Messing Up On Git
Today's Lecture:

1. More Git Commands
   - Useful Git commands if you mess up

2. Reverting another team's mistakes
   - Demo of git reset and an introduction to git revert

3. Advanced Git
   - More useful Git commands
More Git Commands

Useful Git commands if you mess up
You can set up an alias for each command using git config.

For example:

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```
git alias

git config --global alias.lg1 "log --all --graph --decorate --oneline"

[ ~ ]
git lg1
* 9f0dbb2 (HEAD -> main) adds main
* 60e6e78 adds 5
| * 9224164 (feature) adds feat 2
| * 283e527 adds feat 1
|/
* 19e506b adds 2
* 7d58006 adds 1
Git merge vs git rebase

git checkout feature
git merge master
Git merge vs git rebase

git rebase
Git merge vs git rebase
[ ~ ]git log --all --graph --decorate --oneline

* 44cd78c (HEAD -> master) adds 7
* d1ad5e9 adds 6
  * 6cf4d13 (feature) adds 5
  * b7f520f adds 4
  * 755620b adds 3
  /
* adea689 adds 2
* 7b66d7c adds 1
git rebase

Git merge vs git rebase

Git checkout feature
Git rebase main

Git checkout main
Git rebase feature
git reset basics

- Allows user to modify their repository history
- Helps rollback to a specific commit
- Changes back to a specific commit in a brute-force kind of way that disrupts the commit history of a repository.
- Used on your local, private repositories, especially if the repository is shared by others
We have the following sequence of commits

- The above diagram is a linked list of commits
- Let’s say we made 4 commits so far, A,B,C,D
- As we can see our Master and Head pointer points to our latest commit D
git reset basics continued...

Moves both the head AND branch pointers to a specific commit and the commit history is modified.
Recall *git checkout*

Move ONLY the HEAD pointer to a specific commit and the commit history remains untouched.
**git reset --hard <hash>**

- Most dangerous type of reset
- Moves the head and master pointer to the target commit
- Staging area and working directory are changed to match the specific commit
- Files in the staging area prior to running this command are discarded
  - Can cause large amounts of data loss if used incorrectly
git reset --soft <hash>

- Moves the head and master pointer to the target commit
- Staging area and working directory are left untouched
  - This is generally the safest option

And if everything goes wrong:

```
git reset HEAD^ --hard
```  
```
git push -f master
```  

(to be repeated until it works again)

I can hear the cries...
git reset --mixed <hash>

- Meant to be a median between "--soft" and "--hard",
- The DEFAULT option if a mode for reset is not specified
- Moves the head and master pointer to the target commit
- Changes the staging area to match the specific commit
- Files in the current staging area moved back to your current working directory
Fixing a team’s mistake

Demo of git reset and an introduction to git revert
Consider the follow Repository’s Commit History

commit cc692c48ab83425fef6aa91d0f6f3026b9ba6930 (HEAD -> main, origin/main)
Author: 
Date:   Sat Nov 7 14:46:46 2020 -0500

  Commit D

commit ad6ef2a7645daf7e66e210e3f16d1ff0a4094422
Author: 
Date:   Sat Nov 7 14:45:59 2020 -0500

  Commit C

commit 77eaeb4c66fd94d87eb1c39763a5b5687ad080
Author: 
Date:   Sat Nov 7 14:44:37 2020 -0500

  Commit B

commit e04a637ec5cd9a031324c16d772d0061e03b0279
Author: 
Date:   Sat Nov 7 14:41:05 2020 -0500

  Commit A

(END)
Consider the same Repository’s Staging Area and Working Directory

```bash
$ Test_Repo git:(main) x git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   test.txt
```

```bash
$ Test_Repo git:(main) cat test.txt
Commit D
```
**git reset --hard example**

Notice how the staging area is now empty because Commit B’s Staging area was empty.

```
➜ Test_Repo git:(main) × git reset --hard 77eaeb4c66fddf94d8f7eb1c39763a5b5687ad080
HEAD is now at 77eaeb4 Commit B
➜ Test_Repo git:(main) git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```
`git reset --hard` example continued...

Run `git log` to see how the list of commits has been modified.
Notice how the working directory files have been ‘reverted’ and now contain a different test.txt.
**`git reset --soft` example**

Notice how the staging area remains untouched

```bash
$ TestRepo git:(main) git reset --soft 77eaeb4c66fddf94d8f7eb1c39763a5b5687ad080
$ TestRepo git:(main) × git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:  test.txt
```
Notice how the log is the exact same as the log after we ran `git reset --hard`
Notice how the working directory file has been left untouched

[→ Test_Repo git:(main) × cat test.txt
Commit D}
Notice how the staging area is now empty because Commit B’s Staging area was empty.
**git reset --mixed** example continued...

Notice how the log is the exact same as before

<table>
<thead>
<tr>
<th>Commit</th>
<th>Hash</th>
<th>Author</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080</td>
<td>(HEAD -&gt; main)</td>
<td>Sat Nov 7 14:44:37 2020 -0500</td>
</tr>
<tr>
<td></td>
<td>e04a637ec5cd9a031324c163772d0061e03b0279</td>
<td></td>
<td>Sat Nov 7 14:41:05 2020 -0500</td>
</tr>
</tbody>
</table>

Commit B

Commit A
Notice how the modified file we added to the staging directory is now in our working directory.
Popular Usage of git reset:

- If ever, you add a file to the staging area but want to remove the file from staging, we run the following command: `git reset HEAD TARGET-FILE`

- If you ever want to abandon all local changes and start fresh with a copy of your remote repository, run `git reset --hard` and then `git pull`
Clicker Quiz

Which of the following commands only modify the commit history

a) git reset --hard
b) git reset --soft
c) git reset --mixed
d) git reset
Clicker Quiz

Which of the following commands only modify the commit history

a) git reset --hard
b) **git reset --soft**
c) git reset --mixed
d) git reset
git revert

- Used for undoing changes to a repository.
- Revert does not modify the repository history.
- Makes a new commit that that reverses any changes to achieve the state of the specified commit.
- Use this kind of version control on public branches instead.
Notice how the new head and master are essentially just a copy of the second commit
1. Consider the following situation on Test_Repo.
2. A team accidentally adds a file called random.txt
3. We want to revert the other team’s change in a safe manner
To revert the last commit, we copy the hash and use `git revert <hash>`

As we see below, we have reverted their addition of the file and can safely push these changes to the remote repository.
## When to use what?

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>• git revert</td>
<td>• git revert</td>
</tr>
<tr>
<td>• git reset</td>
<td>• git cherry-pick</td>
</tr>
<tr>
<td>• git cherry-pick</td>
<td>• git checkout</td>
</tr>
<tr>
<td>• git checkout</td>
<td></td>
</tr>
</tbody>
</table>
Fill in the blank:
“git revert is __________ , compared to git reset”

a) safer to use locally
b) brute force
c) safer to use remotely
d) more dangerous to use remotely
Clicker Quiz

Fill in the blank:
“git revert is _________ , compared to git reset”

a) safer to use locally
b) brute force
c) **safer to use remotely**
d) more dangerous to use remotely
Advanced Git

Advanced Git commands
More Git Commands

**git commit --amend**
Modifies your most recent commit by combining changes in your staging area with your previous commit.

**git reflog**
Lists the history of updates to ref pointers in your local repository.

**git clean**
Removes up untracked changes files in your repository. Keep in mind that the -n or -f flag is required.
More Git Commands

- `git ls-files -s`
  
  Can be used with the "--deleted", "--modified", or "--others AND --exclude-standard" flag to list the files of each type.

- `git reset --soft HEAD~N`
  
  Removes last N by moving the current HEAD to the specified commit.

- `git diff --cached`
  
  Shows specific changes in files that are currently in the staging area.
Clicker Quiz

Which of the following flags combine changes in your staging area with your previous commit?

a) --add
b) --readd
c) --revert
d) --prevamend
e) --amend
Which of the following flags combine changes in your staging area with your previous commit?

a)  --add  
b)  --readd  
c)  --revert  
d)  --prevamend  
e)  --amend
Git Hooks
Git Hooks

Git can trigger custom scripts that perform certain operations. These scripts are referred to as **hooks**.

[ ~ ] ls .git/hooks

- `pre-commit.sample`
- `applypatch-msg.sample`
- `commit-msg pre-push.sample`
- `fsmonitor-watchman.sample`
- **post-update.sample**
- `pre-applypatch.sample`
- `pre-merge-commit.sample`
- `commit-msg.sample pre-rebase.sample`
- `pre-receive.sample`
- `prepare-commit-msg.sample`
- `update.sample`
Creating a commit-msg Hook

[ ~ ] cd .git/hooks
[ ~ ] cp commit-msg.sample commit-msg
[ ~ ] chmod +x commit-msg
commit-msg Hook

#!/usr/bin/env ruby
message_file = ARGV[0]
message = File.read(message_file)
$format = /\[(\w+)\]/:
if !$format.match(message)
  puts "[POLICY] Your message is not formatted correctly"
  puts "[STANDARD] Your message should be in the format: ‘[module]: commit message’"
  exit 1
end
Test commit-msg Hook

[ ~ ]git commit -m 'test'
[POLICY] Your message is not formatted correctly
[STANDARD] Your message should be in the format: ‘[module]:
commit message’

[ ~ ]git commit -m ”[test]: testing tests”

[main 3457535] [test]: testing tests
1 file changed, 1 insertion(+)
