Module 7 Pre-Read - Git

Wednesday, June 5, 2024 01:51

Git init

```
git init
```

From the command line

Let's say you have a folder and a file inside

Navigate to the folder and then type into the command line `git init`

```
ls -a
```

to see all the files in the project directory

```
touch emptyfolder/.gitkeep
```

If you have an empty folder, git won't recognize it, so add this

Making commits

A "snapshot" of your repository at any moment

```
git status
```

Sees what untracked stuff you have

```
git add
git add --all
```

Stage the file for commit (not yet committed but ready to be committed)

```
git commit -m "put a message for ur commit"
```

Commit and also creates an alphanumeric for that commit

Viewing old commits

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

See the last few commits that you have made

```
git checkout 4637ff8
```

Look at how ur code was at this commit (and use the first 7 digits of the alphanumeric)

```
git checkout main
```

Go back to the current version

Git diffs

See differences between commits

```
git diff 4637ff8 07f807d
```

See the differences between these two commits

```
git diff myfilename
```

See changes that you haven't committed yet

Committing binary files

Try to avoid because it is a lot of storage and will balloon the size of your repository

.gitignore files

```
git rm myfile.pdf
git rm -f myfile.pdf
```

Removes physical file and also stage the deletion for commit
Use `-f` for forcing the removal

```
touch .gitignore
```

Creates a .gitignore file
In this .gitignore file, you can type the files you want to ignore, such as `output/bothmodels.rds` or `output/*.png` etc.

Git reset

```
git reset --hard 4637ff8
```

Hard reset, get back to the way our repository looked at this commit

```
git reset --soft 4637ff8
```

Soft reset, want repository to look the way it looked before, but still keep the stuff we had later. Aka moved directory back in time but the contents are the same as the latest (basically, already staged for commit, but not committed)

```
git reset --mixed 4637ff8
```

Fall back to the way repository looked before but not staged for commit yet. Contents are the same as the latest but not staged for commit yet.

Git revert

```
git revert 4637ff8
```

Goes back in time, but makes this a new commit
Takes us into a VIM, and asks us to put a message in
To save and quit VIM, type `:wq` and hit `enter`

Branching basics

A branch is a pointer that points to a commit
The HEAD pointer references the commit that is ur current working repository. Usually points to a particular branch.

Creating branches

```
git branch test
```

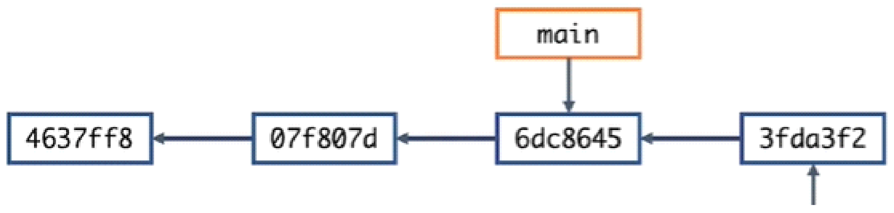
Creates a branch called test

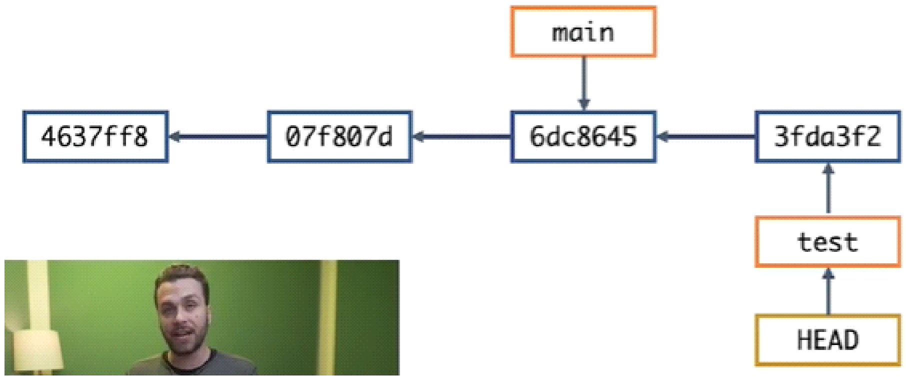
```
git checkout test
```

Switch to branch test (head is now pointing to test branch)
Now you can stage commits on this branch

```
git checkout -b test
```

Creates new branch and also does the checkout. It is a shortcut

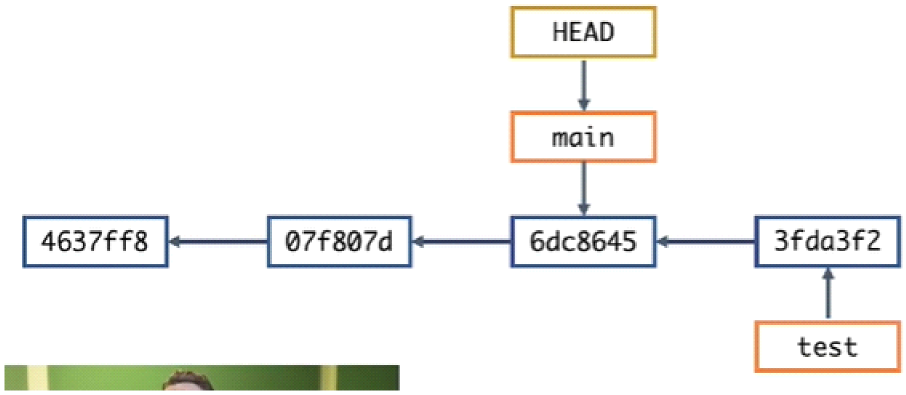




Switching between branches

```
git checkout main
```

Switch back to main branch, and so we won't see the stuff from the test branch. Head points back to main.



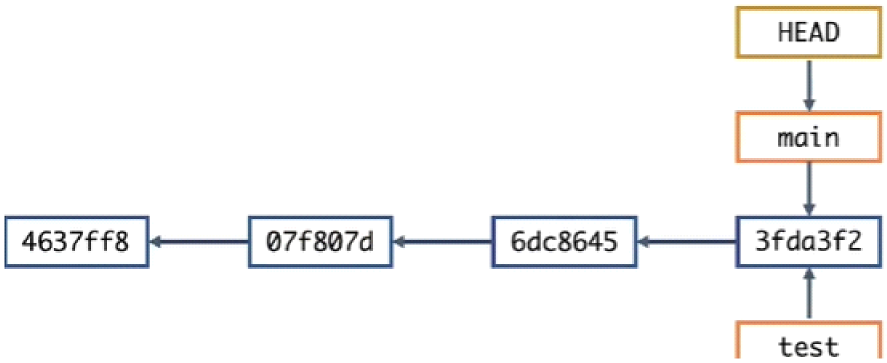
Basic merge (fast forward merge)

```
git checkout main
```

Move head to main branch

```
git merge test
```

Merge test branch into main branch. This is called a fast forward merge.

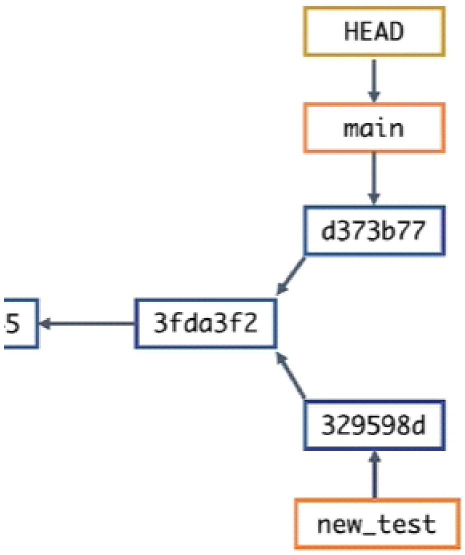


```
git branch -d test
```

Removes the test branch

More complicated merge

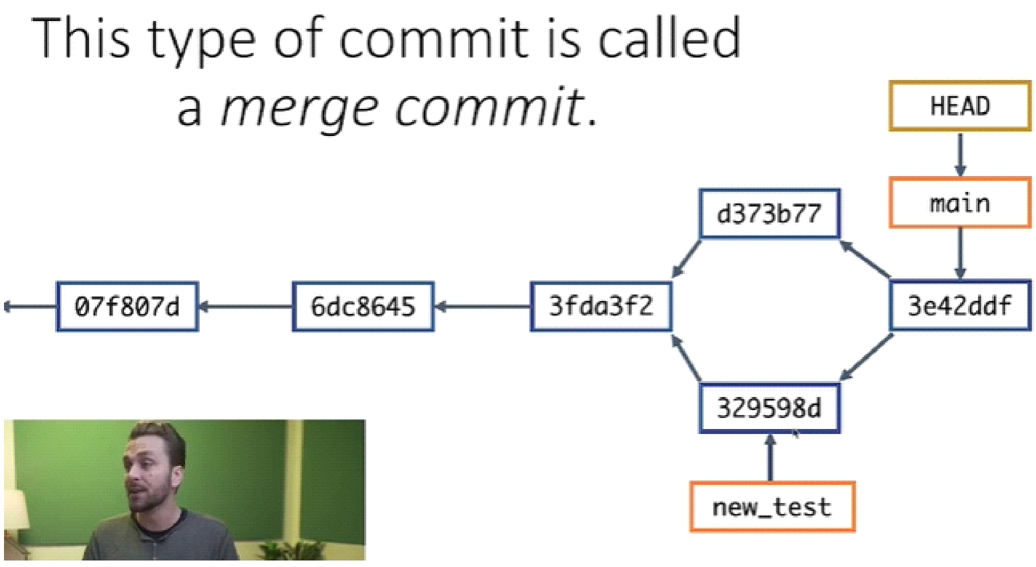
Let's say you have two different branches with two different commits - the history of these commits have diverged.



REMEMBER you want to be on the branch that you want to merge the changes into, most times the main branch. So do `git checkout main`.

```
git merge new_test
```

Adds a new commit that merges the divergent histories. Same command as fast forward merge, but something different is happening under the hood.


This type of commit is called a *merge commit*.

Merge conflicts

When you try to merge, it says automatic merge failed; fix conflicts and then commit the result.

```
<<<<<<<<<< HEAD
All of the lines of code on the main branch that are in conflict w the
lines of code on the conflicting branch
===========
All of the lines of code of the conflicting branch in conflict w the
main branch
>>>>>>>>>> conflict_branch
```

Fix the code and delete the <<<<< and >>>>> stuff

Then you can do `git add` and `git commit` as before.

Real-life workflows with branches
Scenario: Changes may break existing code OR you are working with collaborators.
How to address: Here, we might want to create our own separate branch and then merge at the end.

A more complicated real life scenario - merging main into dev branch
Scenario: Collaborator has requested a very complex change.
How to address: Make periodic updates to development branch based on updates to main branch. So make merges of the main branch INTO the dev branch occasionally.