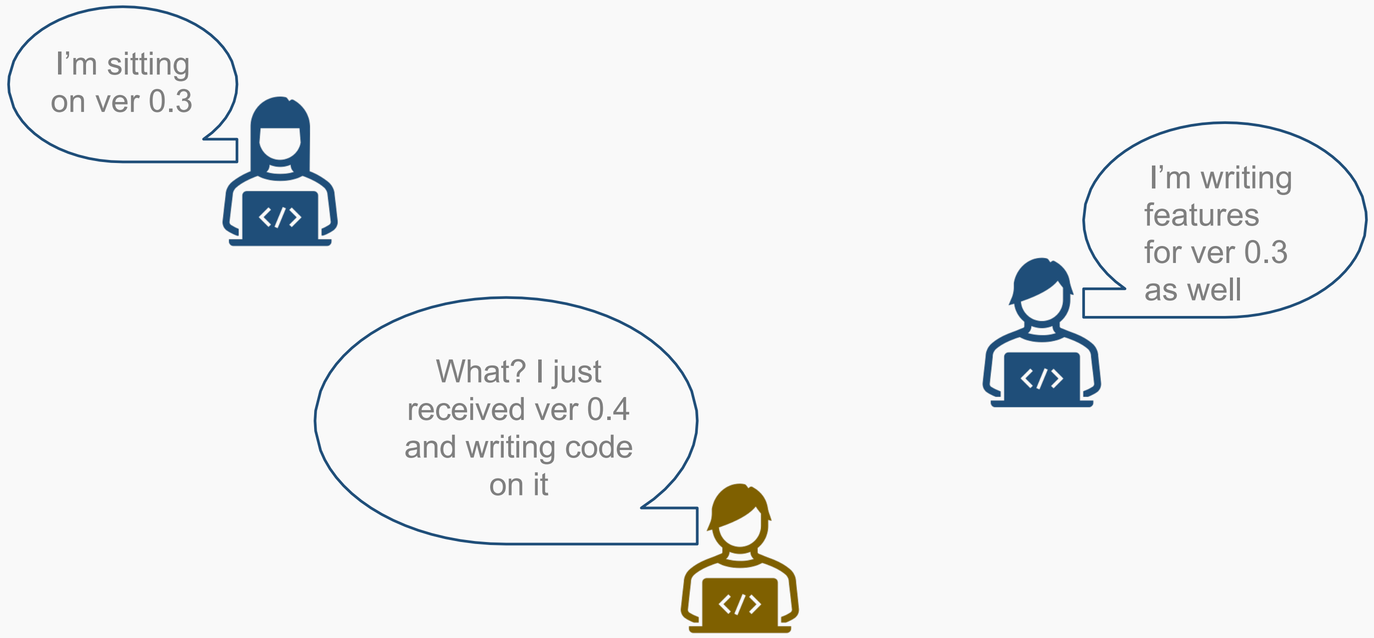
Esoon Ko

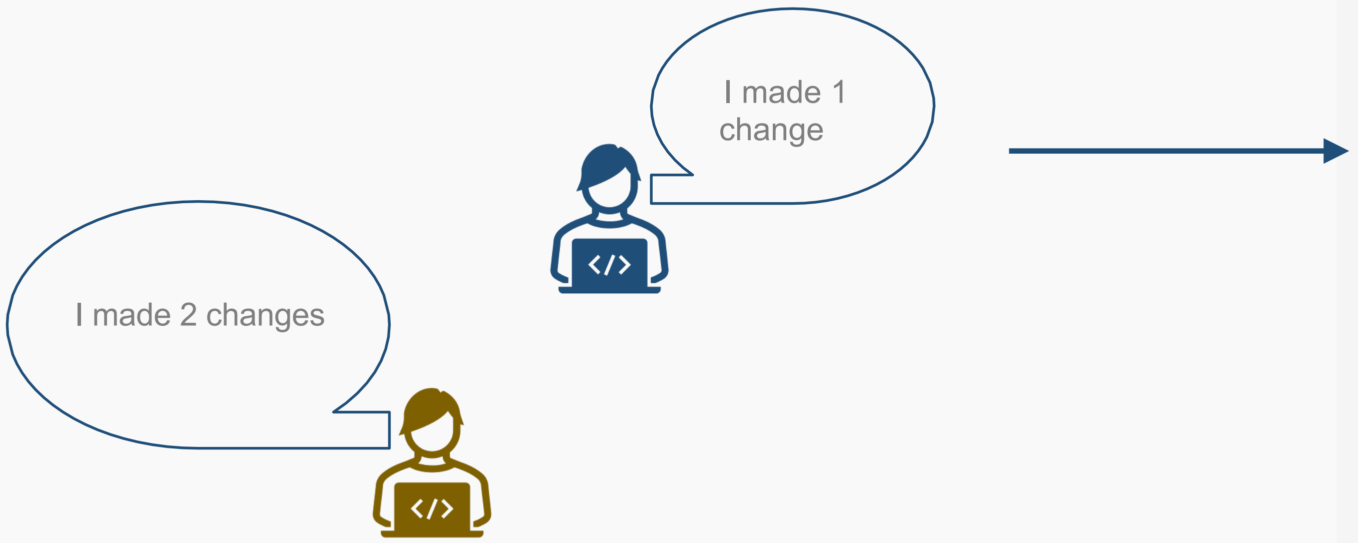
Git and Version Control

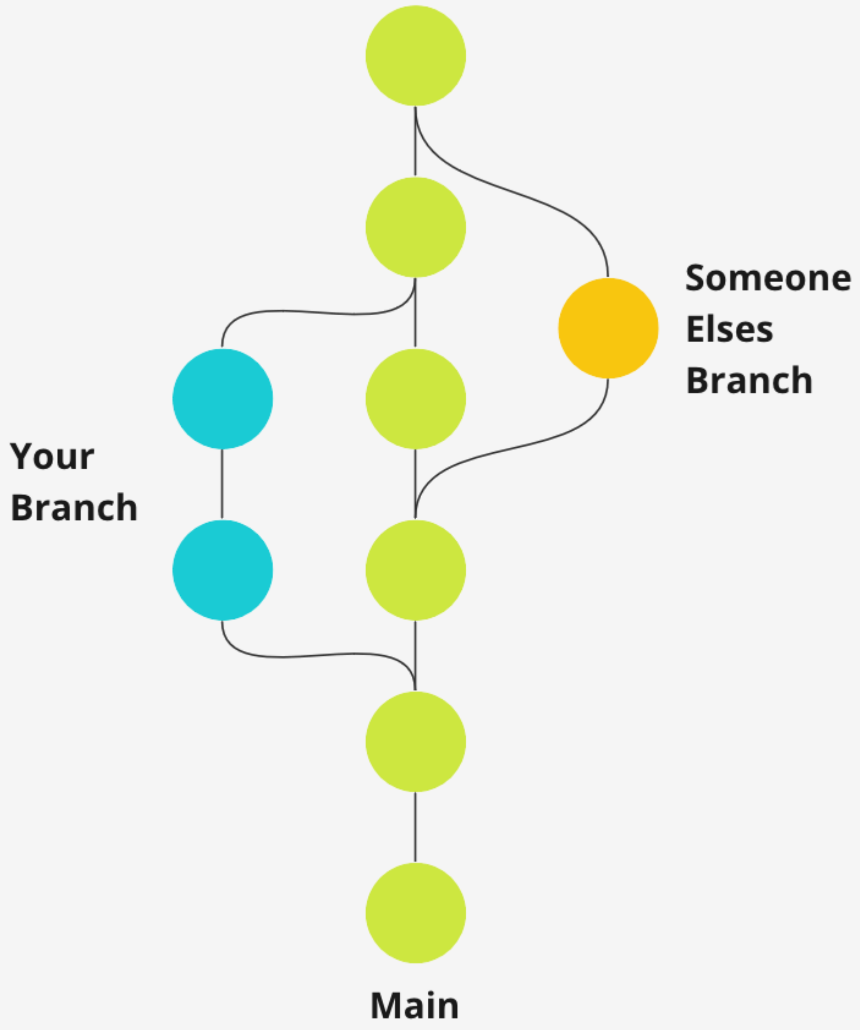
Why **Version Control?**


I'm sitting on ver 0.3
What? I just received ver 0.4 and writing code on it
I'm writing features for ver 0.3 as well

We need a way to **keep track of changes**

We do that through **version control**

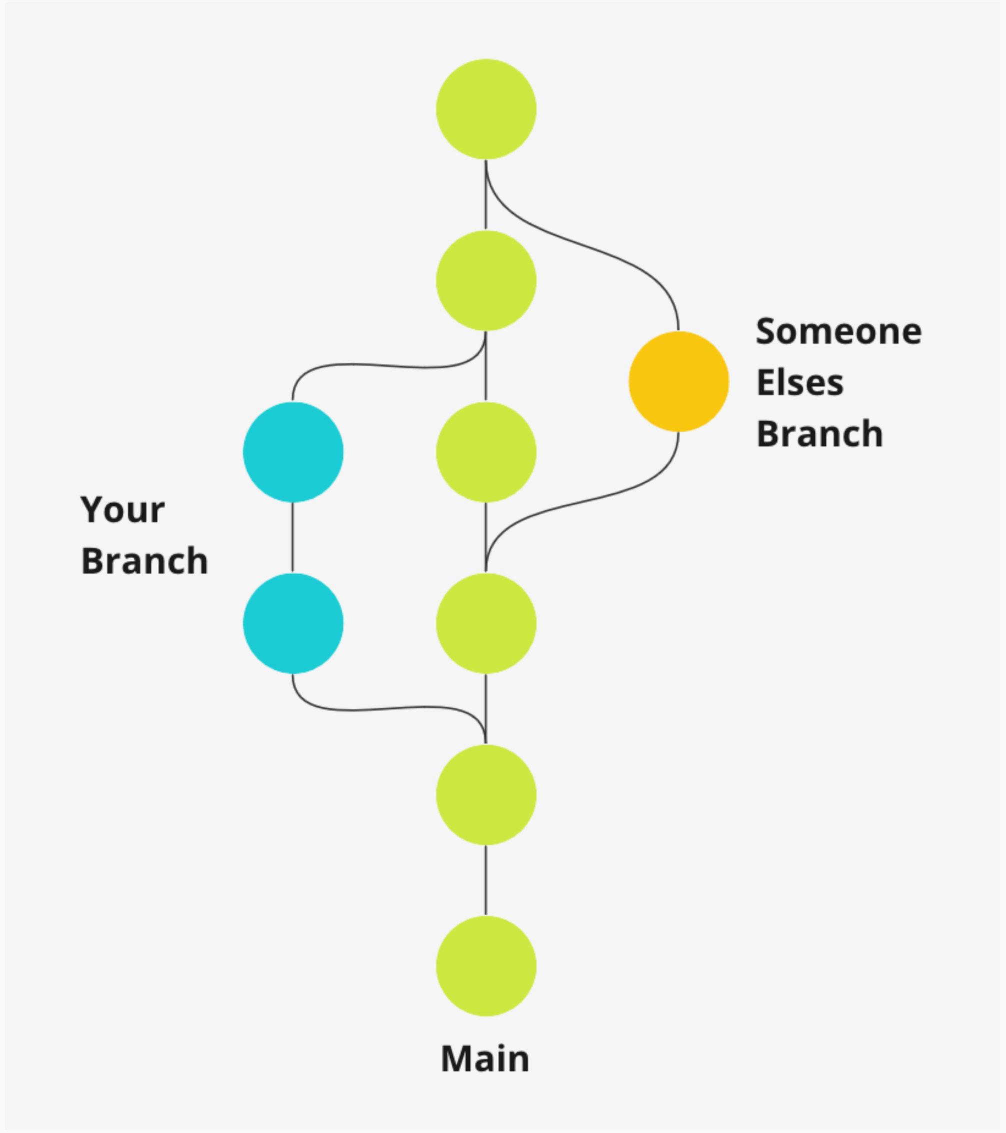




So what is **Version Control?**

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

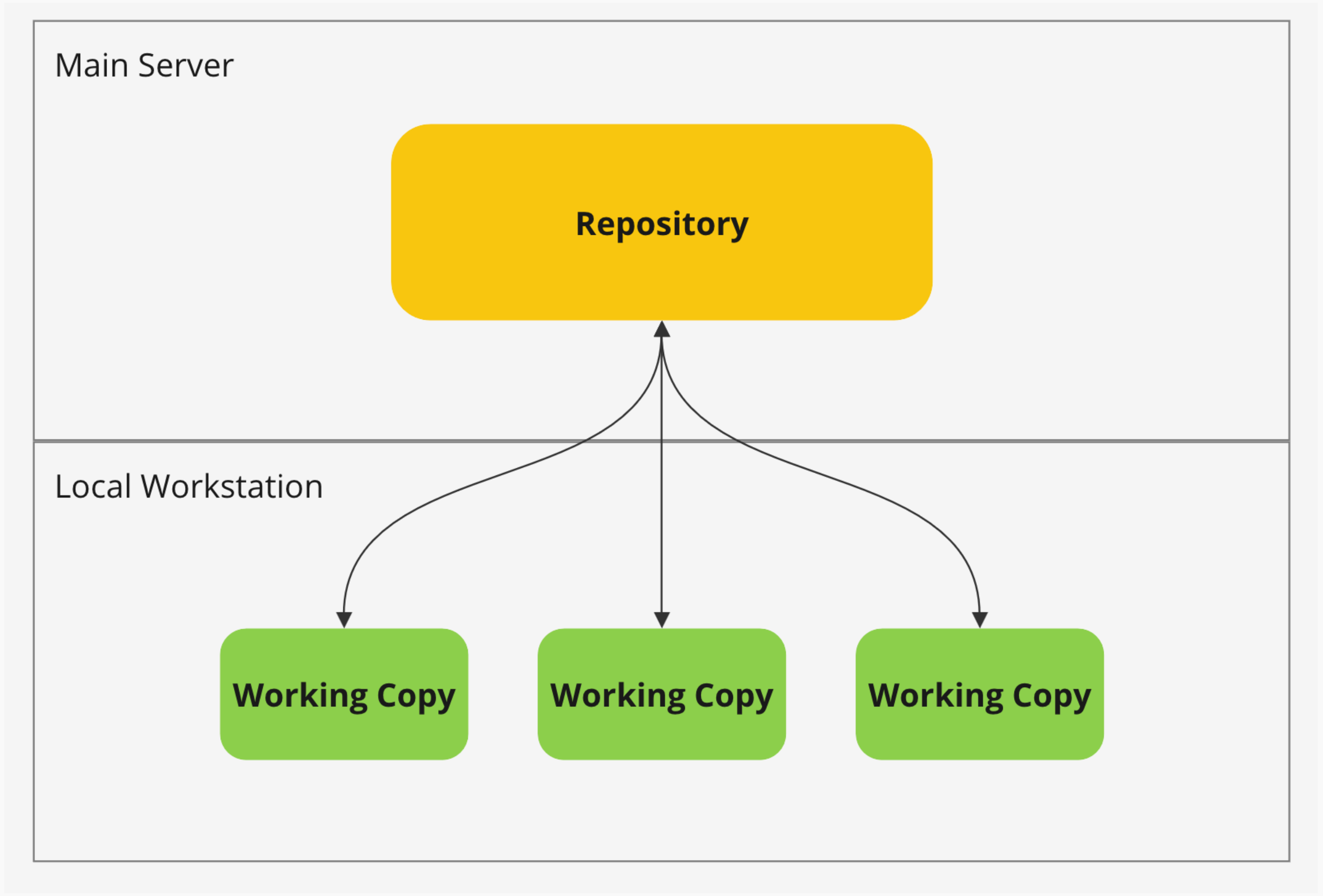
Allows for:
- Collaboration
- Backup and Restore
- Track Changes
- Branching and Merging
- Reduced Duplication and Backup Overhead



Centralized Version Control

Central server to store all versions of a file. Collaborators check files in and out from this central place.

Limitation: Single point of failure. If the server goes down, no one can collaborate or save versioned changes.

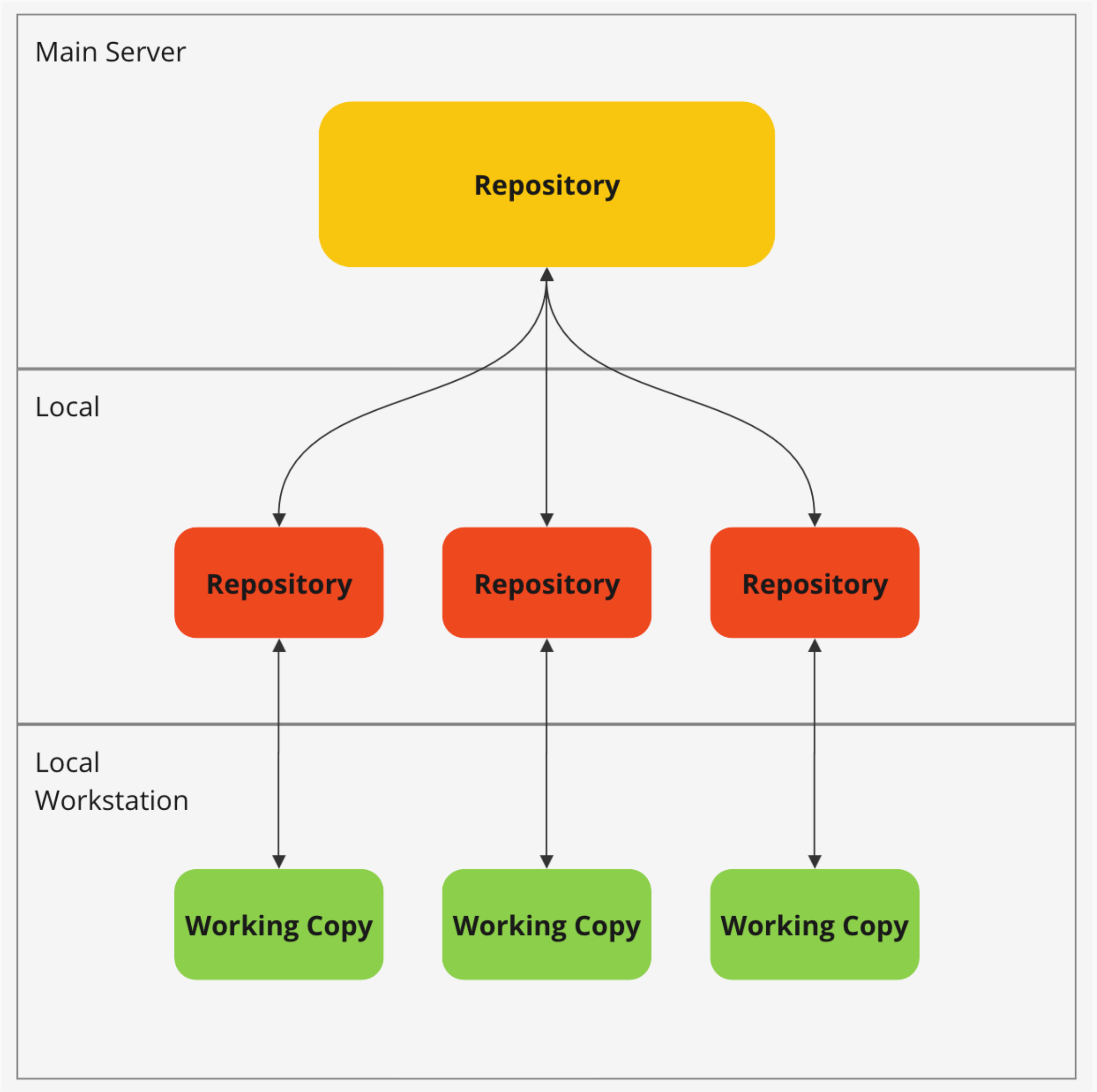


Git: Decentralized Version Control

Every collaborator has a full copy of the entire repository. Each copy is a full backup of all data.

Advantages: Redundancy, no single point of failure, improved collaboration workflow.

Git is a decentralized Version Control!



Buckle up! **Git terminology/structure** time!

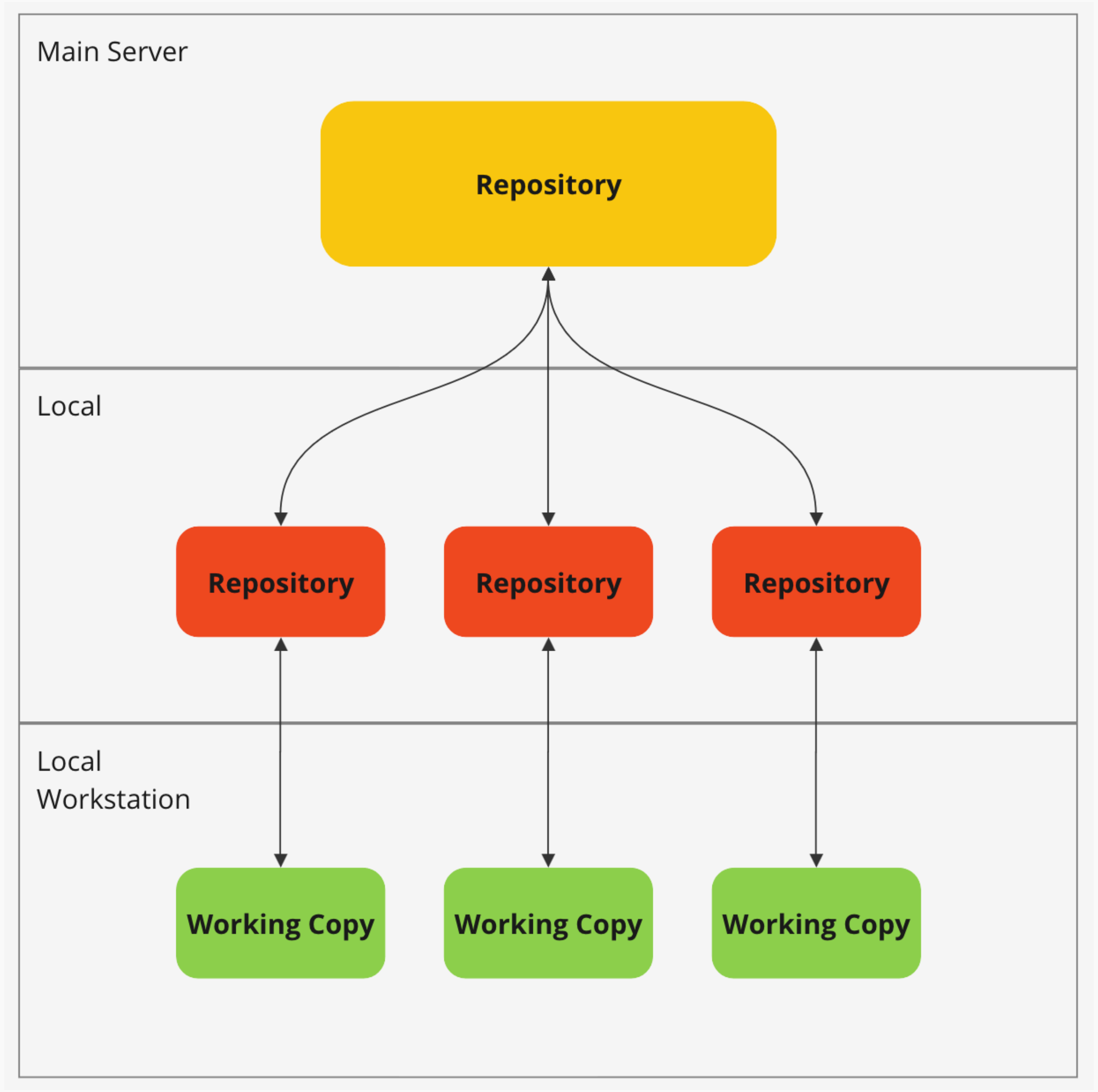
Terminology we will go through:

- Repository (Repo)
- Commit
- Branch
- Merge
- Remote
- Clone
- Push
- Pull
- Fetch
- Pull Request
- Fork
- Stash
- Conflict

Repo

Storage location where your project's files and the entire history of changes are kept.

Think of it as a project's folder that includes all the files and the history of modifications

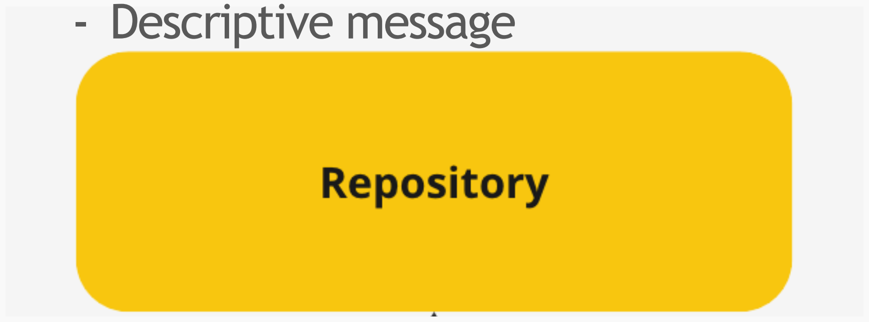


Commit

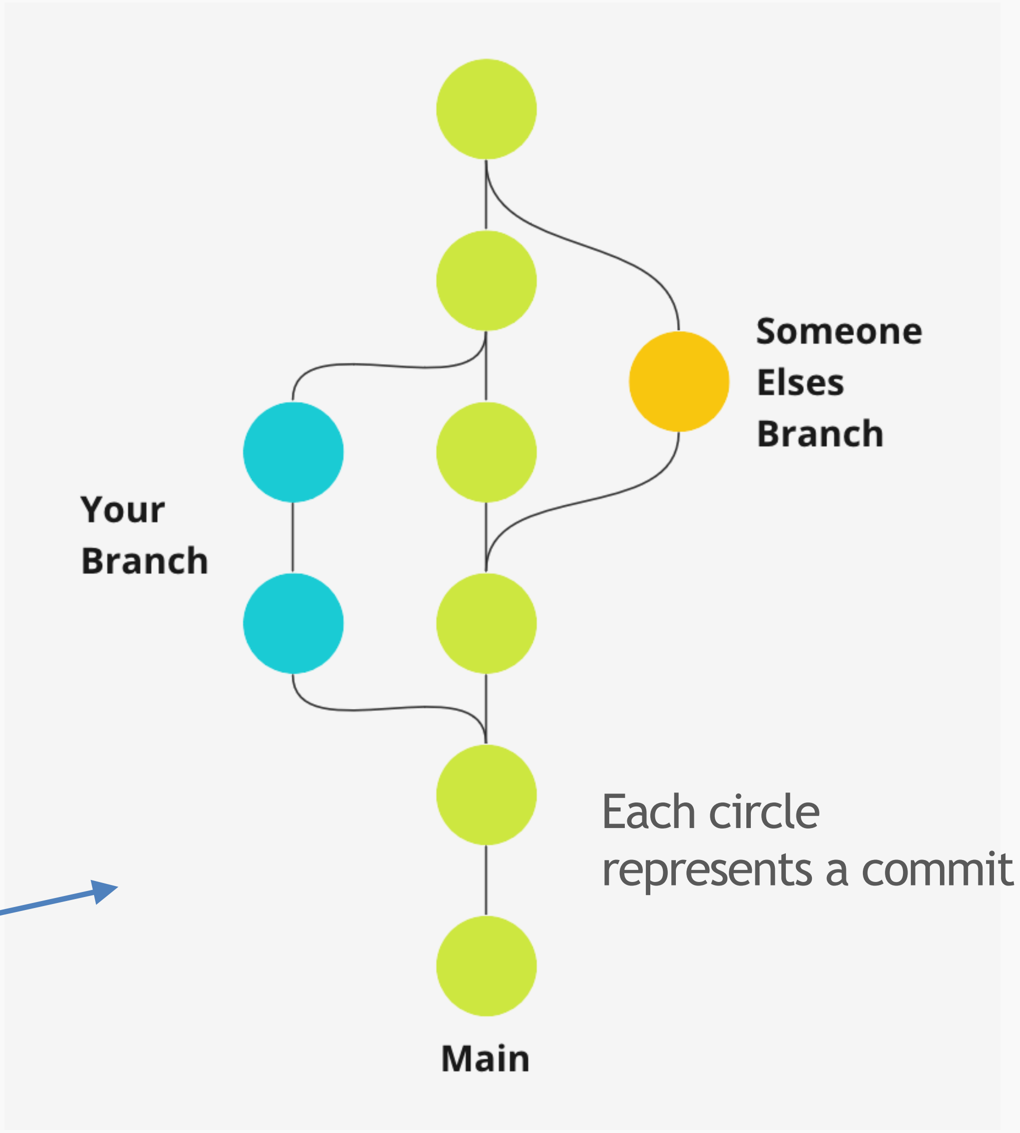
A commit is a snapshot of your repository at a specific point in time.

Every change we want to keep we commit.

Best practice - USE MESSAGE!
- Bad Commit:
    - Add all changes in one commit
    - Messages like “Fixed it...”, “It works...”
- Good Commit:
    - Relevant changes in a commit
    - Descriptive message



Inside the repo

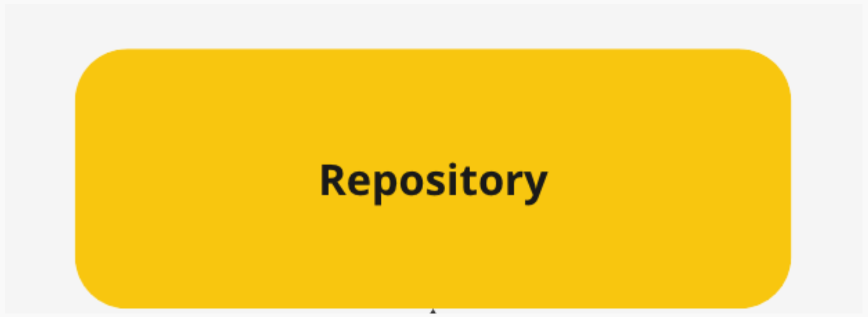


Each circle represents a commit

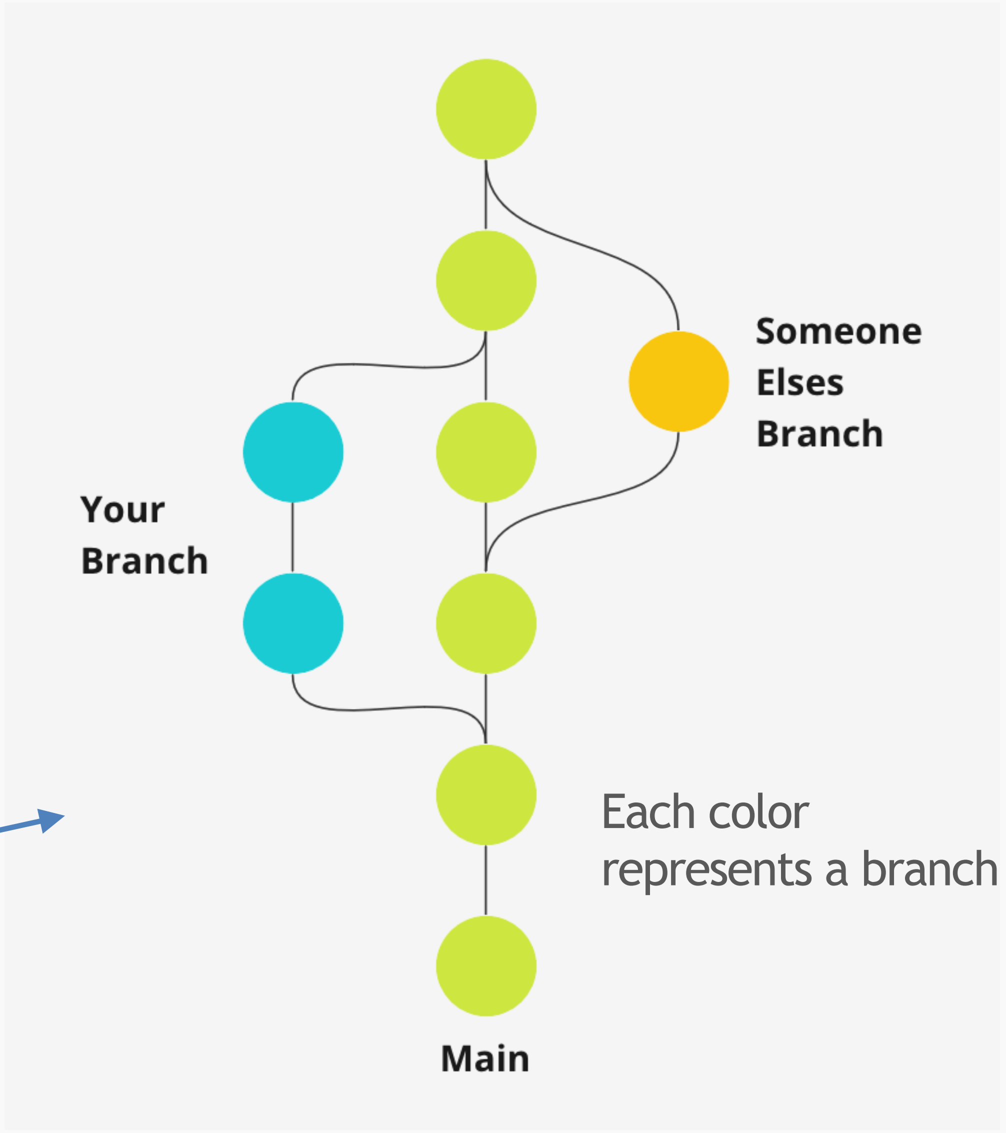
Branch

A parallel version of the repository. It allows you to work on different versions of a project simultaneously.

As the name suggests, a different path that the project takes

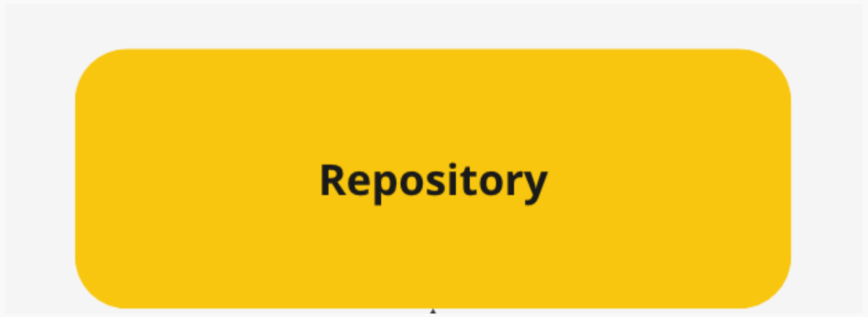


Inside the repo

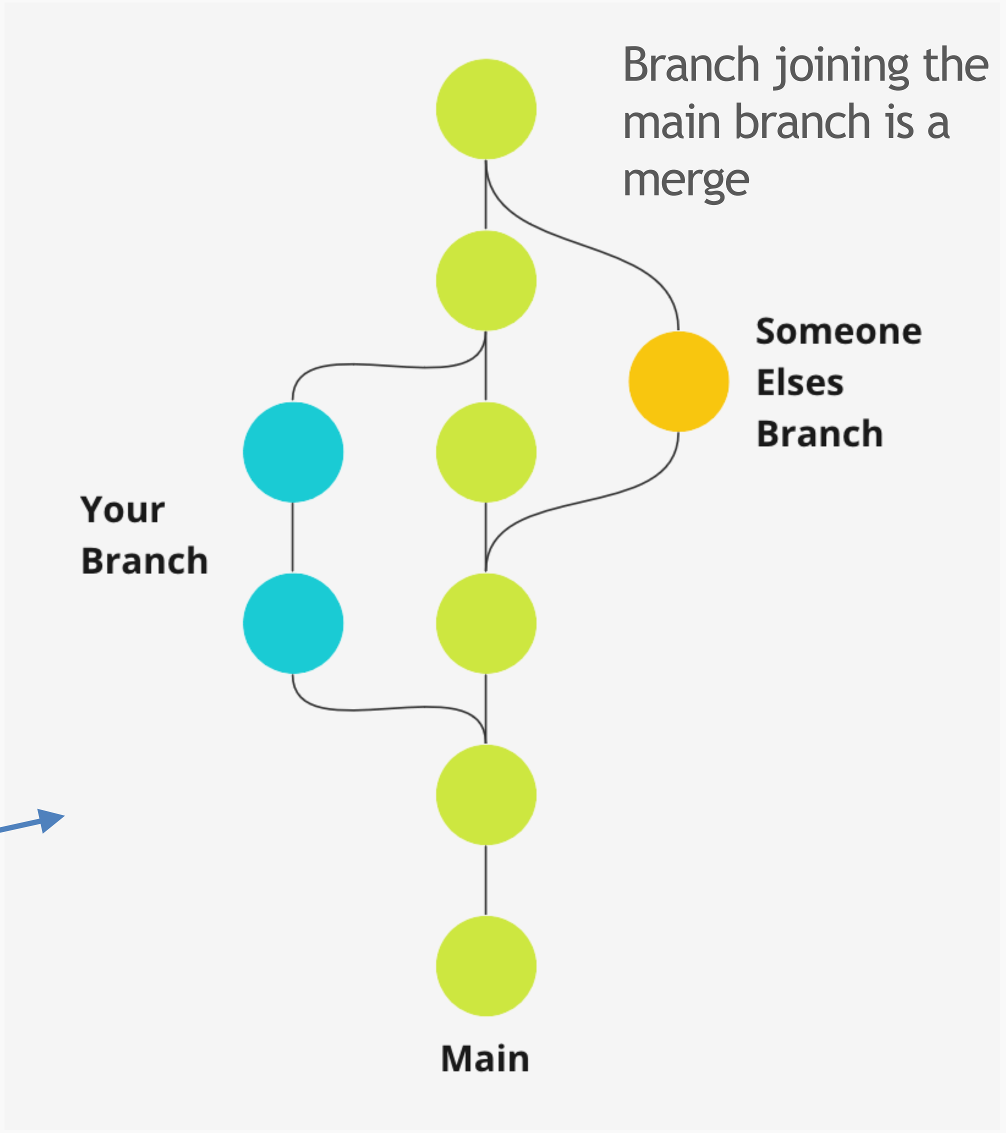


Merge

Merging is the process of integrating changes from one branch into another.



Inside the repo

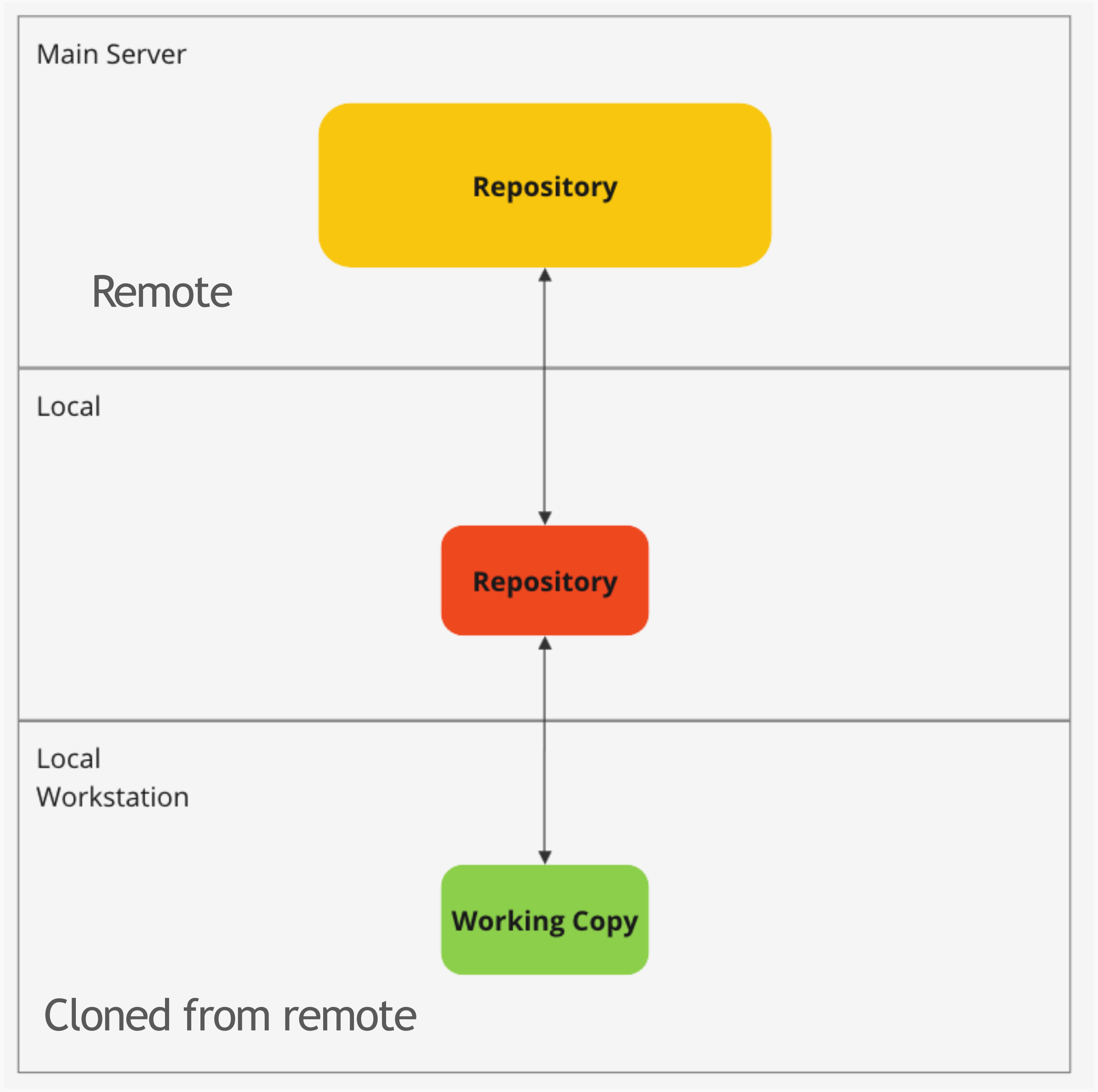


Remote

A remote is a common repository stored on a server, allowing team members to collaborate.

Clone

Cloning is creating a copy of an existing repository.



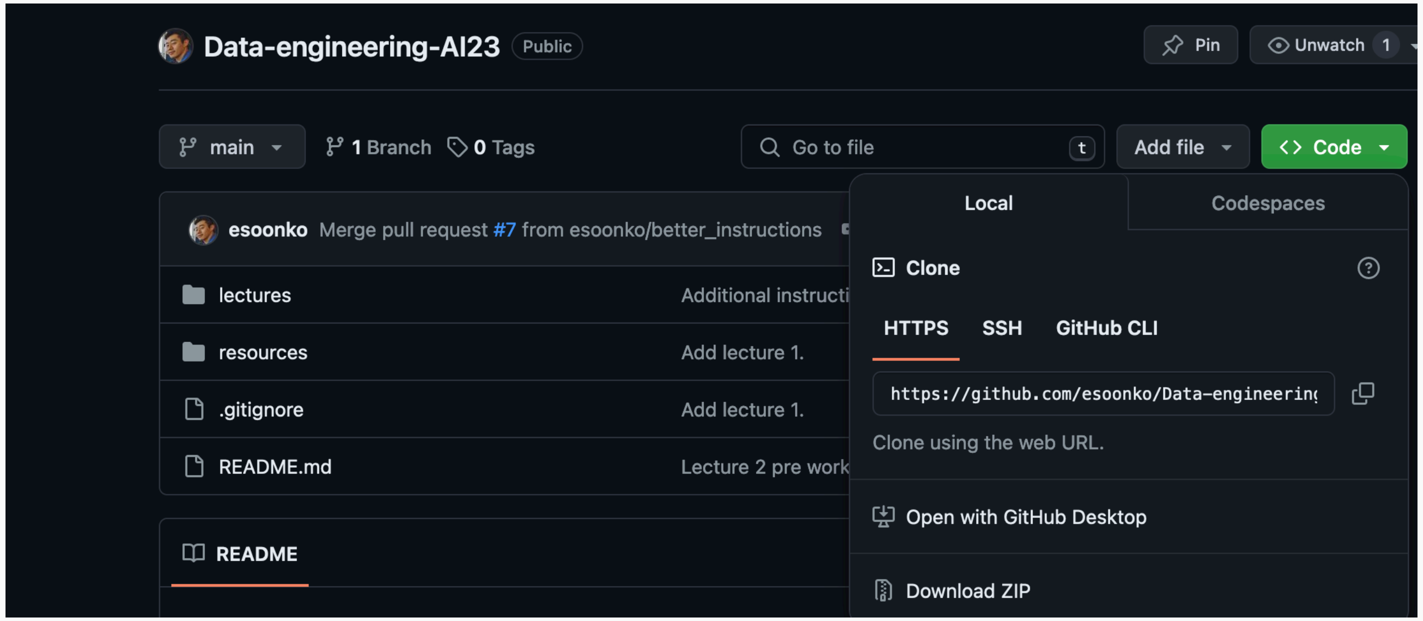
Lets try some Git

First thing first:

- Installing git
- Setting up SSH

Lets try some Git

Lets try cloning a repo! (Those who already cloned my repo can skip this phase)



esoonko Merge pull request #7 from esoonko/better_instructions

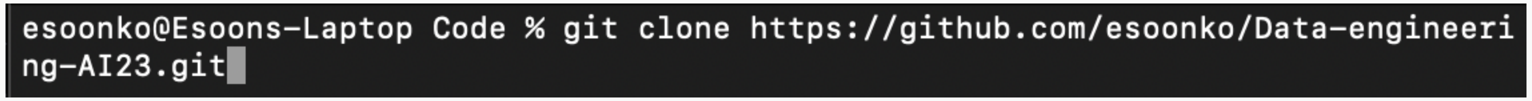
lectures Additional instructi
resources Add lecture 1.
.gitignore Add lecture 1.
README.md Lecture 2 pre work

https://github.com/esoonko/Data-engineering

Clone using the web URL.

Go into my repo to get the HTTPS link.
If you want to clone private repo you do it through the SSH link instead.

Run the command “git clone <repo-url>” to clone the repo in your desired directory.

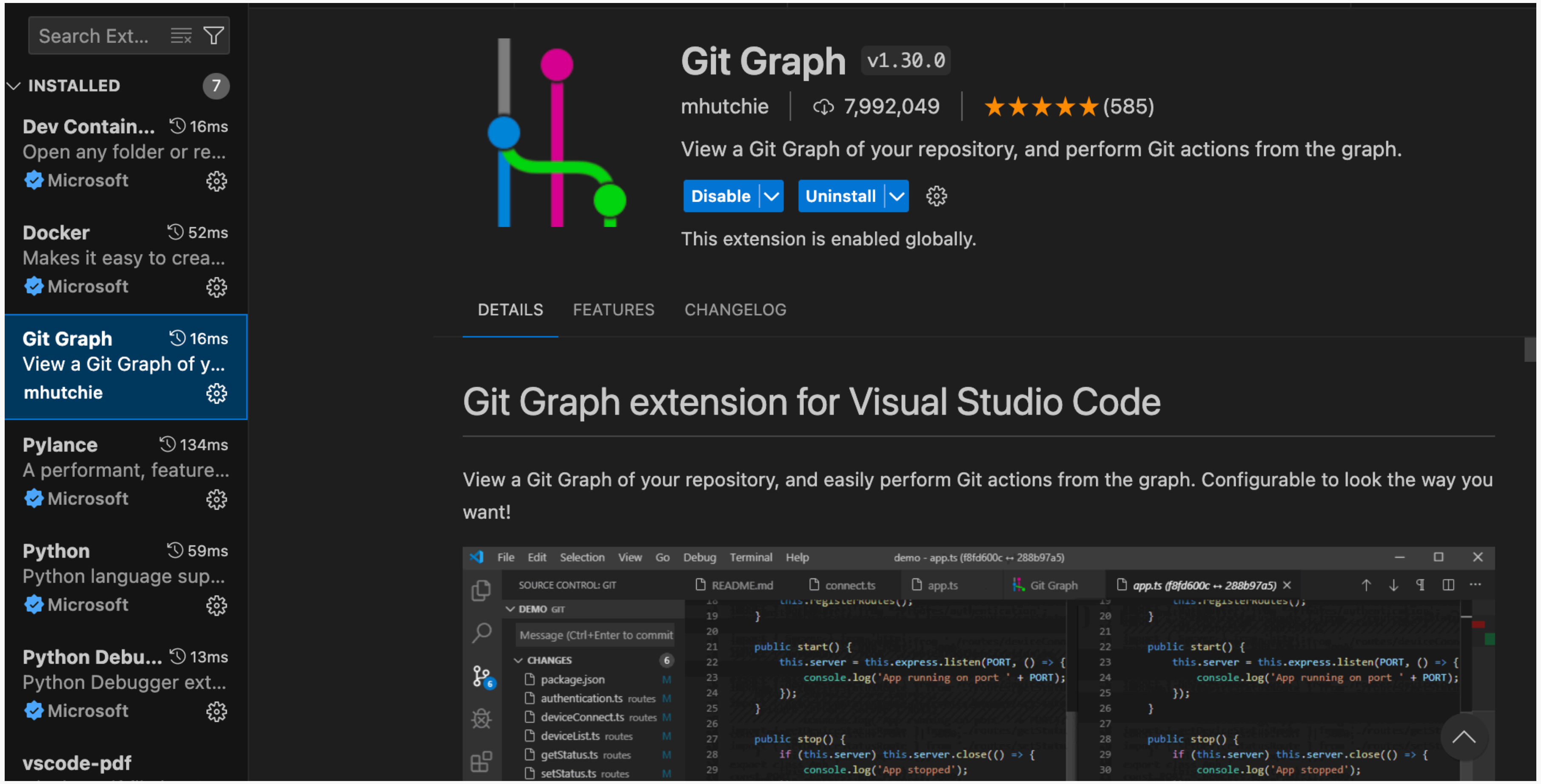


```
esoonko@Esoons-Laptop Code % git clone https://github.com/esoonko/Data-engineeri
ng-AI23.git
```

We have now interacted with:
Repository,
Remote,
Clone

Lets try some Git

If you have VS code I recommend downloading “Git Graph” extension



Dev Contain... 16ms
Open any folder or re...
Microsoft

Docker 52ms
Makes it easy to crea...
Microsoft

Git Graph 16ms
View a Git Graph of y...
mhutchie

Pylance 134ms
A performant, feature...
Microsoft

Python 59ms
Python language sup...
Microsoft

Python Debu... 13ms
Python Debugger ext...
Microsoft

vscode-pdf

Git Graph v1.30.0
mhutchie | 7,992,049 | ★★★★★ (585)
View a Git Graph of your repository, and perform Git actions from the graph.

This extension is enabled globally.

Git Graph extension for Visual Studio Code

View a Git Graph of your repository, and easily perform Git actions from the graph. Configurable to look the way you want!

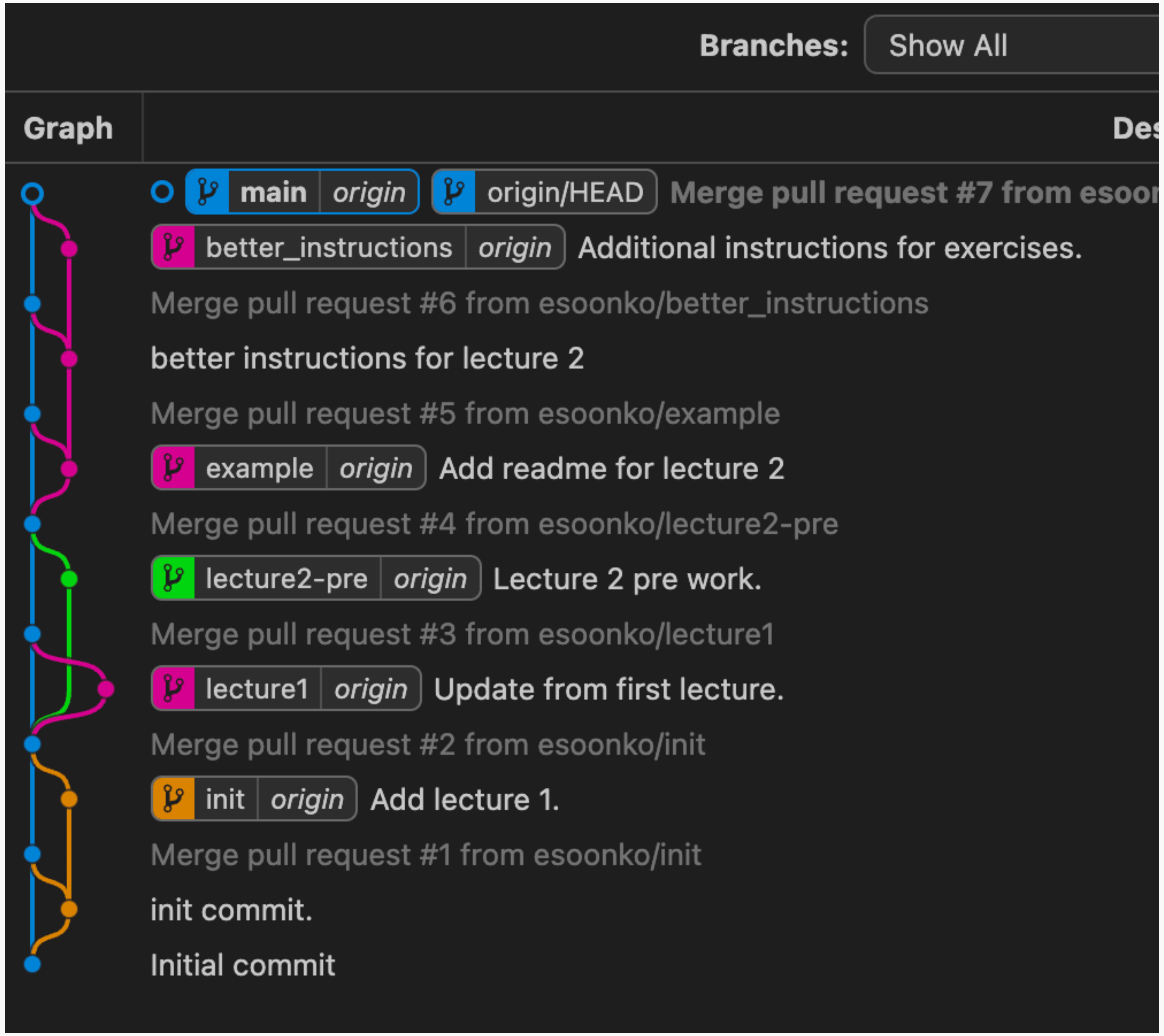
package.json M
authentication.ts routes M
deviceConnect.ts routes M
deviceList.ts routes M
getStatus.ts routes M
setStatus.ts routes M

```
    }

    public start() {
        this.server = this.express.listen(PORT, () => {
            console.log('App running on port ' + PORT);
        });
    }

    public stop() {
        if (this.server) this.server.close(() => {
            console.log('App stopped');
```

```
    }

    public start() {
        this.server = this.express.listen(PORT, () => {
            console.log('App running on port ' + PORT);
        });
    }

    public stop() {
        if (this.server) this.server.close(() => {
            console.log('App stopped');
```

Lets try some Git

With Git Graph we can see:
Commits,
Branches



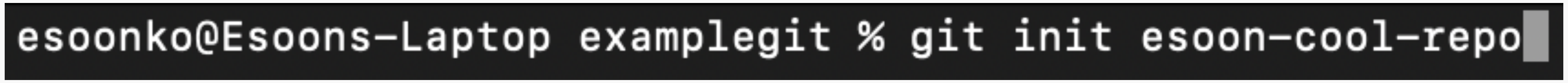
Branches: Show All

Graph | Des

main | origin | origin/HEAD Merge pull request #7 from esoor
better_instructions | origin Additional instructions for exercises.
Merge pull request #6 from esoonko/better_instructions
better instructions for lecture 2
Merge pull request #5 from esoonko/example
example | origin Add readme for lecture 2
Merge pull request #4 from esoonko/lecture2-pre
lecture2-pre | origin Lecture 2 pre work.
Merge pull request #3 from esoonko/lecture1
lecture1 | origin Update from first lecture.
Merge pull request #2 from esoonko/init
init | origin Add lecture 1.
Merge pull request #1 from esoonko/init
init commit.
Initial commit

Lets try creating your own Git

Navigate where you want to create your local git repository and run the command
“git init <your-repo-name>”

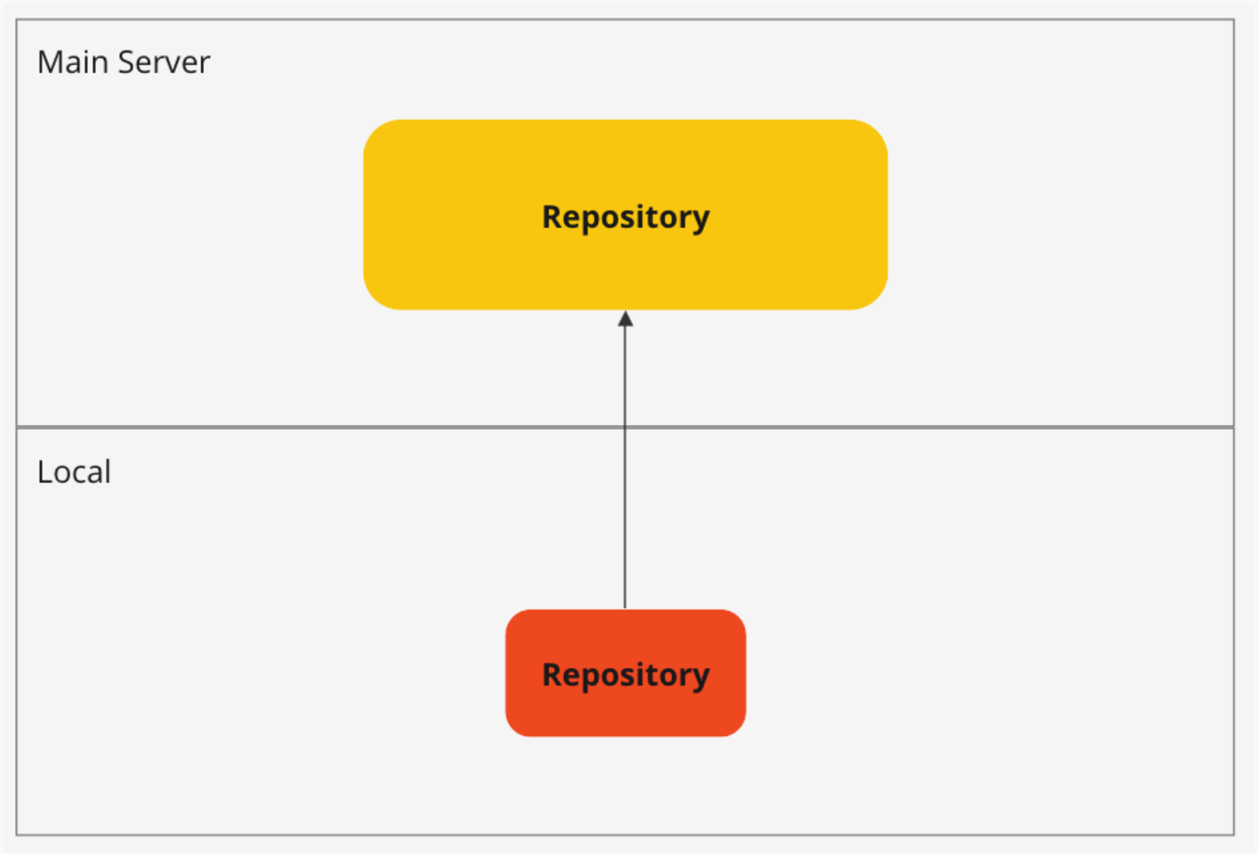


```
esoonko@Esoons-Laptop examplegit % git init esoon-cool-repo
```

Now you have created a empty repo!

Push

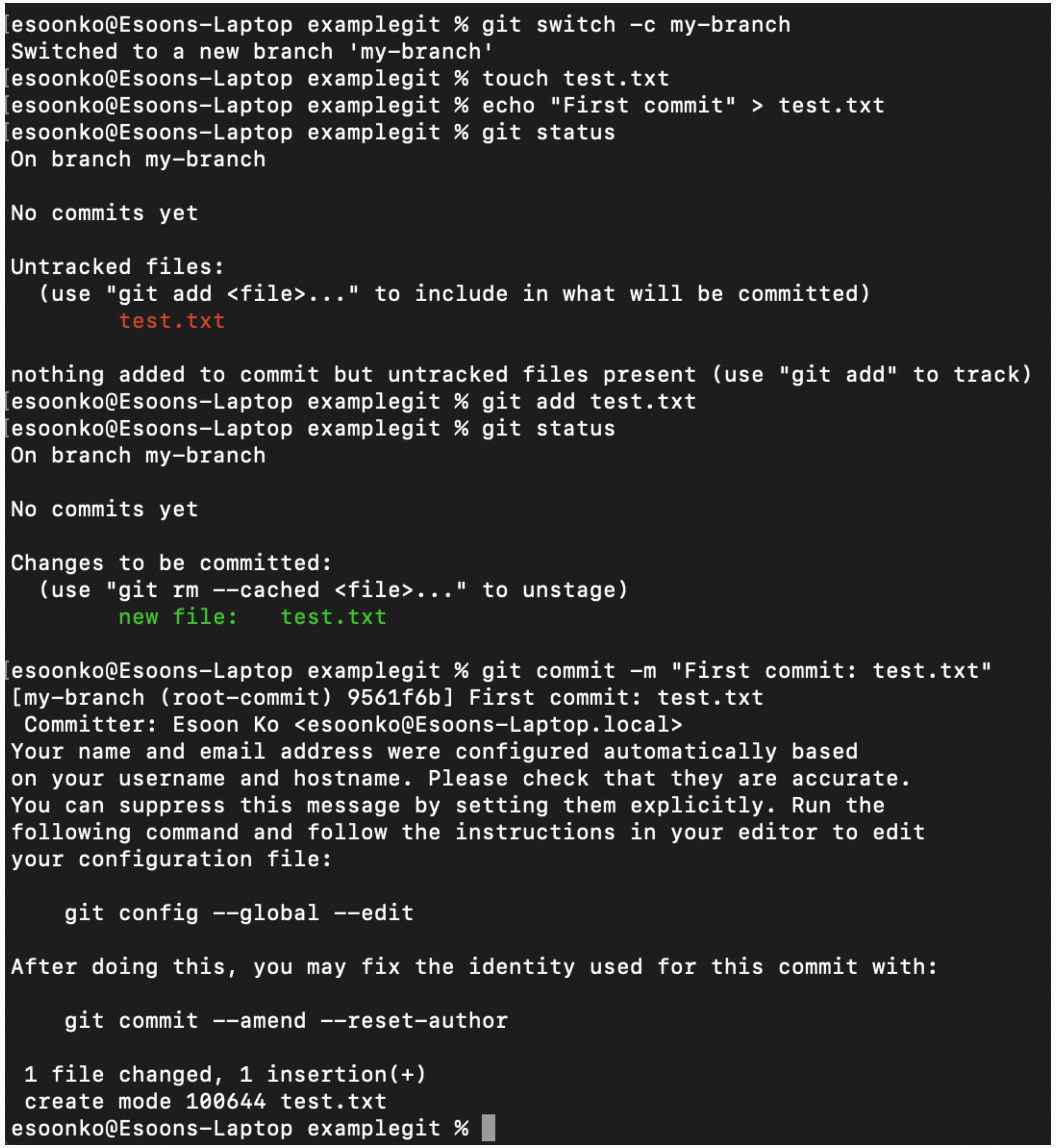
Pushing is sending your committed changes to a remote repository.



More Git

Commit

- Lets create a new branch
    - Either “git switch -c <branch-name>
    - “Git branch <branch-name>” and then “git checkout <branch-name>”
    - (Git wants to go over to switch. -c to create and just switch to switch branch)
- Lets create a new file “test.txt” and insert text
- Lets check the status with git status
- Lets add the files to be committed
- Lets commit the file to the branch.

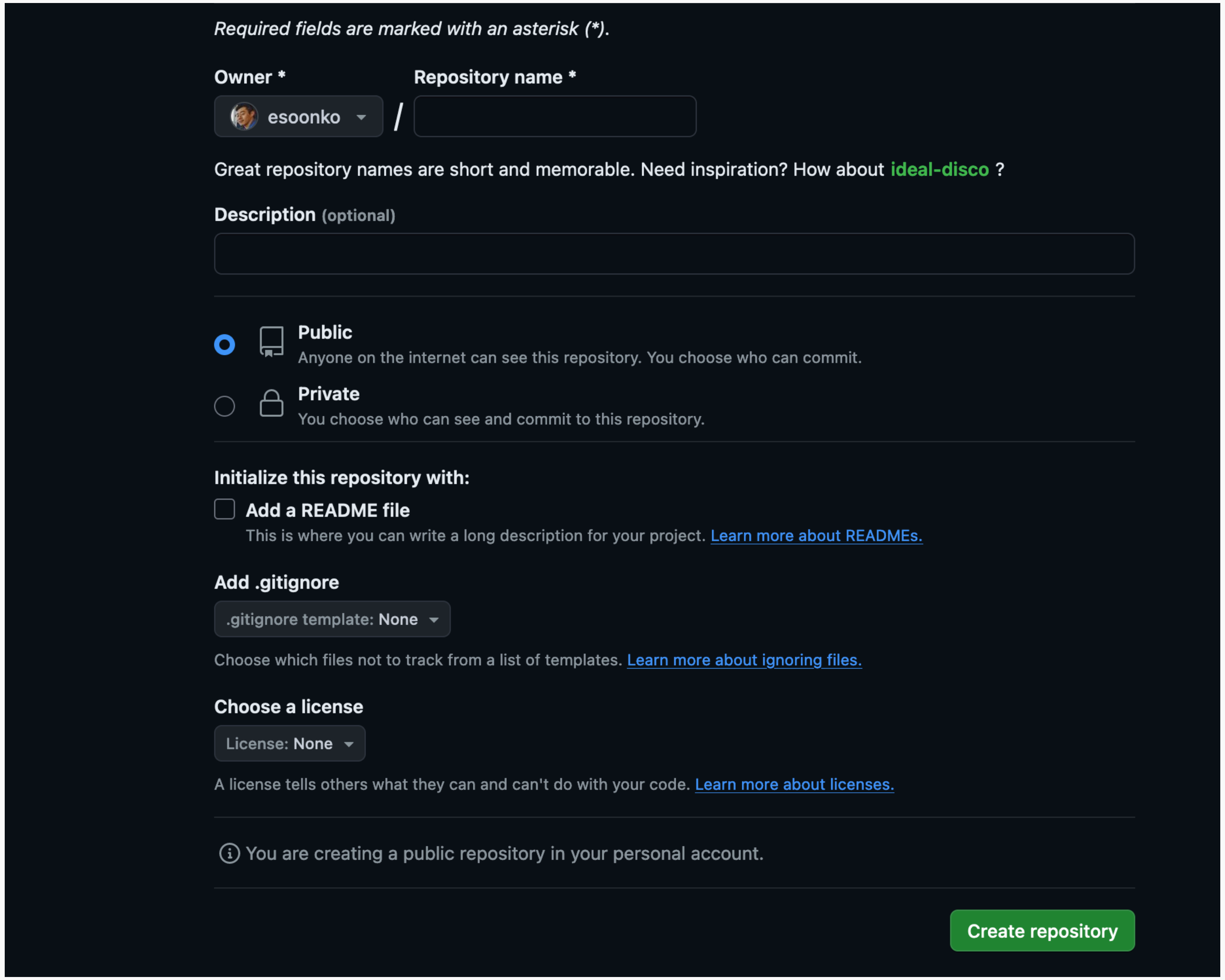


```
esoonko@Esoons-Laptop examplegit % git switch -c my-branch
Switched to a new branch 'my-branch'
esoonko@Esoons-Laptop examplegit % touch test.txt
esoonko@Esoons-Laptop examplegit % echo "First commit" > test.txt
esoonko@Esoons-Laptop examplegit % git status
On branch my-branch

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
esoonko@Esoons-Laptop examplegit % git add test.txt
esoonko@Esoons-Laptop examplegit % git status
On branch my-branch

No commits yet

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

esoonko@Esoons-Laptop examplegit % git commit -m "First commit: test.txt"
[my-branch (root-commit) 9561f6b] First commit: test.txt
 Committer: Esoon Ko <esoonko@Esoons-Laptop.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
esoonko@Esoons-Laptop examplegit % 
```

Connecting local Git to remote

We have tried “git clone”.
We can also connect through
“git remote add origin <repo-url>

- Go to GitHub and create a repo without initializing anything
- Go to your terminal with the git repo you created locally
- Type in “git remote add origin <repo-url>
- Git push -u origin <your-branch>



Required fields are marked with an asterisk (*).

Owner * esoonko / **Repository name ***

Great repository names are short and memorable. Need inspiration? How about ideal-disco ?

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

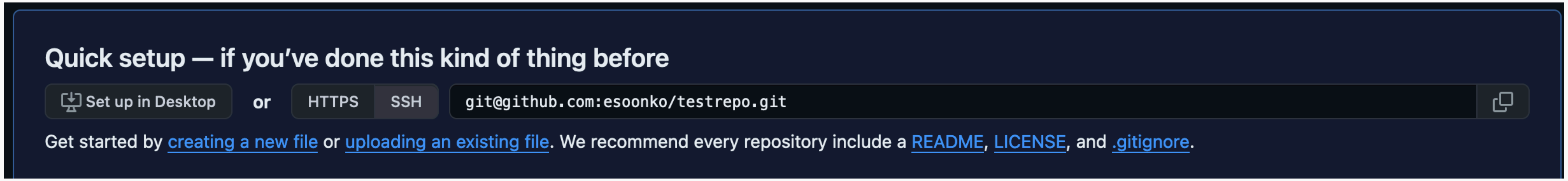
License: None

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.

Connecting local Git to remote

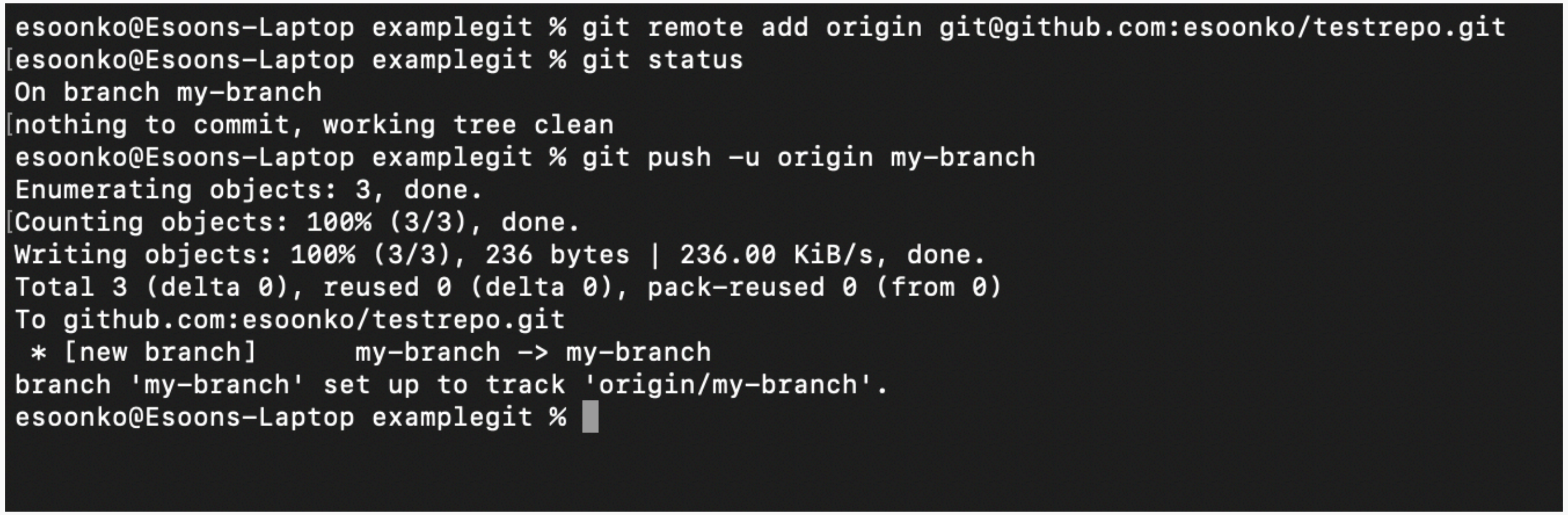
We have tried “git clone”.
We can also connect through
“git remote add origin <repo-url>”


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

Set up in Desktop or HTTPS SSH `git@github.com:esoonko/testrepo.git`

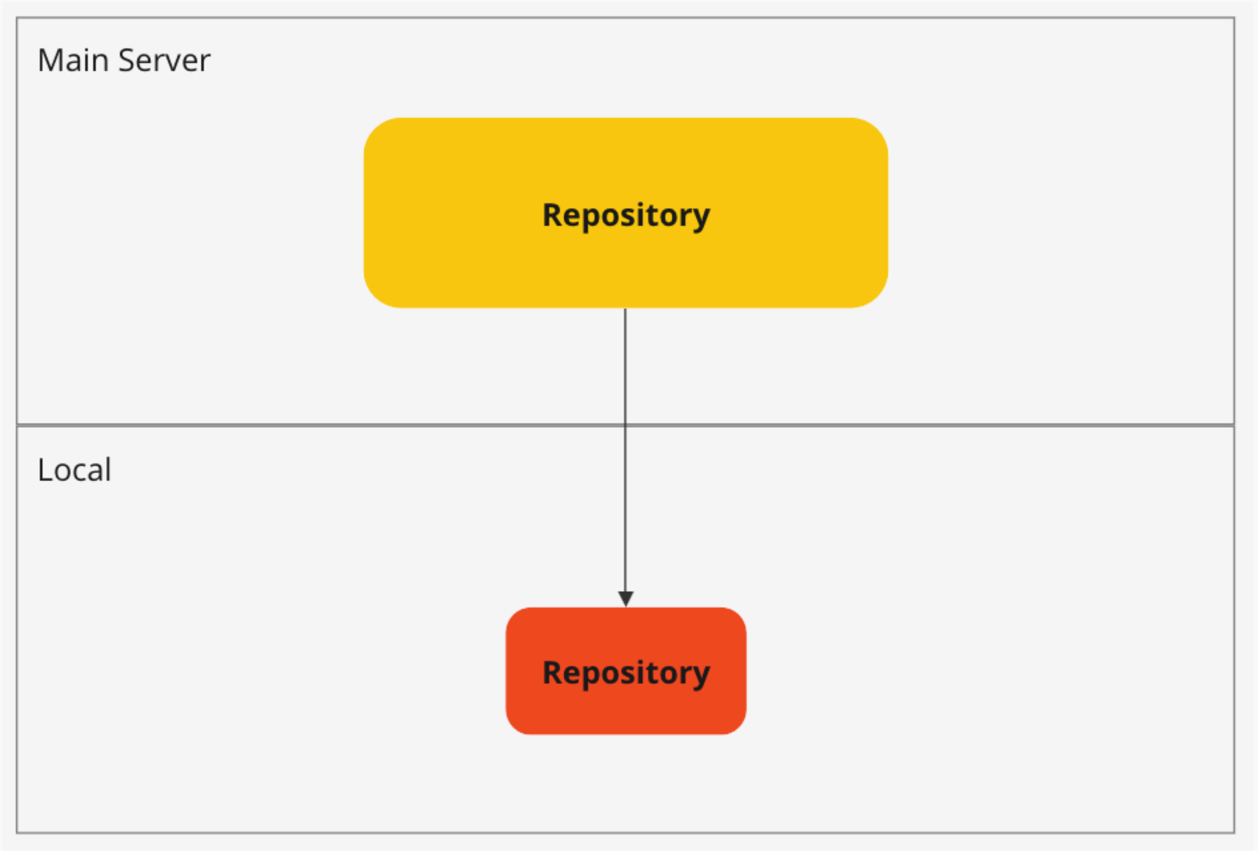
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

- Go to GitHub and create a repo without initializing anything
- Go to your terminal with the git repo you created locally
- Type in “git remote add origin <repo-url>
- Git push -u origin <your-branch>


```
esoonko@Esoons-Laptop examplegit % git remote add origin git@github.com:esoonko/testrepo.git
esoonko@Esoons-Laptop examplegit % git status
On branch my-branch
nothing to commit, working tree clean
esoonko@Esoons-Laptop examplegit % git push -u origin my-branch
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 236 bytes | 236.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:esoonko/testrepo.git
 * [new branch]      my-branch -> my-branch
branch 'my-branch' set up to track 'origin/my-branch'.
esoonko@Esoons-Laptop examplegit % 
```

Pull

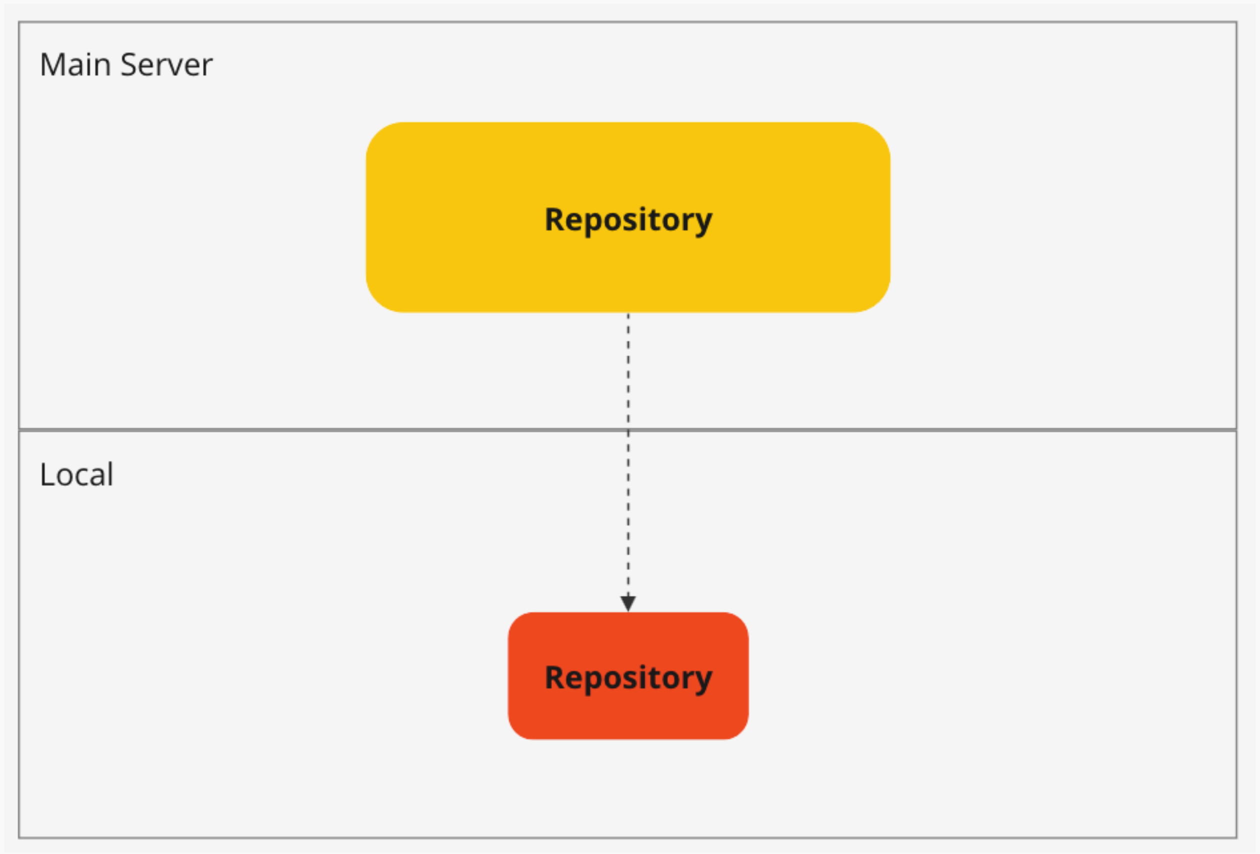
Pulling is fetching and integrating changes from a remote repository into your local repository.



Fetch

Fetching is downloading changes from a remote repository without integrating them into your local repository.

Fetching is like checking for updates on a document but not yet merging them into your current version.



Going back to DE AI - 23 Repo

- **Look at git graph**
- Wait until I say to run “git fetch”
- Look at how the git graph changes
- Notice how there is two main (given that I pushed a new main while you weren’t looking) - One of them has “origin”

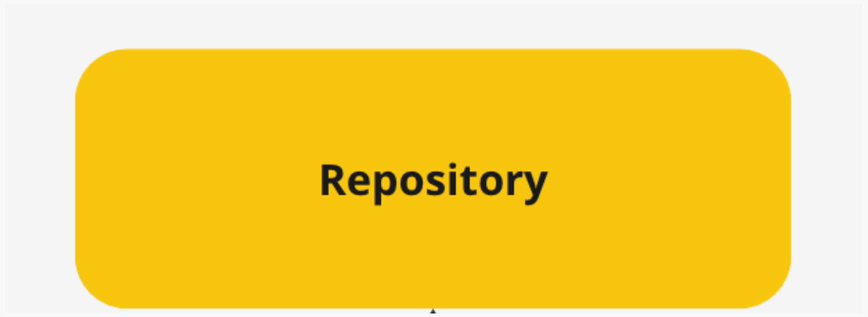
This is because there is a new main in remote

- To confirm pulling the new changes run “git pull” while in main branch.

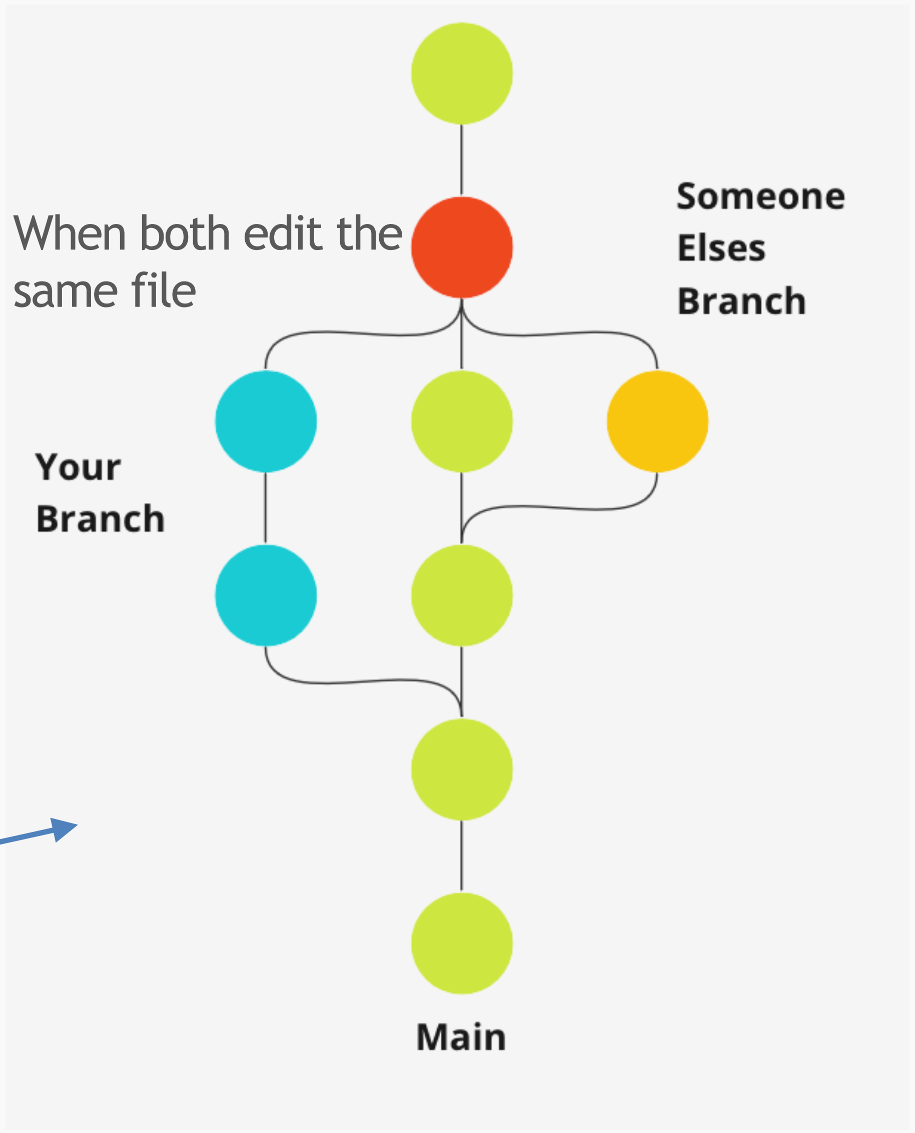
Conflict

A conflict occurs when changes in different branches contradict each other, requiring manual resolution.

Think of a conflict as two different edits on the same paragraph in a document, where you need to decide how to combine them.



Inside the repo



Git:ing Conflict

Lets create a artificial conflict
- Init with “git init”
- Create branch main with “git switch -c main”
- Create a file with text “echo “First commit” > merge-conflict.txt”
- Commit with “git commit -m “This is the first commit.”

- Create a new branch with “git switch -c branch-1”
- Alter the txt file with “echo “Branch-1 commit” > merge-conflict.txt”
- Commit with “git commit -m “This is branch-1 commit.”

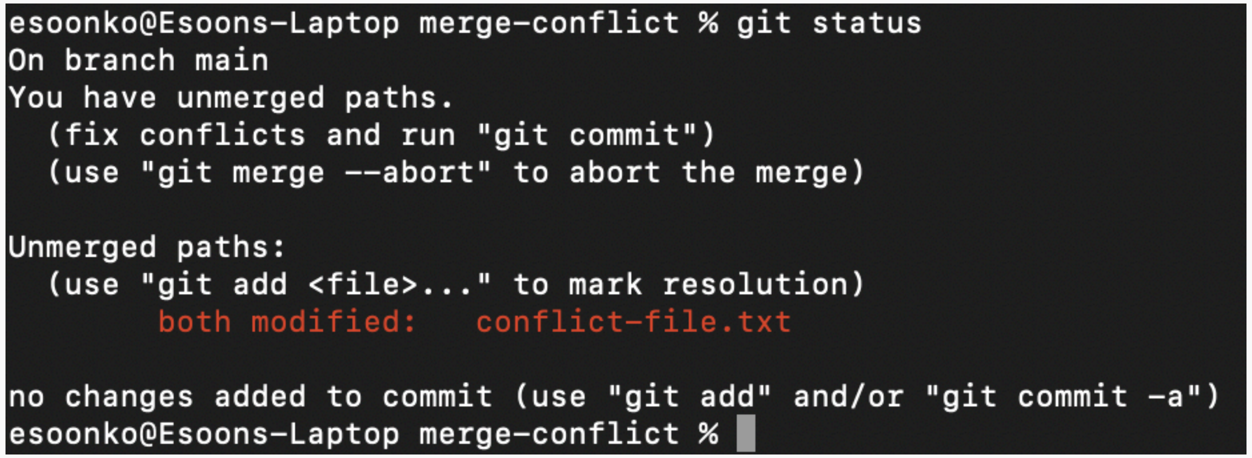
- Switch back to main with “git switch main”
- Check the txt file with “cat merge-conflict.txt”
- Alter the txt file with “echo “Main commit” > merge-conflict.txt”
- Commit with “git commit -m “This is main commit.”

- Merge branch 1 into main with “git merge branch-1 || true”

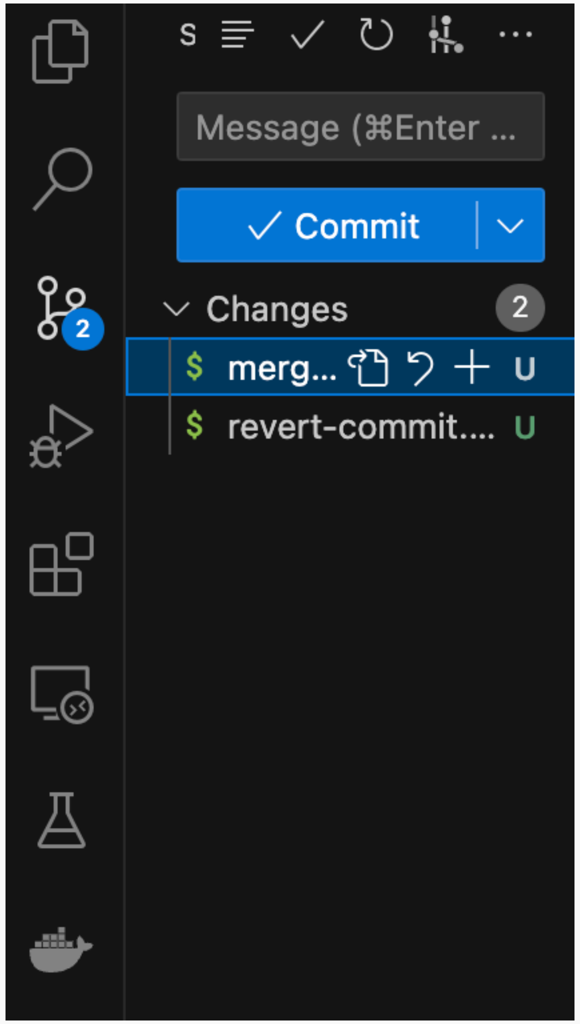
How do we solve it? **Merge conflict**

Several ways:
- Do it through IDE like VS Code
- Do it through the terminal

We have to choose either incoming branch or existing branch.
Delete the other one and commit the changes like a normal commit.



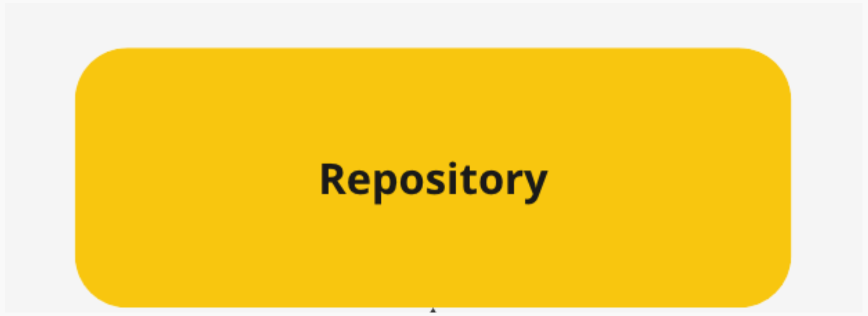
```
esoonko@Esoons-Laptop merge-conflict % git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

no changes added to commit (use "git add" and/or "git commit -a")
esoonko@Esoons-Laptop merge-conflict % 
```



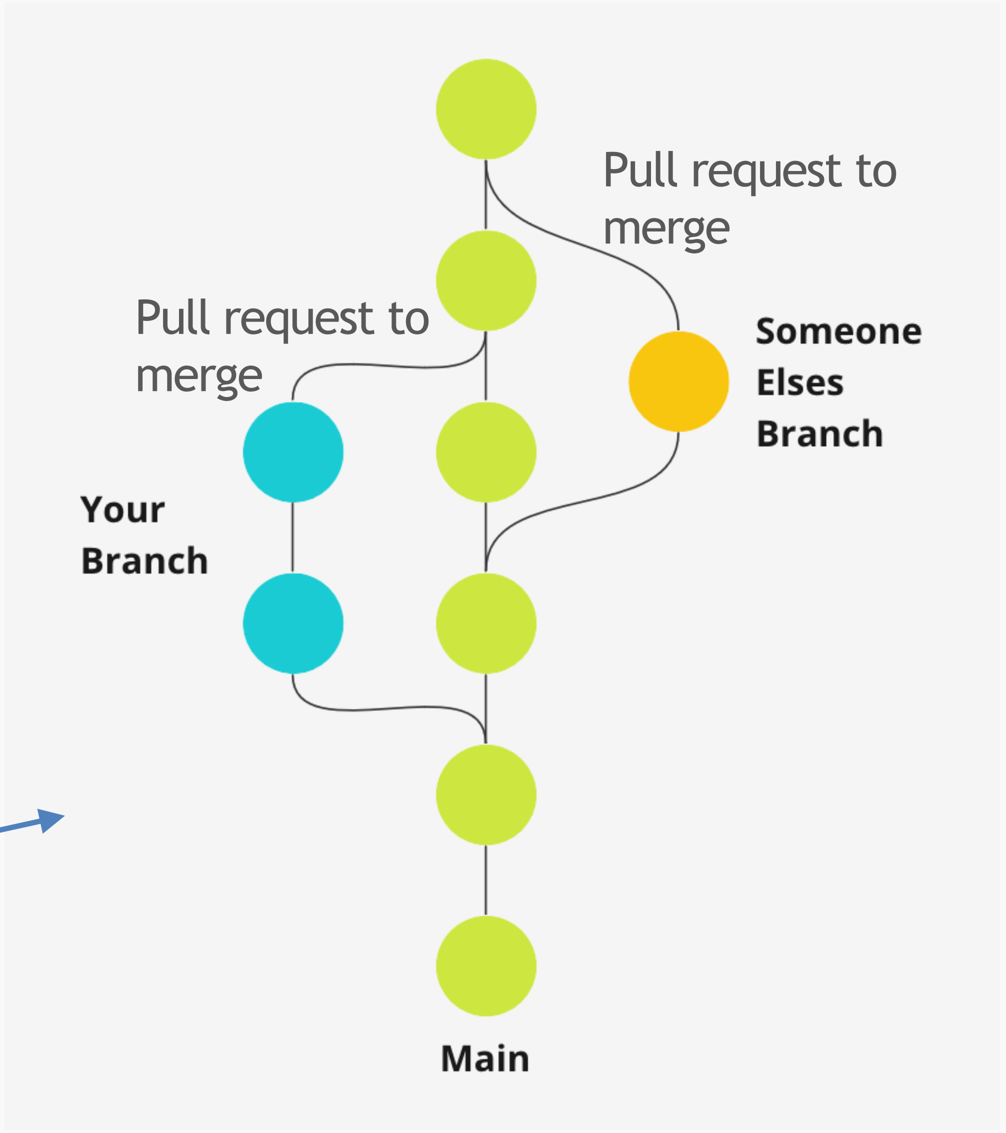
Pull Request

A pull request is a method of submitting contributions to a project. It is a request to merge your changes into the main repository.

Think of a pull request as a proposal to add your changes to the shared project, asking others to review and accept your updates.



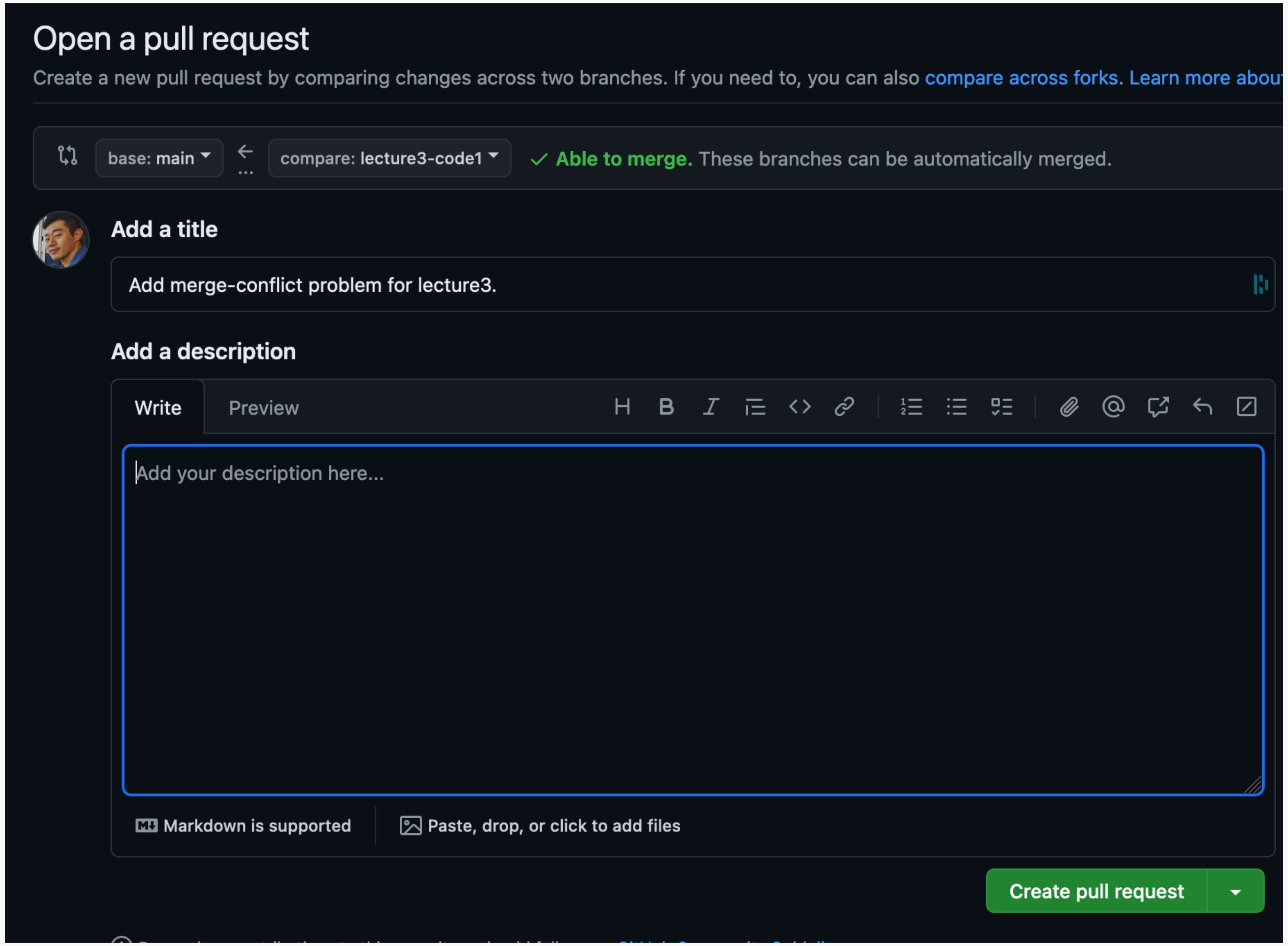
Inside the repo



Pull Request

A pull request is a method of submitting contributions to a project. It is a request to merge your changes into the main repository.

Think of a pull request as a proposal to add your changes to the shared project, asking others to review and accept your updates.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more abou

base: main ← compare: lecture3-code1 ✓ Able to merge. These branches can be automatically merged.

Add a title

Add merge-conflict problem for lecture3.

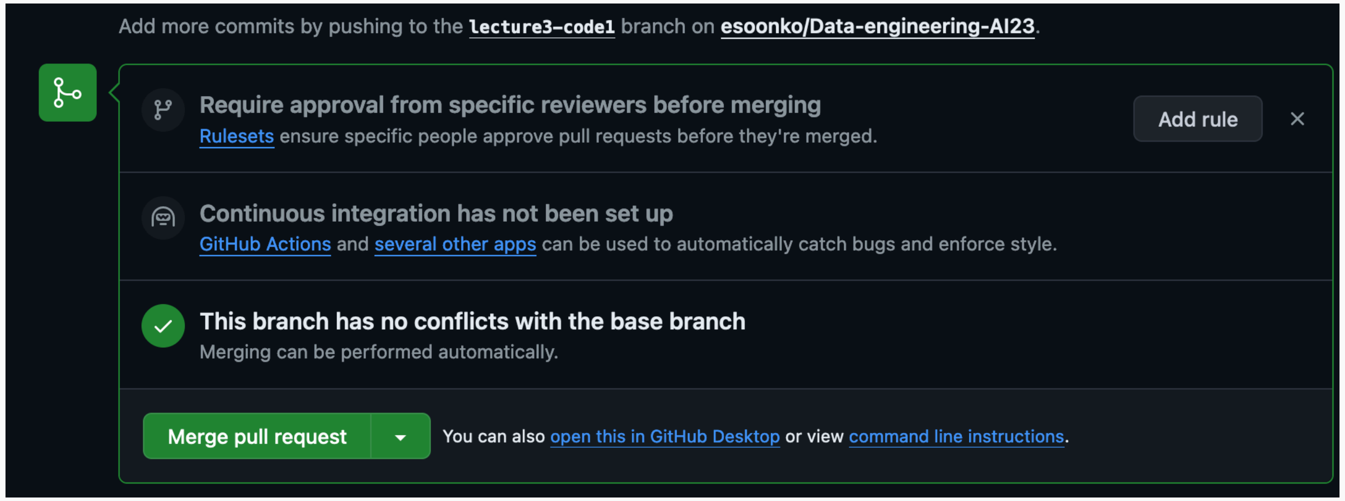
Add a description

Add your description here...

Pull Request: Lets try it

- Go into teams and share to me your GitHub account so I can give access
- Go into the following repo (will be on teams):
- Clone the repo with SSH (Hint - git clone)
- Create a new branch with your name (Hint- git switch -c)
- Create a new folder with your name and a txt file with any name and content.
- Add the changes (Hint git add)
- Commit the changes with a message (Hint: git commit -m)
- Push the changes to remote (Hint: git push)
- In GitHub navigate to the repo and create a pull request.

- Once it is approved you can merge it in


Add more commits by pushing to the `lecture3-code1` branch on esoonko/Data-engineering-AI23.

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

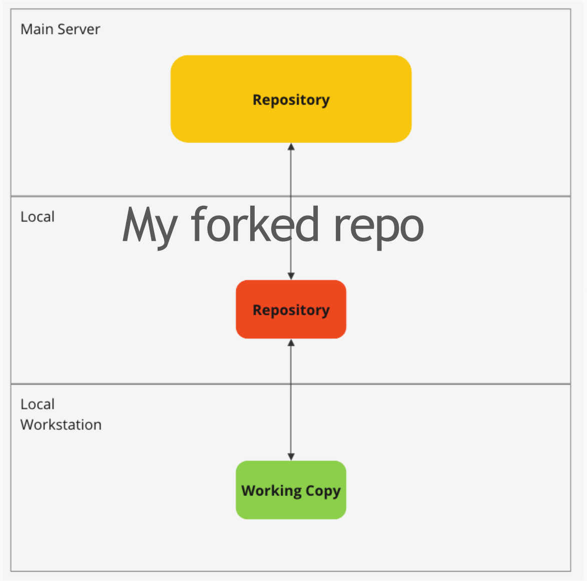
This branch has no conflicts with the base branch
Merging can be performed automatically.

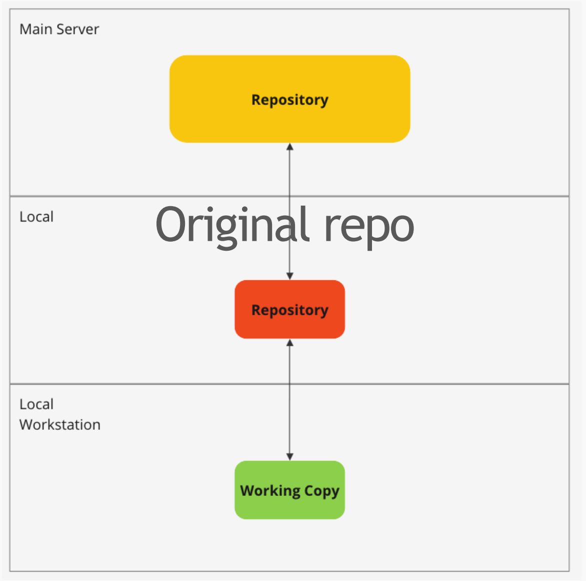
You can also open this in GitHub Desktop or view command line instructions.

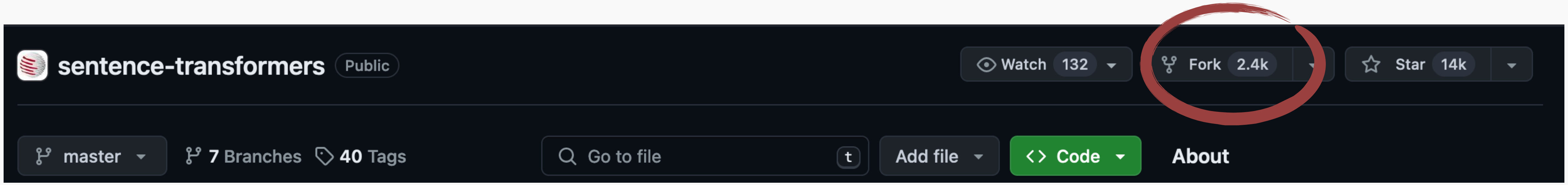
Fork

Forking is creating a personal copy of someone else's repository.

Forking is like making a duplicate of a project to work on independently, without affecting the original.





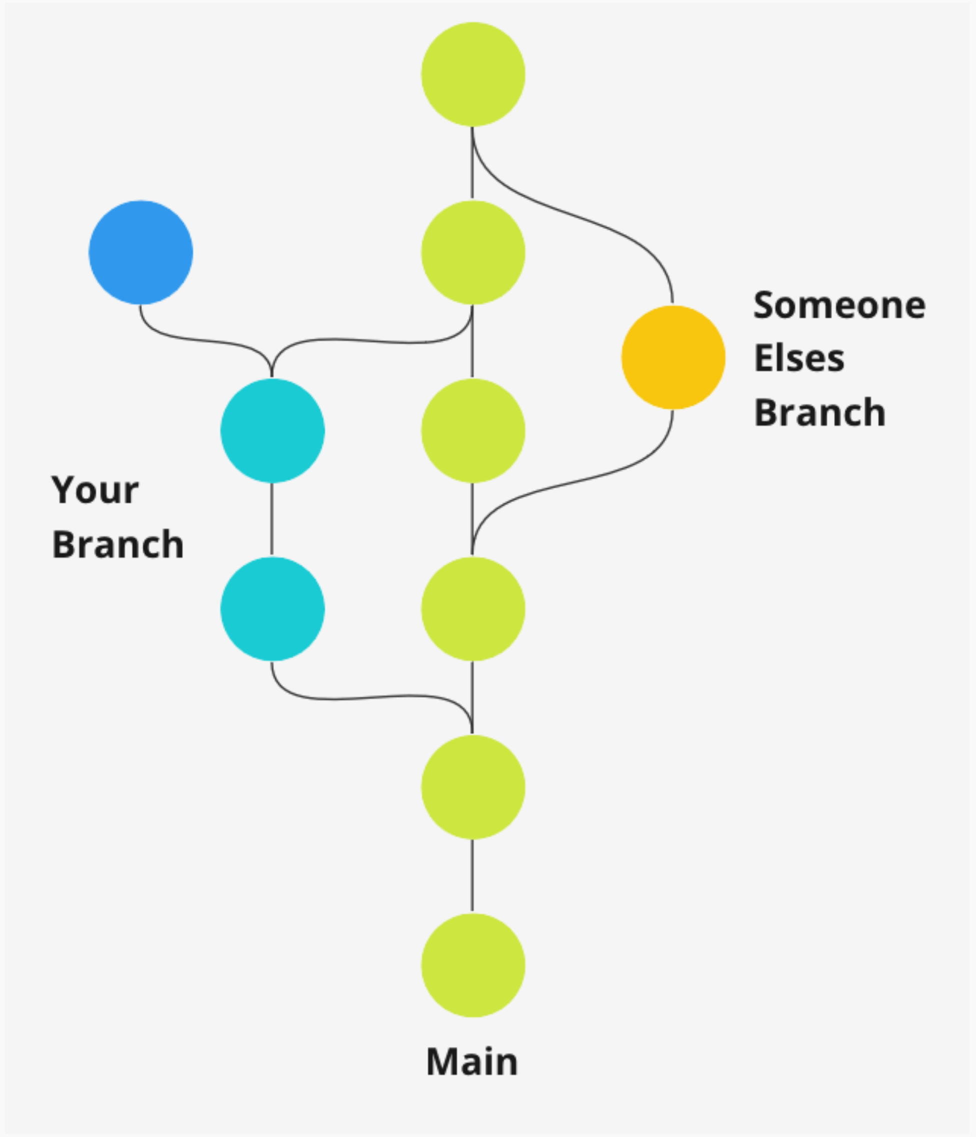


Stash

Stashing is temporarily saving changes that are not ready to be committed, allowing you to switch branches or perform other tasks.

Imagine putting your current work aside on a shelf to deal with later so you can focus on something else.

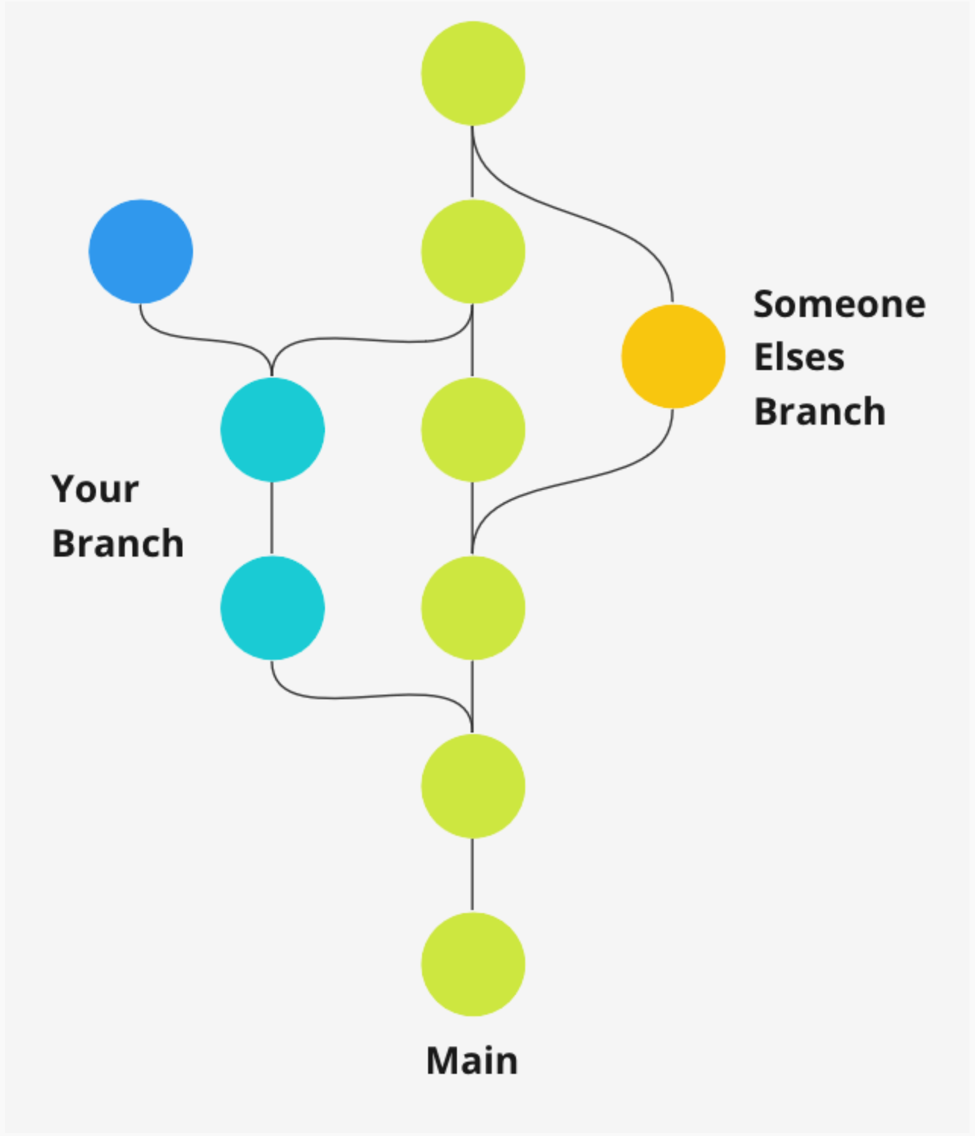
Done through “git stash”
Bring out what you want from the stash through “git stash pop [id]” or “git stash apply [id]” to keep it in stash



Lets **Stash**

- Go into any repo you worked on
- Create changes in any of the files
- Run “git status”. If there is red text then we are good

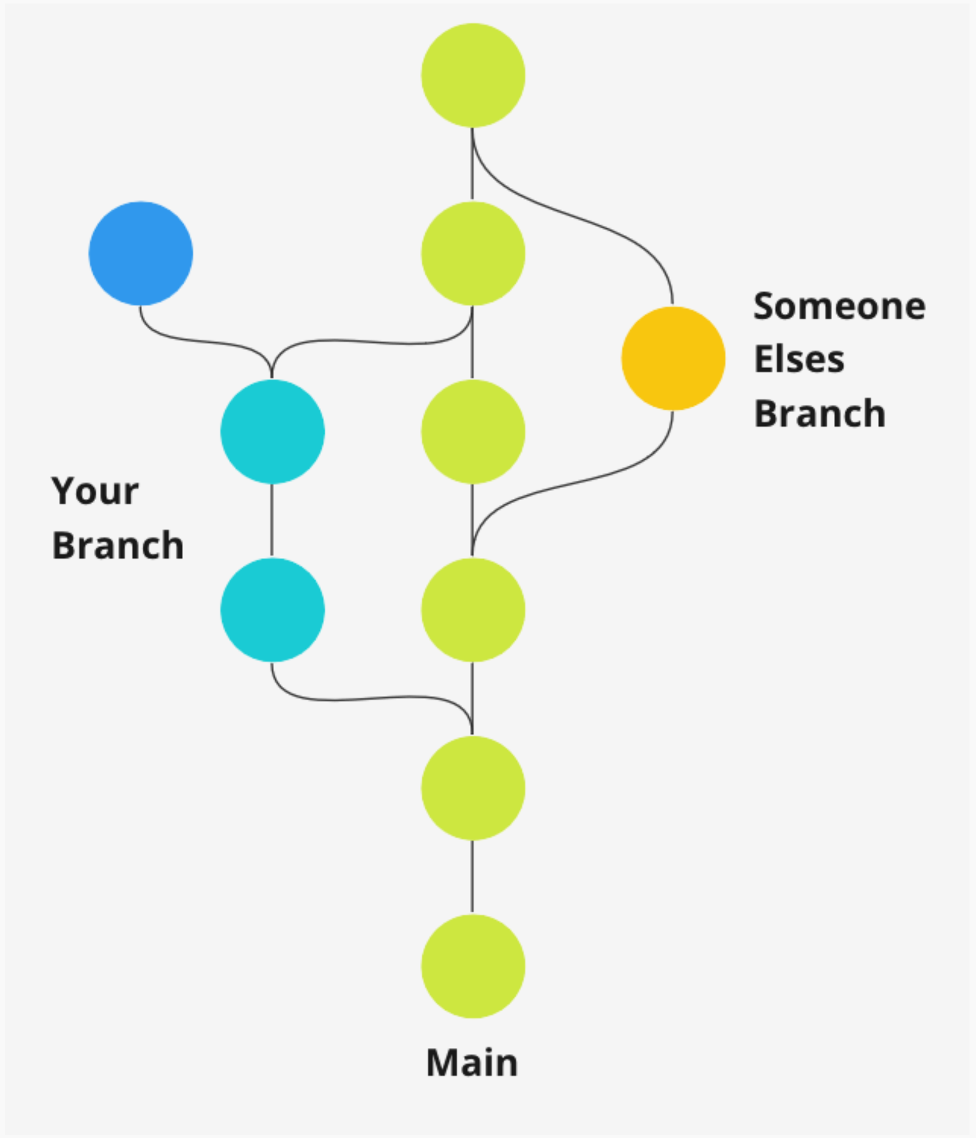
- Run “git stash” to stash it.
- Create changes and stash it again to create multiple stashes
- Run “git stash —list” to see all stashes
- Run “git stash pop 0” to bring out the first stash out.
- Verify through “git status” and “git diff”



Lets **Stash**

- Go into any repo you worked on
- Create changes in any of the files
- Run “git status”. If there is red text then we are good

- Run “git stash” to stash it.
- Create changes and stash it again to create multiple stashes
- Run “git stash —list” to see all stashes
- Run “git stash pop 0” to bring out the first stash out.
- Verify through “git status” and “git diff”



Restore and **Reset**

Restore and Reset reverts changes.

Restore on file level
- Untagged: “git restore <file name>”
- Staged: “git restore —staged <file name>”

Reset on commit level
- Soft reset: Moves the HEAD to the specified commit, but doesn't change the index (staging area) or working directory. The changes will appear as staged.
    - “git reset —soft”
- Mixed reset: Moves the HEAD to the specified commit and resets the index, but not the working directory. The changes will be unstaged.
    - “git reset —mixed”
- Hard reset: Moves the HEAD to the specified commit and resets both the index and working directory. This will discard all changes.
    - “git reset —hard”

Can reset to specific commit by adding commit hash or HEAD~<amount of commit behind>

Restore and **Reset**

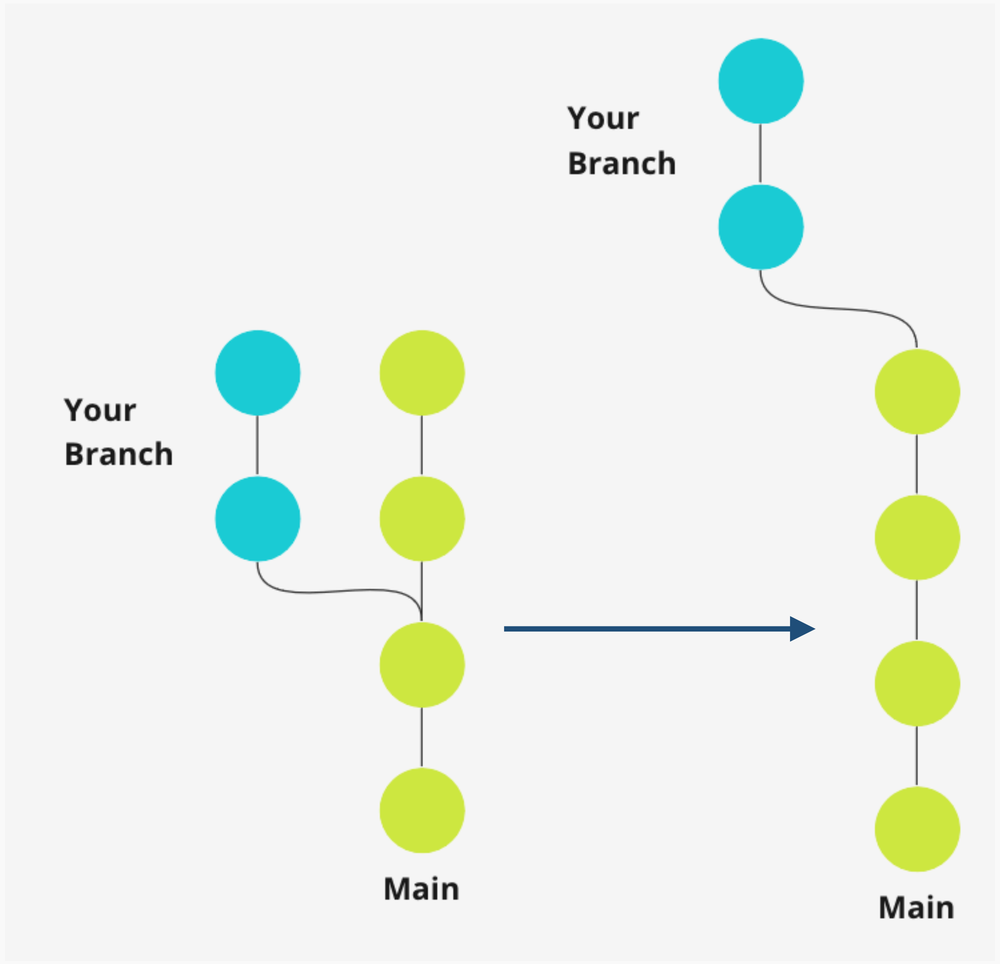
- In the repo you used from stash, try restoring the file through “git restore <file name>

- Apply changes to the file and commit the changes. Do this a few times.
- Try the different reset methods to see the differences
    - git reset —soft HEAD~1
    - git reset —mixed <Specific hash>
    - git reset —hard

Rebase

move or combine a sequence of commits to a new base commit. It is primarily used to maintain a clean and linear project history.

Run with “git rebase <branch-which-you-want-to-move-to>



And thats all the theory!

There are exercises in the course repo

Find the problems you want to solve and run bash on the script to create the scenarios

Try solving all of them.