Lecture 19: Git!
QUICK RECAP
Some “git” commands

- **git init**
  - Create a new empty git repo or convert an existing folder to a git repo

- **git add**
  - Preparing edited files to be saved (committed) to a repo

- **git commit**
  - Records (saves) changes to a repo
  - Accompanied by a short descriptive message

- **git push**
  - Update the remote copy of the repo with the local changes and commits
Staging and committing overview

- Working changes
  - `git add .`
- Staging area
  - `git commit -m "message"`
- git init
- git repo
Inspecting a repository

- **git status**
  - Lists the files which you have changed but not yet committed
  - Working directory
  - Staging area
  - Indicates how many commits have made but not yet pushed

- **git log**
  - Shows the commit history
  - `git log --graph --oneline`
  - Shows branch info as a graph
Working with remote
git commands for interaction with remote

- **git clone**
  - Cloning is the process of creating a working copy of the remote or local repository by passing the following command.
  - `git clone username@git_server_hostname:/path_of_repository`

- **git pull**
  - If we have already cloned the repository and need to update local (only code) respect to the remote server
  - `git pull origin main`

- **git fetch**
  - Fetching is the process of updating (only git information) the local git structure and information from remote repository
  - `git fetch`
Updating changes using pull

Working Directory: git add ...
Staging Area/Index: git commit
Local Repository: git push
Remote Repository: git pull

NOTE: There are way more git commands than what is listed here - this is a simplified model to get us started.
What’s next!

▪ Branching, checkout

▪ Merging, Merge (Pull) Requests

▪ Conflicts

▪ Interacting with a Git Server (GitHub / GitLab)
Skipping files using `.gitignore`

- As a recap, generally we should not have these files in a git repo:
  - Object files (i.e. `.class` files, `.o` files) and executables
  - Huge media files (e.g. videos)
  - Credentials and system files (e.g. `.DS_Store` in Mac)

- Its tedious to mention all the files you want to add into a repo every time you add your changes (and ensure you skip the others)

- To skip above files you can create a file in the root of the git repo folder called `.gitignore`

- This file should contain patterns in file names you would like to skip being added

- [https://github.com/github/gitignore](https://github.com/github/gitignore)
Working with a Git server
GitHub profile

Kushal Jhunjhunwalla

Pinned
- BlockChain-In-HealthCare
  - Python
- CameraApp
  - Java

112 contributions in the last year

Contribution activity

Follow
1 follower
0 following

Highlights
- Arctic Code Vault Contributor
Creating a new repo on GitHub

[Image of GitHub interface with 'New' button highlighted]
Creating a new repo on GitHub

![Create a new repository](https://github.com/new)

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

**Owner** 🔄 **Repository name** 🔄

Great repository names are short and memorable. Need inspiration? How about redesigned-octo-spark?

**Description (optional)**

HW03 Repo for CSE 374

**Public**

Anyone on the internet can see this repository. You choose who can commit.

**Private**

You choose who can see and commit to the repository.

**Initialize this repository with**

Skip this step if you're importing an existing repository.

- Add a README file
  
  This is where you can write a long description for your project. Learn more.

- Add gitignore
  
  Choose which files not to track from a list of templates. Learn more.

- Choose a licence
  
  A licence tells others what they can and can’t do with your code. Learn more.

Create repository
Creating a new repo on GitHub

Quick setup — if you’ve done this kind of thing before

- Set up in Desktop
- HTTPS

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

- echo "# cse374_hw3" >> README.md
- git init
- git add README.md
- git commit -m "First commit"
- git branch -M main
- git remote add origin git@github.com:kushalj/cse374_hw3.git
- git push -u origin main

...or push an existing repository from the command line

- git remote add origin git@github.com:kushalj/cse374_hw3.git
- git branch -M main
- git push -u origin main

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code
Adding local files to the GitHub server

```
$ git status
fatal: not a git repository (or any parent up to mount point /mnt)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).

$ git init
Initialized empty Git repository in /mnt/c/Users/kusha/ta/cse374_hw3/.git/

$ git add .
$ git commit -m "Add starter files"
[master (root-commit) 4b018c4] Add starter files
  5 files changed, 3353 insertions(+)
```

Username for 'https://github.com': kushaljh
Password for 'https://kushaljh@github.com':

Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (77/77), 40.68 KiB | 2.91 MiB/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/kushaljh/cse374_hw3.git
  * [new branch] master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```
Checking the changes on GitHub
Branching
Collaboration – the ideal case

- 3 Collaborators
- Kasey makes a change and updates the repo (git push)
- Kalyani gets Kasey’s changes (git pull), edits some files and updates the repo (git push)
- Kushal gets the latest changes that include both Kasey and Kalyani’s changes (git pull)
- This goes on…
But ...
Collaboration – the reality

- 3 Collaborators

- Kasey makes a change and updates the repo

- Kalyani and Kushal get Kasey’s changes, edit some files and Kalyani updates the repo while Kushal is still working on some edits

- And here you see different people having different version histories
Error!!

kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$ git push
Username for 'https://github.com': kushaljh
Password for 'https://kushaljh@github.com':
To https://github.com/kushaljh/git_practice.git
  ! [rejected] main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/kushaljh/git_practice.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$ |
Error!!

```
kushaljh@LAPTOP-UA25NDJJ:/mnt/d/kusha/OneDrive - UW/Documents/GitHub/git_practice$: git push
Username for 'https://github.com': kushaljh
Password for 'https://kushaljh@github.com':
To https://github.com/kushaljh/git_practice.git

! [rejected]   main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/kushaljh/git_practice.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```
ONE DOES NOT SIMPLY
PUSH TO MASTER
Git Branches

- The scenario mentioned in the previous slides would be tough to solve if we used a single history across all collaborators.
- Instead you can start your own history at any point by “branching” out of the main history for a repo.
- The master / main branch which is central history and hence the source of truth for the whole project.
- We create new version histories based on the main branch and give each of these a branch name.
Git Branches

- Kushal’s improved workflow using branches!

- Kushal creates a new branch called `dev-kushal` when she starts working
  
  - `git branch dev-kushal`
  - `git checkout dev-kushal`

  OR

  - `git checkout -b dev-kushal`

- Now she makes commits on this branch until she is ready to update master
Git Merge

- After Kushal is done with her work, she would like the changes from her branch `dev-kushal` be reflected in the `master` branch.

- Steps:
  - `git checkout master`
  - `git merge dev-Kushal`

- If there are no changes that are made in the same lines by Kalyani and Kushal there are no conflicting differences, and the merge is good to go.

- If not, **WE HAVE A MERGE CONFLICT!**

- Note: this is not the only workflow used for merging.
PULL before you start working!!

I see that you commit your code without doing a pull from Git to get the latest updates.

I also like to live dangerously.
Merge conflicts

- Merge conflicts happen when you are merging two branches that have a diff on the same line(s) of some file(s) in a repo

- When running merge we run into an error in such a scenario

```bash
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git merge feature
Auto-merging file.txt
CONFLICT (add/add): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git status
On branch master
Your branch is up to date with 'origin/master'.

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)
  both added: file.txt

no changes added to commit (use "git add" and/or "git commit -a")
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$
```

```bash
<<<<<<< HEAD
contents
=======
header
>>>>>>> feature

"file.txt" 5L, 53C
```
Looking for the conflict
Resolving the conflict
Checking the status

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
Unmerged paths:
  (use "git add <file>..." to mark resolution)
      both added: file.txt
no changes added to commit (use "git add" and/or "git commit -a")
```
Adding resolved file and saving

```
adding resolved file and saving

(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)

  both added:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git add file.txt
kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$ git status
On branch master
Your branch is up to date with 'origin/master'.

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:

  modified:   file.txt

kushaljh@LAPTOP-UA25NDJJ:/mnt/c/Users/kusha/ta/cse374_hw3$
```
Conflict resolved!!

```
$ git add file.txt
$ git status

On branch master
Your branch is up to date with 'origin/master'.

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:

  modified: file.txt

$ git commit -m "Complete merge"
[master ad8ee9b] Complete merge

$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```
Merge (Pull) Requests

- This was nice to know but we generally do not merge branches into master locally.
- We use a Git server – GitHub / GitLab – to create a request to merge a feature branch into master.
- We must ensure that everyone on the project agrees to what is present in the master branch as this is our source of truth that everyone shares.

The workflow is as follows:
- Create a new branch
- Add and commit changes to the branch and push it to GitHub
- Create a Pull Request on GitHub
- Other collaborators look at the PR and leave their feedback (this is generally called a Code Review)
- Fix issues and then merge
Sample Pull Request

<table>
<thead>
<tr>
<th>Line</th>
<th>Original Sentence</th>
<th>New Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>The file may be</td>
<td>The file may be</td>
</tr>
<tr>
<td>4</td>
<td>Copyrighted in</td>
<td>Copyrighted in</td>
</tr>
<tr>
<td>6</td>
<td>indicating that</td>
<td>indicating that</td>
</tr>
<tr>
<td>7</td>
<td>every year in the</td>
<td>every year in the</td>
</tr>
<tr>
<td>8</td>
<td>range, e.g.,</td>
<td>range, e.g.,</td>
</tr>
<tr>
<td>10</td>
<td>indicating that</td>
<td>indicating that</td>
</tr>
<tr>
<td>11</td>
<td>every year in the</td>
<td>every year in the</td>
</tr>
<tr>
<td>12</td>
<td>range, e.g.,</td>
<td>range, e.g.,</td>
</tr>
<tr>
<td>14</td>
<td>indicating that</td>
<td>indicating that</td>
</tr>
<tr>
<td>15</td>
<td>every year in the</td>
<td>every year in the</td>
</tr>
<tr>
<td>16</td>
<td>range, e.g.,</td>
<td>range, e.g.,</td>
</tr>
<tr>
<td>17</td>
<td>See</td>
<td>See</td>
</tr>
<tr>
<td>19</td>
<td>how to report bugs</td>
<td>how to report bugs</td>
</tr>
</tbody>
</table>
Working with branches & more
IN CASE OF FIRE

1. git commit
2. git push
3. git out!
Useful resources

- **Try Git (resources and tutorial)**

- **The Git Cheat Sheet**

- **Stack Overflow’s definitive guide for beginners**

- **When you are terribly stuck with git**
  - DO NOT PANIC! Even experienced developers get stuck with git issues
  - [https://ohshitgit.com/](https://ohshitgit.com/)
  - [https://stackoverflow.com/questions/tagged/git](https://stackoverflow.com/questions/tagged/git)