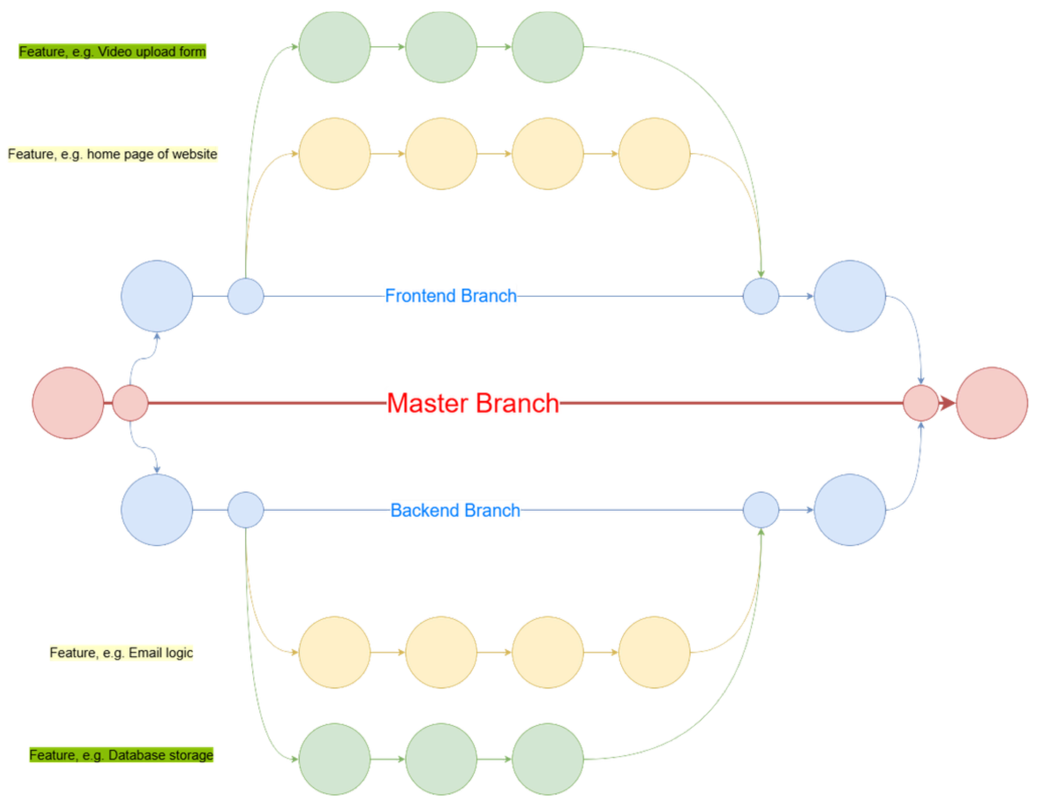
GitHub and Version Control Standards

This project will make use of a simplified implementation of Git Flow. The structure of the development environment for this project is as follows:


Feature, e.g. Video upload form
Feature, e.g. home page of website
Frontend Branch
Master Branch
Backend Branch
Feature, e.g. Email logic
Feature, e.g. Database storage

Git Flow

- The project contains a Master/Main branch.
- The project involves 2 essential sub-branches:
  - Frontend: This branch will be related to all front end development
  - Backend: This branch will be related to all back end development
- Further, Feature branches will be added as needed. Features are never implemented on or straight off of Master, but rather they branch off of the relevant sub-branch (front end or back end). This is to ensure isolation of components and features to ensure bugs if they are to exist do not majorly harm the system.

🚨 DO NOT USE YOUR OWN STYLE FOR A COMMIT MESSAGE. EVERYONE MUST USE THE CONVENTION BELOW. THE STANDARDS ARE TO BE FOLLOWED STRICTLY.

Commit Conventions

Every commit message **must** follow the format below:

-m `"<type>[<scope>]"` -m “ `<description>` “

Type

Can be any of the following:

- `feat` – a new feature is introduced with the changes
- `fix` – a bug fix has occurred
- `chore` – changes that do not relate to a fix or feature and don't modify src or test files (for example updating dependencies)
- `refactor` – refactored code that neither fixes a bug nor adds a feature
- `docs` – updates to documentation such as a the README or other markdown files
- `style` – changes that do not affect the meaning of the code, likely related to code formatting such as white-space, missing semi-colons, and so on. Also used for styling front end elements.
- `test` – including new or correcting previous tests
- `perf` – performance improvements
- `ci` – continuous integration related. This is for any merges between Frontend/Backend with main branch
- `build` – changes that affect the build system or external dependencies
- `merge` – merges two branches together (not with main)
- `branch` - creates a new branch.
    - Has structure `branch[<name of new branch>]: Created <new branch> branch`
- `revert` – reverts a previous commit

Please ensure you use the most relevant one of the above.

Scope

Scope refers to what the commit relates to on a high level, think of it to be somewhat like a heading. For example (notice the capitalization):

- Frontend-Navbar

- Frontend-Video Upload Form
- Backend-Video Upload
- etc

Notice the format. The first scope example means that the feature is on the front end. The last example is on the Backend. This would indicate the respective branch is the one where the change is being developed on.

Description

This is basically what the commit actually does in more detail than the scope lists. Ensure this is long enough to provide sufficient detail to reviewers but not too much detail that would be redundant.

It is worth noting that if a commit is too long then it could have been multiple commits instead of just one.

Example commits (corresponding to the scope examples provided earlier):

- **feat**[Navbar]: Add the navigation bar component including logo and buttons.
- **fix**[Video Upload Form]: Fixed the email field on the form to only accept input that has an email format
- **perf**[Video Upload]: Improved code performance to upload videos of size ~20MB within 10 seconds

More example:

- **refactor**[About Us]: Moved the About Us section code from `App.jsx` file to a separate component file
- **docs**[Video Framing]: Added comments to the `frames.py` file
- **style**[Video Requirements]: Styled the requirements section of the homepage

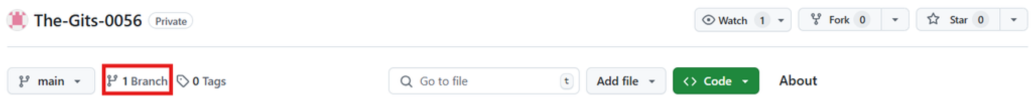
Guide to Developers

If you are helping to develop the system, **you must read the following**.

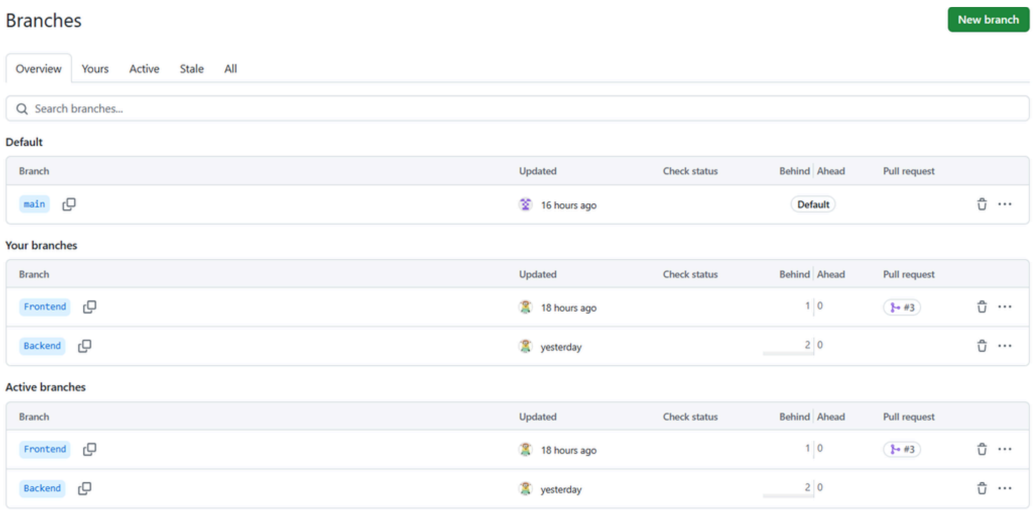
Step 1: Familiarize yourself with GitHub

Visit the repository: asojan03/The-Gits-0056 (github.com)

Click on `Branch`:



View the current branches in the project (There should mainly be Frontend and Backend):


Branches

New branch

Overview Yours Active Stale All

Search branches...

Default

Branch	Updated	Check status	Behind	Ahead	Pull request
main	16 hours ago		Default		

Your branches

Branch	Updated	Check status	Behind	Ahead	Pull request
Frontend	18 hours ago		1	0	#3
Backend	yesterday		2	0	

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
Frontend	18 hours ago		1	0	#3
Backend	yesterday		2	0	

Branches

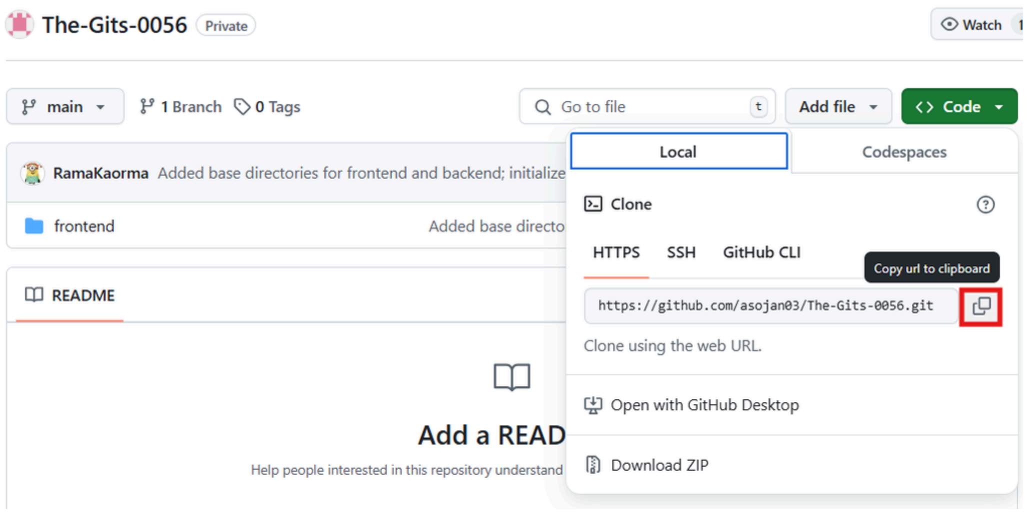
Notice that some branches will be behind each other. *That is okay.* This is because we are keeping the front end and back end **completely isolated** during development phase. Meaning, front end and/or back end will naturally be a few commits behind.

Step 2: Clone the repository onto your local device

The Actual Repo

Visit the repository: asojan03/The-Gits-0056 (github.com)

Click on `Code`. Then, click on the ⧉ icon to copy the HTTPS link



RamaKaorma Added base directories for frontend and backend; initialize

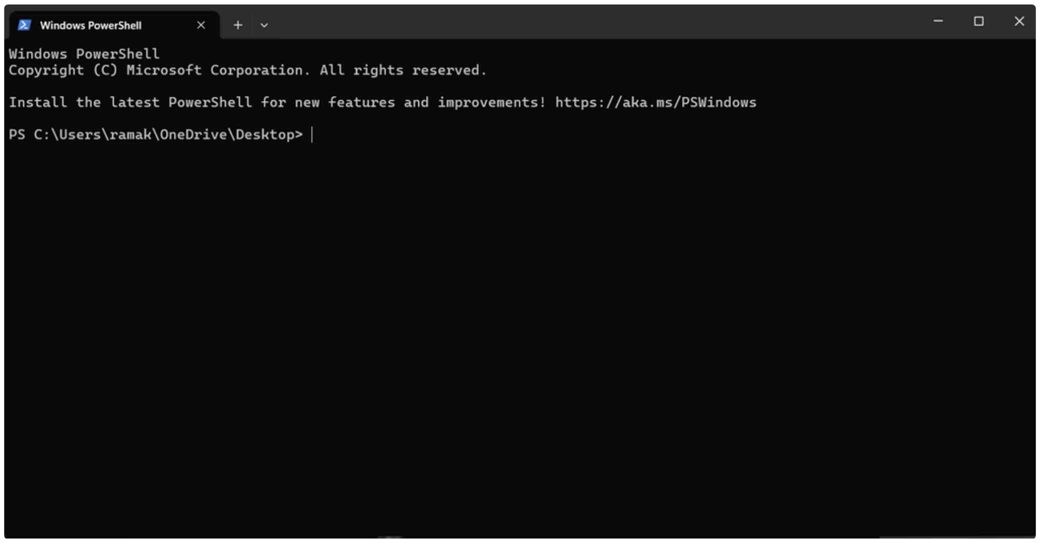
```
https://github.com/asojan03/The-Gits-0056.git
```

Clone using the web URL.

Help people interested in this repository understand

Copy link to repository

Go to your local directory where you want the project folder to sit. I want it to sit on my desktop, so I open the terminal and navigate to my desktop directory.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\ramak\OneDrive\Desktop>
```

Note that from here, the Git commands will be on a Windows system. If you are using a Mac or another system, the commands may differ slightly. It is **your responsibility** to check what the commands for your system should be.

Clone the repository using the HTTPS link you copied from GitHub earlier:

```
git clone https://github.com/asojan03/The-Gits-0056.git
```

The Branch

Now that you have the repository, you need to clone the Frontend and Backend branches from remote to be able to work on them.

Note that when you clone a repository, you create a copy on your local machine and this copy tracks changes on the `main` remote branch. If you want to work on another branch, you need to track changes in that particular branch. We will do that now.

Check what the remote branches are:

```
git branch -a
>>* main
    remotes/origin/Backend
    remotes/origin/Frontend
    remotes/origin/HEAD -> origin/main
    remotes/origin/main
```

Pulling a remote branch for the first time

Execute the following 2 commands (one at a time) to:

1. Fetch remote changes
2. Create a local copy of the remote branch that tracks changes in that same remote branch. **Make sure the first instance of <branch-name> matches the actual name of the remote branch so you do not get confused**.

```
git fetch origin
git checkout -b <branch-name> origin/<branch-name>
```

(If you happen to mess this up e.g. by using the wrong branch name, use the following command to *REMOVE* the local copy of the remote branch):

`git branch -D <branch-name>`

Then you can do fetch & checkout again (correctly).

Now, check which branches you are tracking (each branch has to have the previous step done to it):

```
git branch
>>* main
    Frontend
    Backend
    ...
```

Now, whenever you need to make changes to work on your local (remote-tracking) branch, ensure you pull any changes just as you would do with the main

```
git fetch origin <branch-name>
```

Step 3: Create your feature branch

Ask yourself these questions:

1. What feature am I trying to develop?
2. Is this feature a frontend feature or a backend feature?
3. Is this feature big enough to have its own branch, or is it an extension of a pre-existing feature that already has a branch setup?

Based on your answers to the previous questions, you should make a decision regarding:

- Whether your feature branches directly from frontend/backend
- Whether your feature branches of another sub-branch of frontend/backend

Now, consider this example of implementing the navigation bar feature. The navigation bar will be visible on every page of the website, hence is a universal feature of frontend. This implies that it should have its own feature branch.

Create the `Navbar` branch

```
git checkout -b Navbar Frontend
```

Explanation:

- `git` : Indicates that this is a git command
- `checkout` : Switches us to a branch
- `-b` : Flag to create a new branch
- `Navbar` : The name of the new branch
- `Frontend` : Indicates that Navbar is made off of Frontend. What this basically means is the initial state of Navbar is whatever Frontend was when Navbar was created. It is a clone of Frontend, hence whatever the other branches contain will not affect Navbar.

Start Development

Now that you have created a new branch, you must as a safety measure check that you are on your feature branch, before you start developing.

```
git branch
>>* Navbar
    main
    backend
    ....
```

If you see an asterisk (*) next to your feature branch, it means you are now on this branch and any changes you make and commit will only happen on this branch.

Now write all the code that you want to write. Test regularly and ensure it works as intended.

Commit changes to feature branch

You are now done with development and your code should be added to the main project.

After ensuring you are still on your feature branch (like we did in the previous step), perform the following:

Stage

```
git add .
```

Commit

```
git commit -m "feature[Frontend-Navbar]" -m "Addded navigation bar"
```

Note the format of the commit message. It follows from the commit rules and conventions listed above. Notice the double -m:

- The first one is for the heading of the commit,
- The second one is for the description of the commit

Merge

Switch the parent branch. In this example, that would be `Frontend` branch as we originally branched straight off of that.

```
git checkout Frontend
```

And then merge back into parent with this command

```
git merge --no-ff Navbar -m "merge[Navbar]" -m "Merged Navbar into Frontend"
```

Remember to include the `--no-ff` command. This will ensure that an explicit merge commit is created and added to the commit history and is visible to us when we look back and can then see when the merges were done.

Remember to include the -m command followed by a quote as the commit description.

Also note that the commit must always be of the form:

`merge[<changed branch>]: Merged <changed branch> into <parent branch>`

Now, if you go to view your parent branch (in this case Frontend), you will see your old code, plus the Navbar!

Push

Now that you've made a change and committed it, you need to ensure that you push the new branch itself and its parent

```
git push origin Frontend
```

and then

```
git push origin Navbar
```

Congratulations, you have made a successful change, and it's all logged on GitHub!

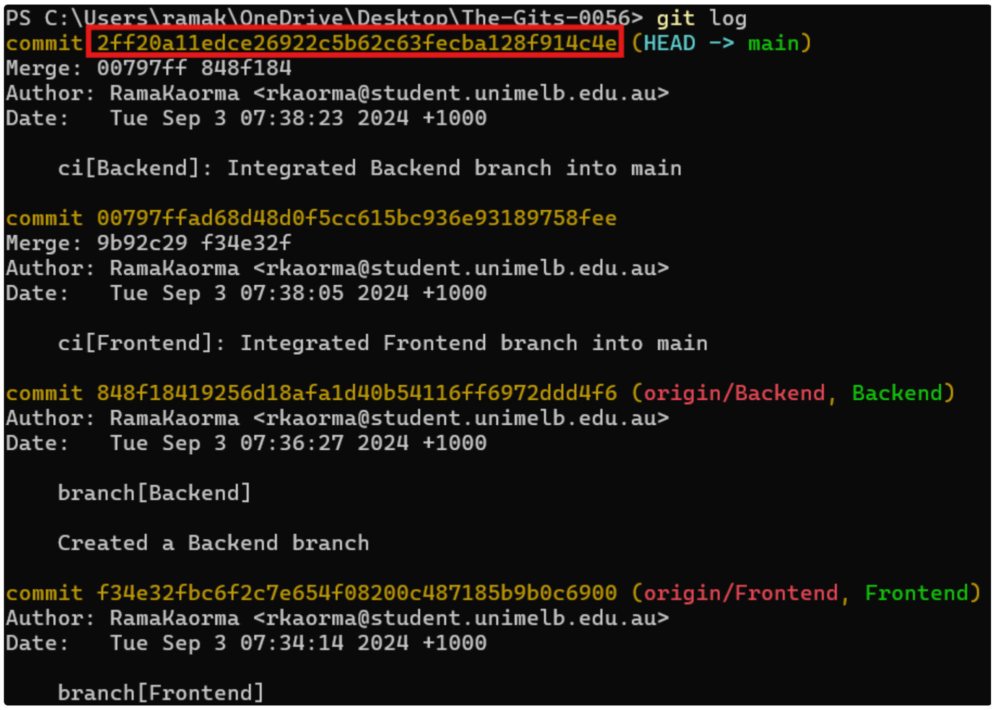
What if I messed up a commit message?

Revert

Check your commit history

```
git log
```

You will see an output similar to this



```
PS C:\Users\ramak\OneDrive\Desktop\The-Gits-0056> git log
commit 2ff20a11edce26922c5b62c63fecba128f914c4e (HEAD -> main)
Merge: 00797ff 848f184
Author: RamaKaorma <rkaorma@student.unimelb.edu.au>
Date:   Tue Sep 3 07:38:23 2024 +1000

    ci[Backend]: Integrated Backend branch into main

commit 00797ffad68d48d0f5cc615bc936e93189758fee
Merge: 9b92c29 f34e32f
Author: RamaKaorma <rkaorma@student.unimelb.edu.au>
Date:   Tue Sep 3 07:38:05 2024 +1000

    ci[Frontend]: Integrated Frontend branch into main

commit 848f18419256d18afa1d40b54116ff6972ddd4f6 (origin/Backend, Backend)
Author: RamaKaorma <rkaorma@student.unimelb.edu.au>
Date:   Tue Sep 3 07:36:27 2024 +1000

    branch[Backend]

    Created a Backend branch

commit f34e32fbc6f2c7e654f08200c487185b9b0c6900 (origin/Frontend, Frontend)
Author: RamaKaorma <rkaorma@student.unimelb.edu.au>
Date:   Tue Sep 3 07:34:14 2024 +1000

    branch[Frontend]
```

git log output

Find the commit that you want to go back to (This will likely be the latest one, that is the topmost). Reset to that commit

```
git reset --hard <commit_before_merge>
```

Now repeat the staging and committing process from before. Make sure your commit message follows the standards!

If you push is failing, and you are getting an error such as:

```
RPC failed; HTTP 400 curl 22 The requested URL returned error: 400
send-pack: unexpected disconnect while reading sideband packet
```

It may be true, that you are uploading a large file, perhaps an AI model.

In which case, try running the following command:

```
git config --global http.postBuffer 157286400
```

This helps your network process the large file.

After running the above command, you can **re-run the git push command**.

Best of luck!

Help Resources

Git pull remote branches (graphite.dev)

Delete Commit from a branch (GitHub)