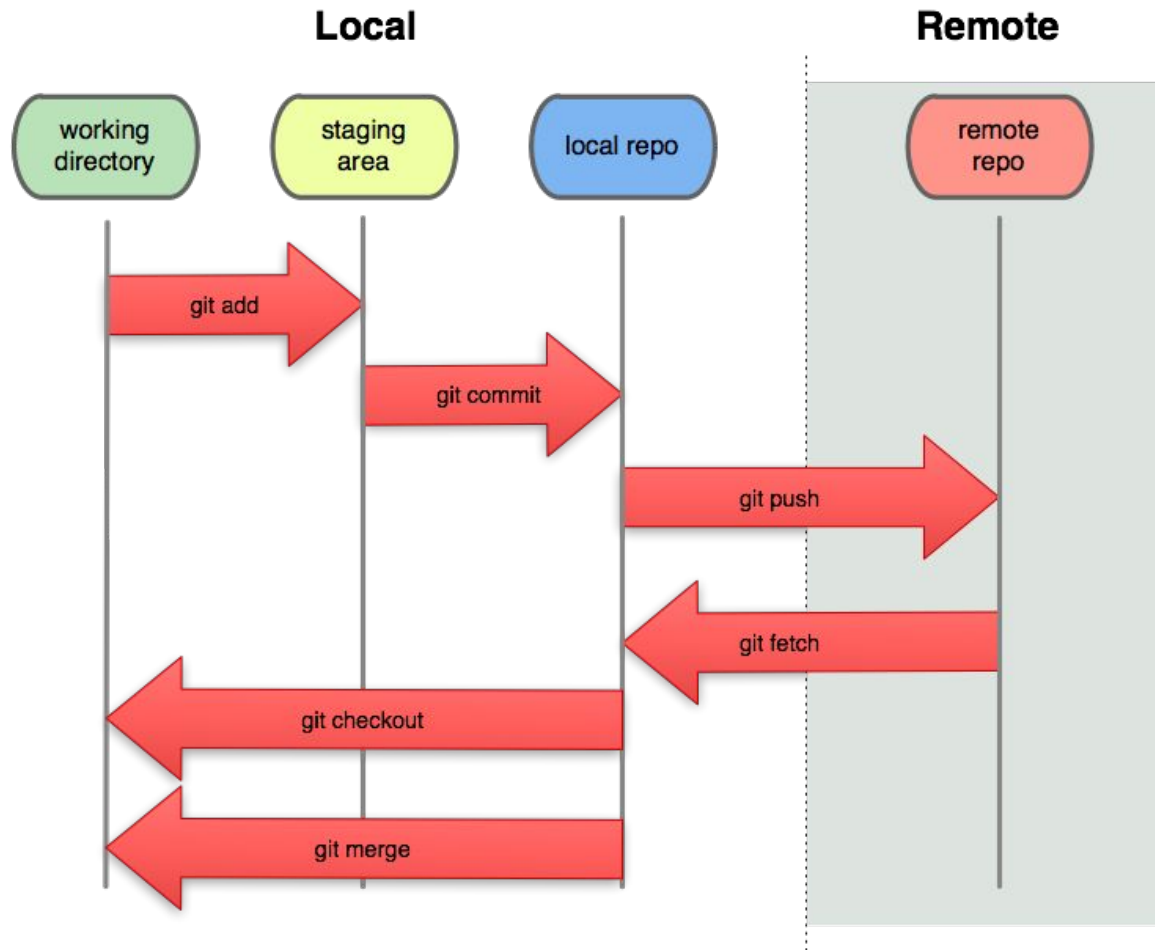
- The remote repository is the *defacto* central repository

Each user **clone** the repo to create a local copy

A user make changes, **add** and **commit** those to their copy and **push** to save them in the remote repository

All users **pull** from the central server periodically to get changes (instead of from each other).





Git Commands

git init, git clone, git add, git commit,
git pull, git push, git status, ...

The “git” Command

- The `git` command is the primary way of interacting with git
- You must `cd` into the folder where your repo is stored, or any subfolder within it
- Used like any other shell command!

```
[amckinn@calgary cse374]$ cd cse374-24wi-amckinn/  
[amckinn@calgary cse374-24wi-amckinn]$ git status  
On branch main  
Your branch is up to date with 'origin/main'.  
  
nothing to commit, working tree clean
```

Git Commands

Command

Operation

<code>git clone <i>url</i> [<i>dir</i>]</code>	Copy a Git repository so you can add to it
<code>git add <i>file</i></code>	Adds file contents to the staging area
<code>git commit</code>	Records a snapshot of the staging area
<code>git status</code>	View the status of your files in the working directory and staging area
<code>git diff</code>	Shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	Get help info about a particular command
<code>git pull</code>	Fetch from a remote repo and try to merge into the current branch
<code>git push</code>	Upload your local commits to the remote repository/server

Creating a Git Repo

Two common scenarios (only do one of these):

1. To create a new local Git repo in your current directory:

```
git init
```

- This will create a .git directory in your current directory.
- Then you can commit files in that directory into the repo.

2. To clone a remote repo to your current directory:

```
git clone url [localDirectoryName]
```

- This will create the given local directory, containing a working copy of the files from the repo, and a .git directory.

What is a "commit"?

- A **commit** is a **single set of changes** made to your repository
- Also records:
 - The name of the author
 - The date and time
 - A **commit message**: short sentence describing what that commit did
- Identified by an ID, or "SHA"

```
commit 88267581cfbed040b0f7a8679191879cf3bebeb5
Author: Alex McKinney <alexmckinney01@gmail.com>
Date:   Tue Dec 26 09:36:50 2023 -0800

    ci: Add find_packages to setup.py
```

Commit Messages

- Commit messages are the way you remind yourself and tell others what you did
- Commit messages should be **descriptive**
 - E.g. "Added test for predicting null string"
 - *not* "changed test"
- Commit messages should be **short/medium length**
 - If you want to know *exactly* what code was changed, you can check the full changes.

Commit History

- A repository's history is a series of "commits"
- Each commit makes changes to the files in the repo
- *Commit history* serves as a log of the changes everyone made
- `git log` to view the commit history

Commit early and often!

```
commit 862a4ef5666d98481aa7c1989d96c7bd20938198
Author: Alex McKinney <alexmckinney01@gmail.com>
Date: Thu Jan 18 15:00:36 2024 -0800
```

Add lecture 7 slides

```
commit 7bd2df0dfca000850cb036169003b858fe1d66a0
Author: Alex McKinney <alexmckinney01@gmail.com>
Date: Thu Jan 18 11:47:52 2024 -0800
```

Add lec07.md code

```
commit 143841fc143b4aa3aa9588b32c520f9ea0668dc1
Author: Alex McKinney <alexmckinney01@gmail.com>
Date: Wed Jan 17 09:02:31 2024 -0800
```

Add exercise links

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJ\$LKDFJ\$DKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Add and commit a file

The first time we ask a file to be tracked, and every time *before* we commit a file, we must **add** it to the staging area:

```
git add path/to/file.txt
```

- Takes a snapshot of these files, adds them to the staging area so it will be part of the next commit.
- Example: `git add hello.c goodbye.c`
 - Adds / stages all of the files in the current directory: `git add .`

To move staged changes into the local repo, we commit:

```
git commit -m "<message>"
```

Viewing changes

To view status of files in working directory and staging area:

`git status` or `git status -s` (short version)

- Lists the files which you have changed but not yet committed
- Indicates how many commits have made but not yet pushed

To see what is modified but unstaged: `git diff`

To see a list of staged changes: `git diff --cached`

To see a log of all changes in your local repo: `git log` or `git log --oneline` (shorter version). Press q to exit.

File *mv* or *rename*

- Once files have been committed to gitlab repository:

```
git mv files
```

```
git rm files
```

- git will make changes locally then update the remote GitLab repo when you push

~ If you use regular shell mv/rm commands, git will give you all sorts of interesting messages when you run git status and you will have to clean up

Gitlab remote use: sharing changes

- Good practice – update with remote changes:

```
git pull
```

- Also do this any time you want to merge changes pushed by your partner
- Test, make any needed changes, do git add / git commit to get everything cleaned up locally
- When ready, push accumulated changes to server

```
git push
```
- If push blocks because there are newer changes on server, do a git pull, accept any merge messages, cleanup, add/commit/push again

IN CASE OF FIRE



git commit



git push





exit building


IN CASE OF FIRE



 git commit

 git push

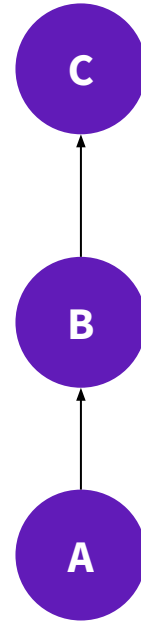
 exit building

 git pull

Collaborating

Collaboration: Ideal

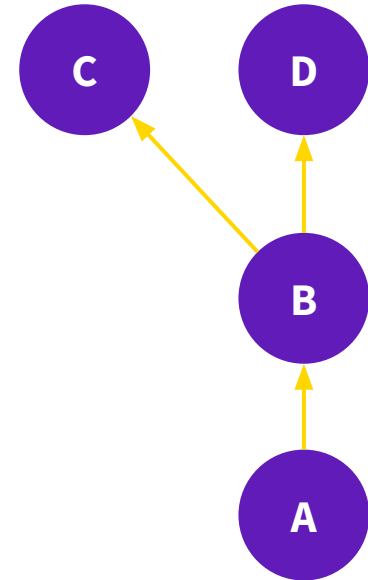
- A "linear" history
 - **Alice** makes a commit and **pushes**
 - **Bob pulls**, makes a change, commits the change, and **pushes**
 - **Alice pulls**, makes a change, commits, and **pushes**
 - ...etc.



Collaboration: Reality

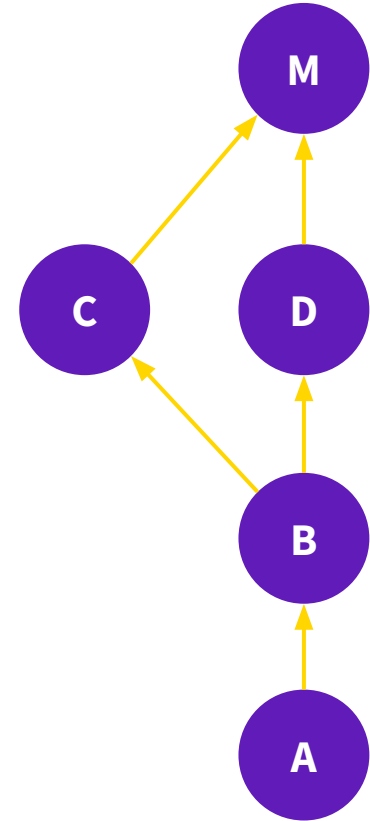


- We said the "commit history" is a list of commits, so what happens here?
 - **Charlie** makes a change and creates commit C, but **doesn't push**
 - **Diane** also makes a change and commit D, and **pushes**
 - **Charlie pulls** from the remote repo
 - It's no longer a list! The history has **diverged**
- Does Charlie just have to delete, pull and start over?



Merging

- A **merge commit** is a commit which has two "parents"
 - Combines the changes in each
 - Commit "M" includes all of Diane's changes, *plus* all of Charlie's



How do we merge?

`git pull`

- Automatically fetches the changes and merges them into yours
- Then, `git push`
 - This push your local changes to the remote repo
 - Others can now work off of your combined changes

Sometimes, the changes you make will ***conflict*** with the changes others make

- e.g. you both edit the same line
- Resolving merge conflicts is more complicated; we will teach the basics here but it takes a lot of practice - come to OH or post on Ed!

Merge conflicts

Git will tell you which files had merge conflicts (use `git status` to see conflicts), and the files will be edited to identify the conflict:

```
<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=====
<div id="footer">
  thanks for visiting our site
</div>
>>>>>> SpecialBranch:index.html
```

} branch 1's version

} branch 2's version

Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct). You must modify the section to contain the code you want, then save, ***add***, and ***commit*** the merge.

Fixing Mistakes

- Set local repository to the last commit (forget all changes that you've made), you can run `git reset --hard HEAD`
- Here "HEAD" refers to the most recent commit.
- If one of your past commits was BAD, you can undo it using
`git revert`
- If the second-to-last commit was bad, you can undo it by saying
`git revert HEAD~1`
 - a. HEAD is the most recent commit and "1" signifies the one before it. This will create a NEW commit that is the opposite of the original commit.
- Commits aren't completely static and permanent. If you make a commit but then realize you forgot one little thing, you can "amend"/modify your previous commit
`git commit --amend`

.gitignore

Git may be used to store any types of files.

However...

Do not store files that are unnecessary.

- Backup files (like *.swp vim files)
- Files that can be recreated (such as .o files) should not be added.
- System specific files

‘.gitignore’ lists files not to upload to HEAD. Below is a sample .gitignore file content:

```
# Ignore vim temporary files
*.swp

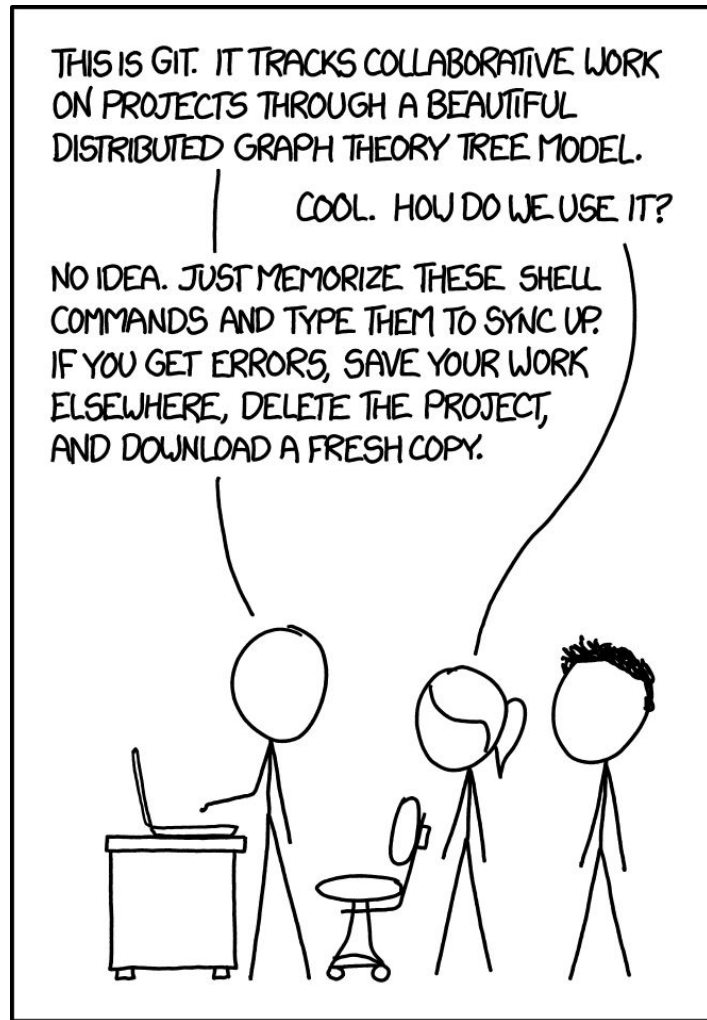
# Ignore OS X finder info
files
.DS_Store

# Ignore built object files
*.o
```

How do I fix git?!

You will inevitably run into frustrating situations with `git`

- Even for experienced users, sometimes you may accidentally get your git repo into an undesirable situation
- There is always a way to fix this, although it's not always obvious how
- Online resources are helpful
 - <https://ohshitgit.com/>



Learning more

- Lots of interesting and useful topics, including:
 - Branching, checkout
 - Resolving merge conflicts, merge tools
 - "Merge requests" (a.k.a. "Pull requests")
 - Rebase
- The web is your friend!
 - Official documentation
 - "Git Book": <https://git-scm.com/book/en/v2>

P.S. git is complex!

Three of the top four most-upvoted questions on StackOverflow.

Everyone is learning!

24,044,301 questions Newest Active Bountied 171 Unanswered More Filter

27187 votes **Why is processing a sorted array faster than processing an unsorted array?**
✓ 25 answers
1.9m views

`#include <algorithm> #include <ctime> #include <iostream> int main() { // ...`

`java` `c++` `performance` `cpu-architecture` `branch-prediction`

GManNickG 497k asked Jun 27, 2012 at 13:51

26141 votes **How do I undo the most recent local commits in Git?**
✓ 105 answers
13.6m views

I accidentally committed the wrong files to Git, but didn't push the commit to the server yet. How do I undo those commits from the local repository?

`git` `version-control` `git-commit` `undo`

Community wiki **93 revs, 64 users 11% Peter Mortensen**

20388 votes **How do I delete a Git branch locally and remotely?**
✓ 41 answers
11.3m views

Failed Attempts to Delete a Remote Branch: \$ git branch -d remotes/origin/bugfix error: branch 'remotes/origin/bugfix' not found. \$ git branch -d origin/bugfix error: branch 'origin/bugfix' not...

`git` `version-control` `git-branch` `git-push` `git-remote` Community wiki **Matthew Rankin**

13761 votes **What is the difference between 'git pull' and 'git fetch'?**
✓ 37 answers
3.4m views

What are the differences between git pull and git fetch?

`git` `version-control` `git-pull` `git-fetch` **Pablo Fernandez 282k** asked Nov 15, 2008 at 9:51

GitLab

CSE Gitlab

- Github and Gitlab are just websites that store git repos
- You can create a repo on the website and `git clone` to edit it on your computer (e.g. laptop, calgary, etc.)
- CSE has its own version of Gitlab where you will be given a repository
 - <https://gitlab.cs.washington.edu/cse374-24au-students>
 - We'll use this to distribute and submit homework assignments

374: Gitlab

Resources on line -

<https://gitlab.cs.washington.edu/help>

<https://courses.cs.washington.edu/courses/cse374/24au/resources/git.html>

<https://git-scm.com/book/en/v2>

<https://about.gitlab.com/images/press/git-cheat-sheet.pdf>

Subsequent assignments will ask you to use gitlab, which provides starter code and allows you to upload your code and submit it to gradescope.

Don't store things like .o files and executable programs that don't belong in a repository.

You must use the provided repository even if you have separate machines or accounts of your own that you use for other projects.

You should regularly commit and push changes you are making on code bases. The more often you do this, the less work you lose if things go awry.

Bonus Slides



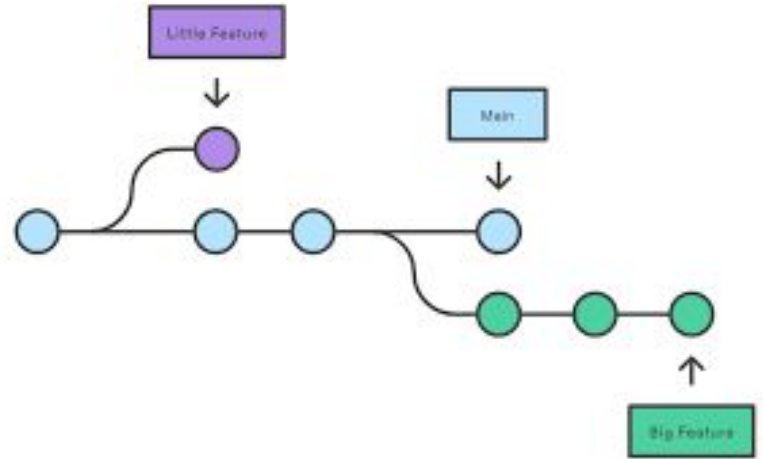
Branch

Git supports **branches**, which are used to split out a line of work.

- A branch is composed of one or more **commits**.
- A repository contains one or more branches.

The **main** branch is the primary source of truth!

- Default when a repository is created.



repository > branch > commit

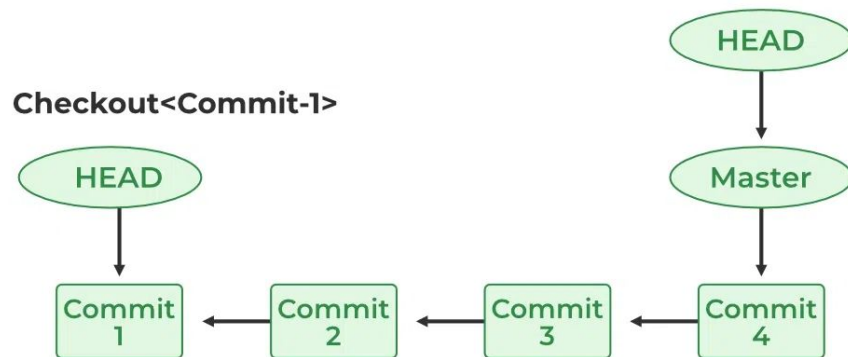
HEAD Commit

You will often see references to **HEAD**, which is the **latest commit**.

- Acts like a bookmark
- Every branch has its own **HEAD**

Display the content of the latest commit:

```
git show HEAD
```



[Link](#)

What is Git?

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.

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..."; and eventually you'll learn the commands that will fix everything.



Repository Access

A repository can be:

- Local: run git commands in repo directory or subdirectory
- Remote: lots of remote protocols supported (ssh, https) depending on repository configuration
 - Specify user-id and machine
 - Usually need git and ssh installed locally
 - Need authentication (use ssh key with GitLab)
- cse374 uses ssh access to remote GitLab server
- Feel free to experiment with GitLab



CSE374-22WI

Project ID: 73031

3 Commits 1 Branch 0 Tags 113 KB Files 113

Shared project for the 22 WI quarter

```
emacc@localhost:localdomain
└─$ ssh mh75@localhost -i /cse374-22wi

File Edit View Search Terminal Help
[mh75@localhost ~]$ git clone git@gitlab.cs.washington.edu:mh75/cse374-22wi.git
Cloning into 'cse374-22wi'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (2/2), done.
[mh75@localhost ~]$ git pull
fatal: not a git repository (or any parent up to mount point /)
```



Auto DevOps

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Learn more in the [Auto DevOps documentation](#)

Enable in settings

main

cse374-22wi /



History

Find file

Web IDE



Clone



Merge branch 'mh75-main-patch-35196' into 'main'

Megan Hazen authored 18 hours ago

Upload File

README

Add LICENSE

Add CHANGELOG

Add CONTRIB

Set up CI/CD

Configure Integrations

Clone with SSH

`git@gitlab.cs.washington.edu:mh75/`

Clone with HTTPS

`https://gitlab.cs.washington.edu/m`

Open in your IDE

Visual Studio Code (SSH)

Name

Last commit

Local Additions & Editing

- Edit a file “`stuff.c`”
- Add file(s) to list to be saved in repo on next commit

```
git add stuff.c
```

- Commit all added changes

```
git commit -m "reason/summary for commit"
```

- Repeat locally until you want to push accumulated commits to GitLab server to share with partner or for backup...

Git commit -m 'messages should be useful'

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJ\$LKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

Example Commands

- Update local copy to remote

```
git pull
```

- Make changes

```
git add file.c
```

```
git mv oldfile.c newfile.c
```

```
git rm obsolete.c
```

- Commit changes to local repo

```
git commit -m "fixed segfault in getmem"
```

Examine changes

`git status` (see uncommitted changed files, or how to revert changes, etc.)

`git diff file` (see uncommitted changes in *file*)

`git log` (see history of commits)

- Update GitLab shared repo to reflect local changes

```
git push
```

.gitignore

Git may be used to store any types of files.

HOWEVER

Do not store files that are unnecessary.

- Backup files (like *~ emacs backups)
- Files that can be recreated (such as .o files) should not be added.
- System specific files

‘.gitignore’ lists files not to upload to HEAD

```
# emacs backup files
```

```
*~
```

```
# OS X finder info files
```

```
.DS_Store
```

```
# built object files
```

```
*.o
```