

Practical

Contents

Practical	1
Table of contents	1
Setup	1
First repository	1
First commit	4
Fork and pull request workflow	7
Merge conflicts	12
Conclusion	14

Practical

This document contains all the practical information (from chapters 1 to 5) condensed into a single document - stripped of most explanations.

While this is a useful document for a quick glance, it is strongly advised that you read the referenced chapters per practical to ensure a proper understanding of the Git version control system.

Table of contents

1. Setup
2. First repository
3. First commit
4. Fork and pull request workflow
5. Merge conflicts

Setup

Setup Git and GitHub to begin following this guide. Original chapter [here](#).

1. Install Git on your machine. Instructions [here](#).

Note* If you are using Windows, the Git installer will likely have installed something called Git Bash. For the remainder of this practical, this will be your goto terminal. While command prompt may be configured for use, the guide includes the use of bash commands that is only available on Git Bash. For more information about the basics on bash commands, refer to this chapter.

2. Create a new GitHub account. Instructions [here](#).
3. Configure Git on your local machine. The credentials used should match the credentials used for GitHub.

```
git config --global user.name "<Your name>"
git config --global user.email "<Your email used when setting up GitHub>"
```

First repository

Next, you will be creating your first repository. While there are two ways to do so (as discussed in [chapter 2](#)), this practical uses the “GitHub first” approach.

Note* that the terms “repository” and “project” are used interchangeably in the practicals.

Note* that the term “terminal” is synonymous with Git Bash or any UNIX-based terminal that uses bash.

1. Login to GitHub
2. Expand drop-down menu in the top-right corner of the menu bar
3. Select the “New Repository” option

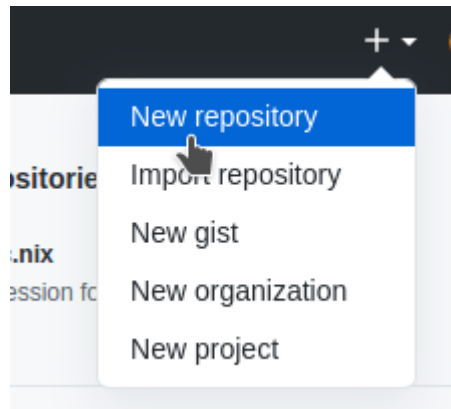


Figure 1: New repository option

4. Configure and create the repository, creating a remote repository

For now, only change the project name to “learning-git” and the project visibility to “Private”. Leave everything else as it is. For more information about the various options, refer to chapter 2.

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 *



woojiahao ▾

/

learning-git



Great repository names are short and memorable. Need inspiration? How about **jubilant-goggles?**

Description (optional)



Public

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



Private

You choose who can see and commit to this repository.

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



Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

Figure 2: New repository details

5. Clone the repository to create a local repository

For this practical, we will store all project folders inside a **Projects/** folder in the user directory of our local machine.

```
cd ~/Projects/
```

You will notice a repository link provided by GitHub. Copy that to your clipboard and use it when cloning the repository.

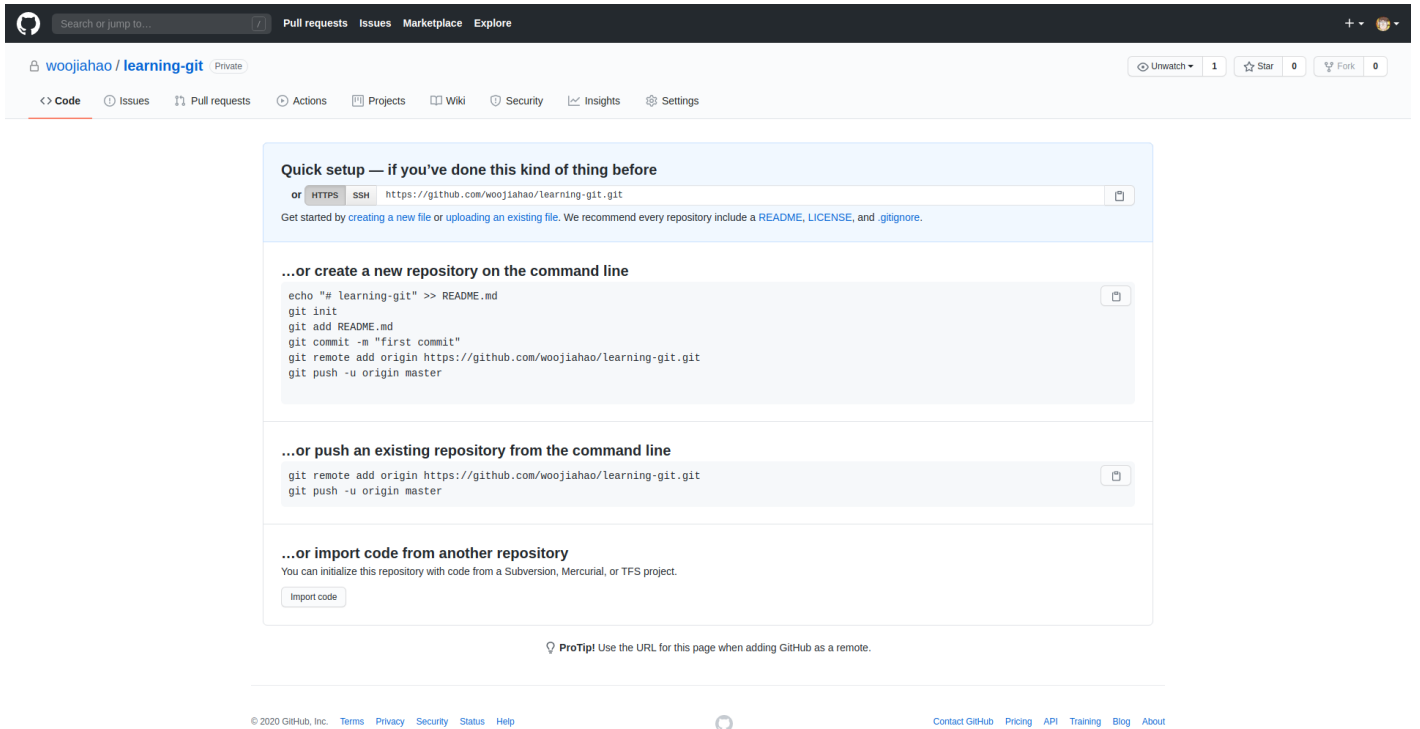


Figure 3: Repository URL

```
git clone https://github.com/woojiahao/learning-git.git
```

You will be prompted for your GitHub credentials if it is a private repository.

To verify that the cloning is successful, use `ls` and you will find that a new folder called `learning-git` is now in the `Projects/` folder.

6. Navigate to the local repository

```
cd learning-git/
```

To verify that you are in your local repository, you can use `pwd`. The current folder should be `learning-git/` now.

```
λ chill [~/Projects/learning-git] at  master ✓
→ pwd
/home/chill/Projects/learning-git
```

Figure 4: `pwd`

First commit

After creating the repository, you can start committing changes to your local repository and pushing them to your remote repository.

This chapter relies heavily on a broad understanding of the staging area (and the related locations a file can exist). These concepts are further explained in [chapter 3](#).

This practical picks up right where the previous one ended.

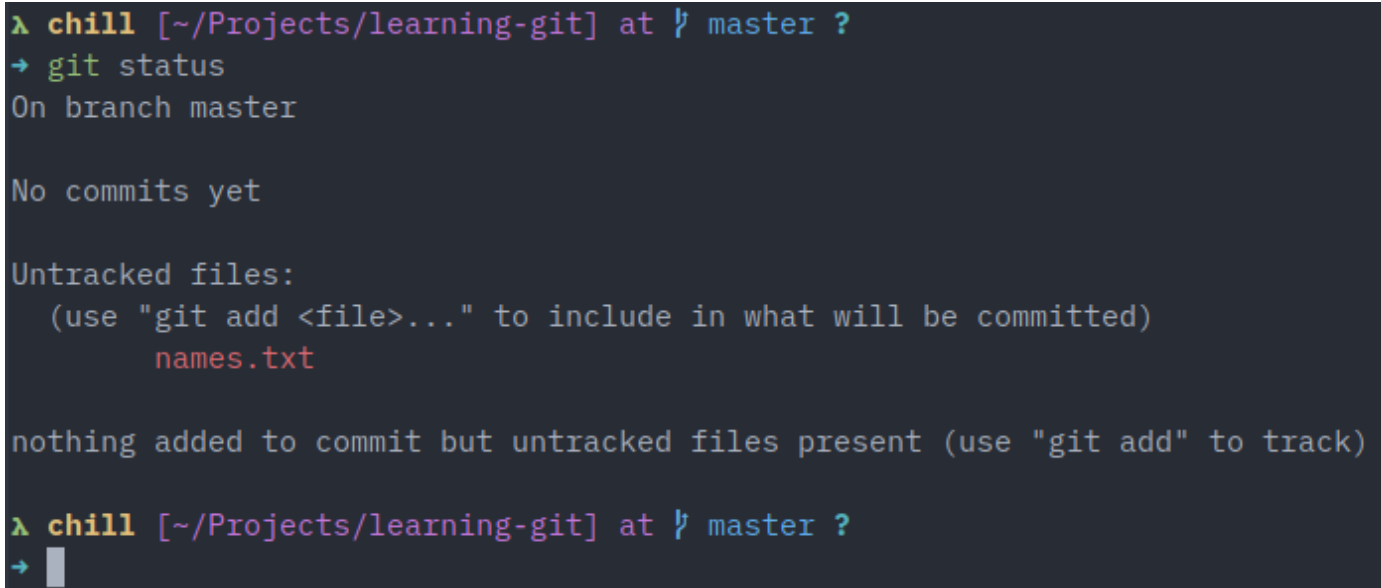
1. Create a new file in the root of the project folder and add your name to the first line of the file

This can be done using the file explorer or through the terminal.

```
echo "Woo Jia Hao" > names.txt
```

2. View the current status of the working directory

```
git status
```



```
λ chill [~/Projects/learning-git] at ʘ master ?
→ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

λ chill [~/Projects/learning-git] at ʘ master ?
→
```

Figure 5: git status new file

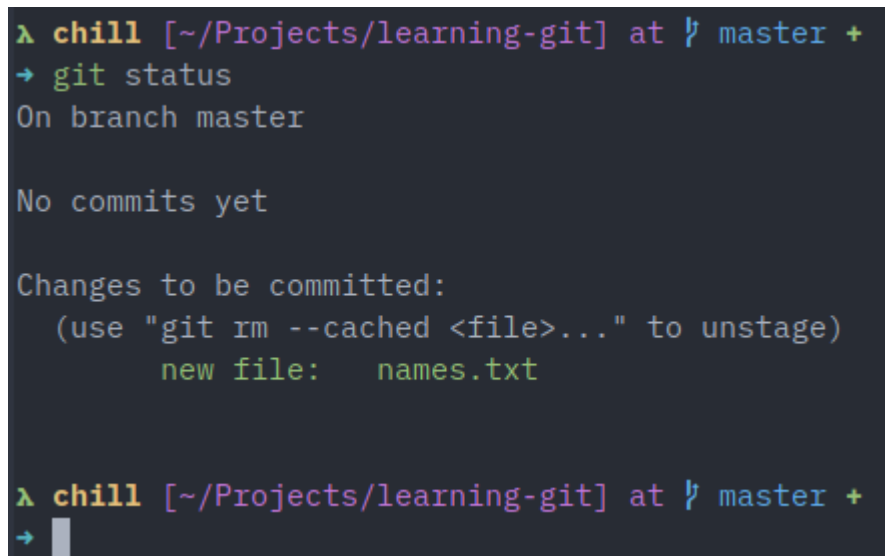
The new file should be in red under the “Untracked files” section as this is a new file that has yet to be staged once.

3. Add `names.txt` to the staging area to track it

```
git add names.txt
```

4. Verify that the file has been added using `git status` once again

The file should now be in green under the “Changes to be committed” section.



```
λ chill [~/Projects/learning-git] at ʘ master +
→ git status
On branch master

No commits yet

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

λ chill [~/Projects/learning-git] at ʘ master +
→
```

Figure 6: git status add file

5. Commit `names.txt`, confirming that you wish to keep the changes made (in this case, creating a new file and adding your name to it)

```
git commit -m "Add names.txt"
```

The text that follows `-m` is the commit message. It should be a description of the changes made in the commits.

6. Verify that the file has been committed using `git status`

```
λ chill [~/Projects/learning-git] at ʘ master ✓
→ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean

λ chill [~/Projects/learning-git] at ʘ master ✓
→
```

Figure 7: git status committed file

The file will now no longer be shown.

7. Push this commit to the remote repository to upload these changes to the remote repository

`git push origin master`

```
λ chill [~/Projects/learning-git] at ʘ master ✓
→ git push origin master
Username for 'https://github.com': woojiahao
Password for 'https://woojiahao@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 226 bytes | 226.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/woojiahao/learning-git.git
 * [new branch]      master -> master
I
λ chill [~/Projects/learning-git] at ʘ master ✓
→
```

Figure 8: git push

You may have to enter your GitHub credentials.

8. You can verify that the changes have been pushed to GitHub on GitHub under the repository page

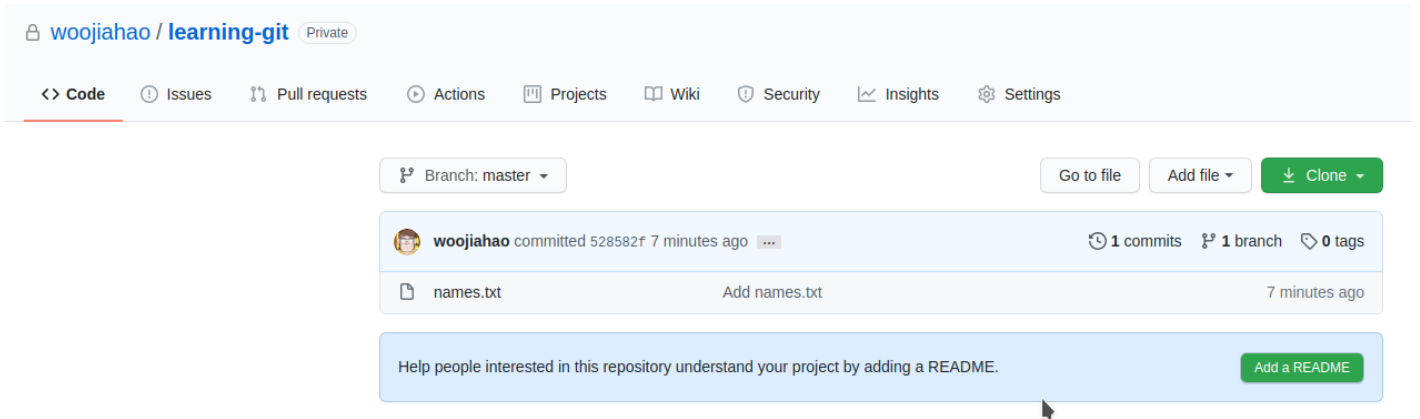


Figure 9: File added to GitHub

There will be a new file added and you can view it in the built-in file browser for GitHub.

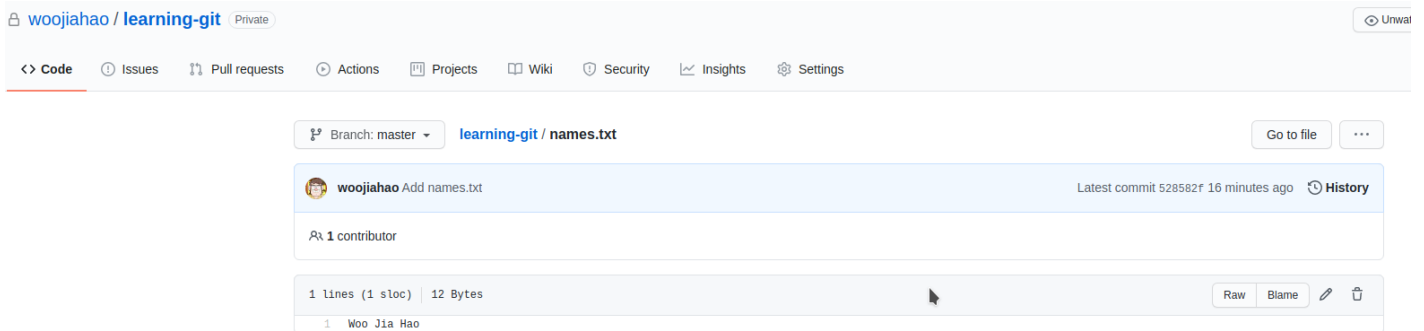


Figure 10: names.txt file content

Fork and pull request workflow

Now that you know how to push changes on your own. Let's explore how you can collaborate with others and share code among a team for group projects.

For this practical, appoint someone in your group to be the owner of the practice repository. The owner will then proceed to create a new repository, following the steps discussed in the previous chapters (including the first commit).

Since the new repository is a private repository, the owner of the repository must add your team members as collaborators for them to have access to it (refer [here](#)).

Note* The following screenshots will have the repository name as **learning-git**. However, you will have to replace the name of the repository with whatever the owner sets the repository name to be.

All members (excluding the owner) are to perform the following steps first.

1. Fork the repository

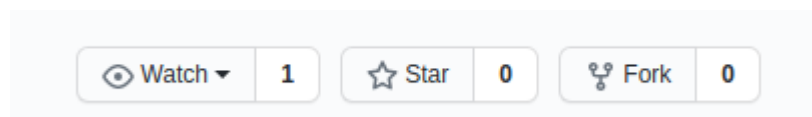


Figure 11: Fork option

GitHub should begin creating a remote copy for your own account. Once done, you should see a page like this.

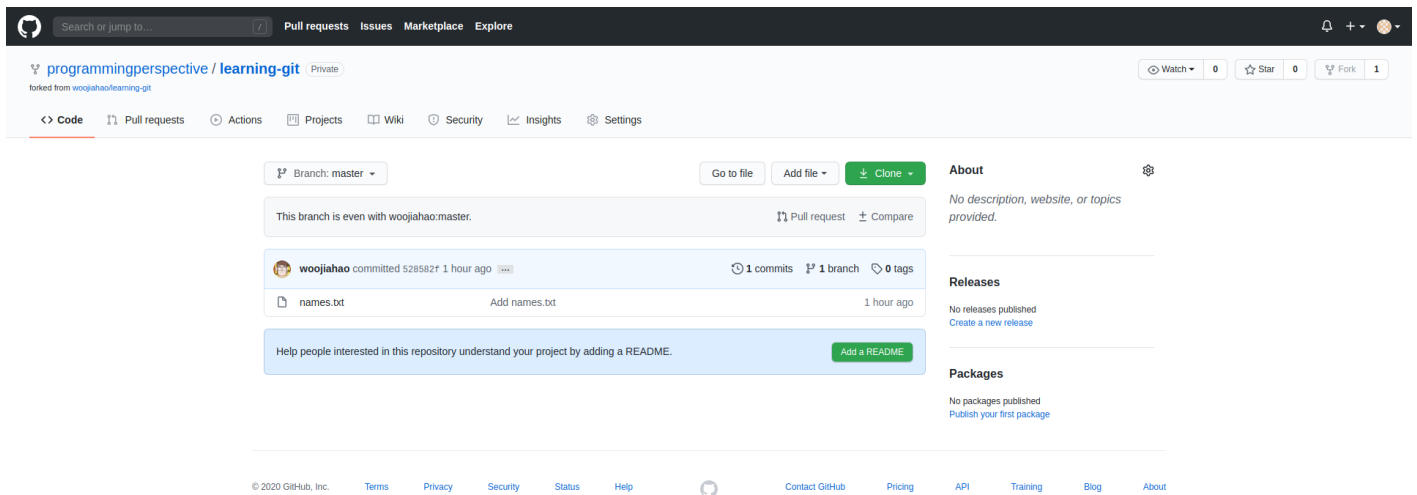


Figure 12: Forked repository

2. Clone the repository by clicking on the green “Clone” button and using the repository URL
3. Navigate to the file in the file explorer or terminal
4. Create an **upstream** remote to the original repository

```
git remote add upstream <original repository URL>
```

```
λ chill [Projects/git-guide-fork/learning-git] at ♣ master ✓
→ git remote add upstream https://github.com/woojiahao/learning-git.git

λ chill [Projects/git-guide-fork/learning-git] at ♣ master ✓
→
```

Figure 13: git remote add

The repository URL is the same URL that you used to clone a repository. However, this time, it can be found in the original repository.

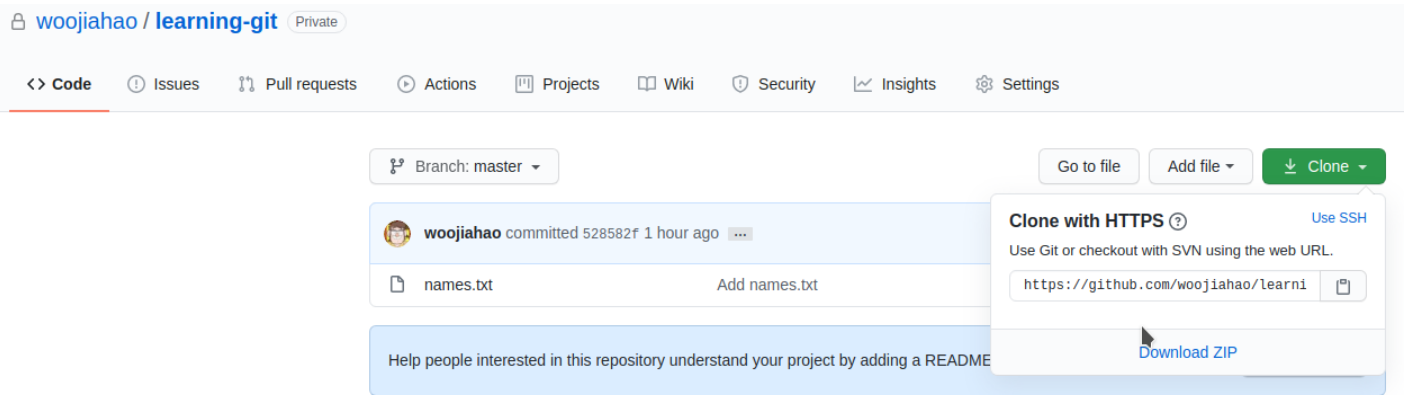


Figure 14: Original repository URL

5. Use `git remote -v` to view both the alias of the remote and the URL of the matching repository

```

λ chill [Projects/git-guide-fork/learning-git] at  master ✓
→ git remote -v
origin  https://github.com/programmingperspective/learning-git.git (fetch)
origin  https://github.com/programmingperspective/learning-git.git (push)
upstream      https://github.com/woojiahao/learning-git.git (fetch)
upstream      https://github.com/woojiahao/learning-git.git (push)

λ chill [Projects/git-guide-fork/learning-git] at  master ✓
→

```

Figure 15: View remotes

Ensure that you have both an `origin` and `upstream` remote.

Note* the repository URL for the `origin` and `upstream` remote must be different. The `origin` remote must point to your forked repository while the `upstream` remote must point to the original repository.

The following steps must be carried out by each member one after another. The owner of the repository must accept the pull request from each member before the next member can proceed.

1. Pull the latest changes of the repository

```
git pull upstream master
```

2. Add your name to the file `names.txt`

This can be done in either a terminal or text editor.

```
echo "Andrew Ng" >> names.txt
```

3. Commit this change and push it to your local repository

```
git add names.txt
git commit -m "Add name (Andrew Ng) to names.txt"
git push origin master
```

4. Create a pull request in Github

When you open your remote repository, you can create a pull request under the “Pull requests” tab.

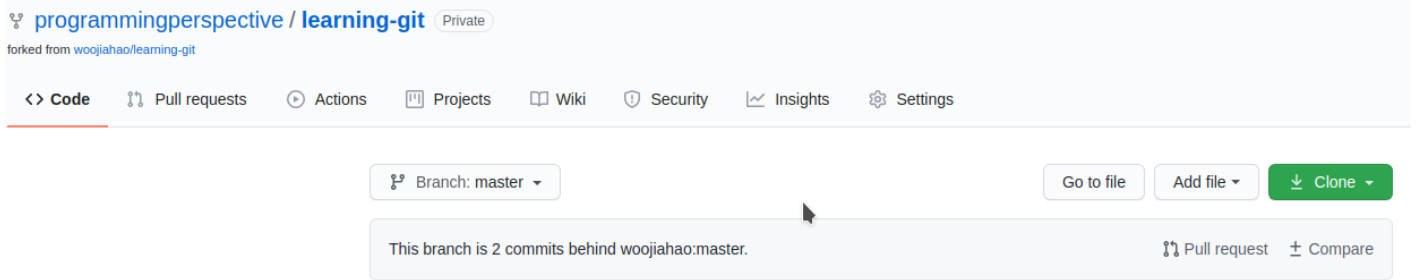
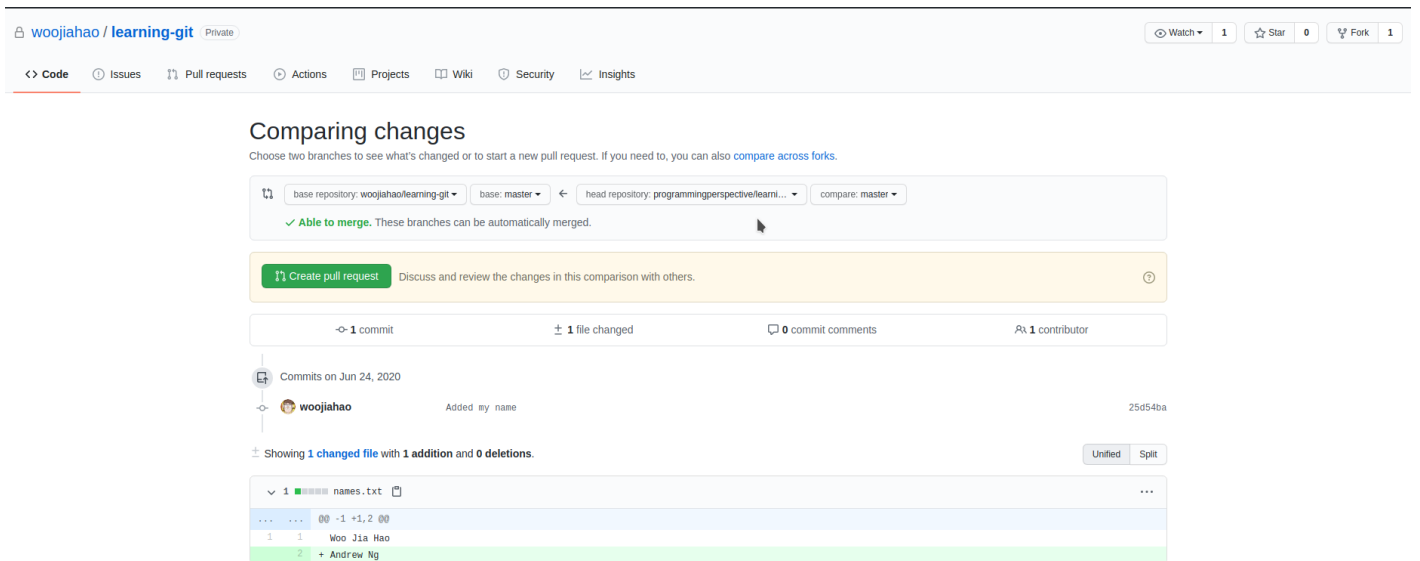


Figure 16: Create PR button

Confirm that you wish to create a pull request and you can leave the details of the pull request as it is (note that the title is always mandatory while the other fields are optional). More about pull requests is explored in [chapter 4](#).



No commit comments for this range

Figure 17: Create PR page

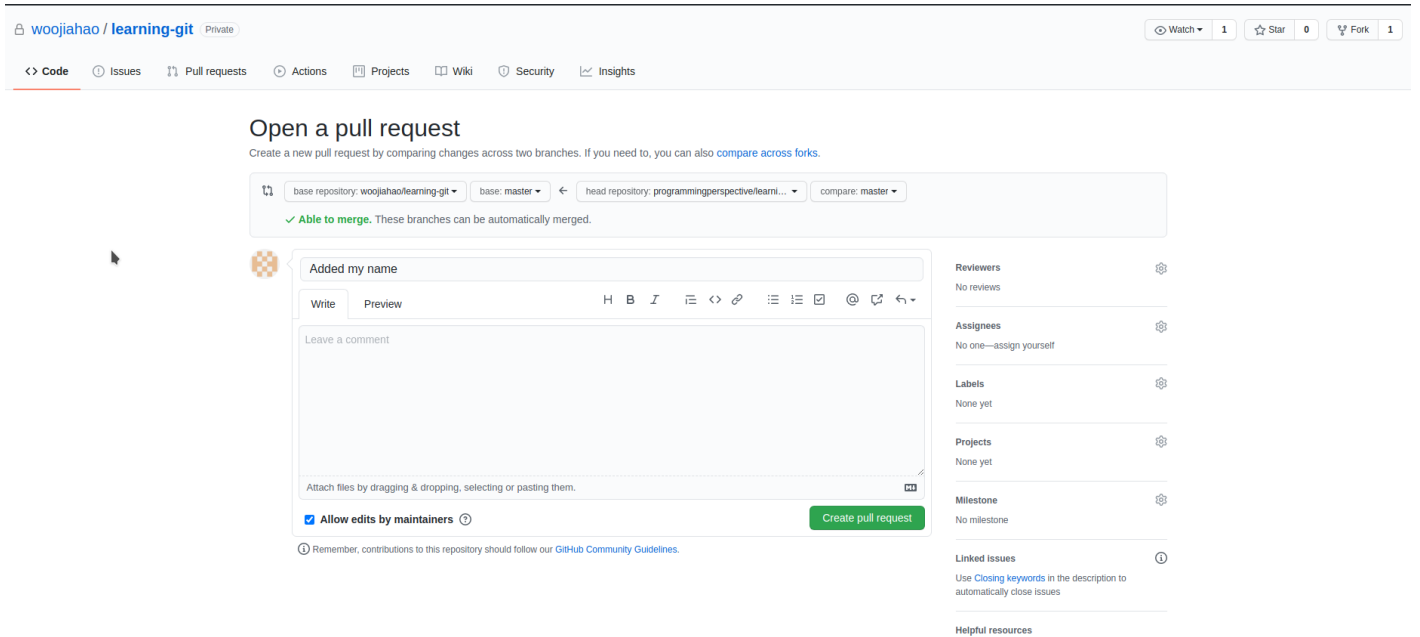


Figure 18: PR details

Once the details of the pull request has been confirmed, you can select the “Create pull request” button.

Now that the pull request has been created, the owner can view it under the “Pull Requests” tab. The owner must accept the pull request which will merge the member’s changes into the original repository. When in the “Pull Requests” tab, they will be able to view all pending pull requests in a list. For this practical, select the only pull request shown, so select that pull request and merge it but clicking on the “Merge pull request” button.

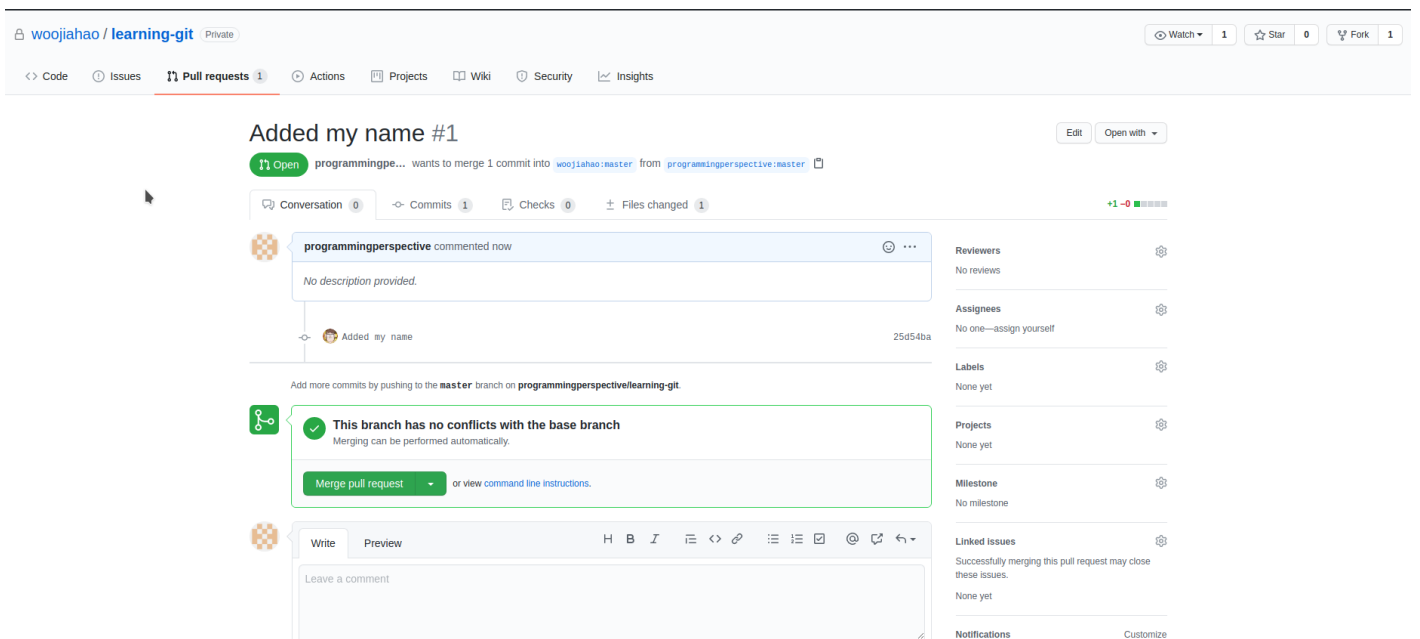


Figure 19: View PR

Now, if you view `names.txt` in the original repository in GitHub, you will be able to see the new name.

Once the owner has accepted the pull request, the next member can proceed. Repeat the steps above, ensuring that the latest changes are pulled from **upstream** every time.

After one round with all members, the original repository should have all the team member's names in `names.txt`. Repeat this process a couple more rounds to properly understand the commands. You can add any text.

Note* The owner can pull the latest changes directly from the `origin` remote as they own the repository already. There is no need to setup the `upstream`.

```
git pull origin master
```

Merge conflicts

When working with others, you may encounter merge conflicts. These often happen when the same line of a file is modified by two sources and these sources are attempting to merge with one another.

As explained in [chapter 5](#), we can simulate this by having the owner update and push a change to `names.txt` while the same line is also modified by another member without pulling the latest changes by the owner.

The following exercise should be conducted between the owner of the repository and another member, one at a time.

1. The owner must pull the latest changes from their remote repository

```
git pull origin master
```

2. The owner will modify the first line of `names.txt` and change it to any text they want
3. The owner will commit and push this change to the original repository

```
git add names.txt
git commit -m "Change first line to favourite color"
git push origin master
```

4. The member will make a modification (different from the owner) to the first line and commit it

```
git add names.txt
git commit -m "Add my favorite color"
```

5. The member will pull the latest changes from the original repository, causing a merge conflict

```
git pull upstream master
```

```

λ chill [Projects/git-guide-fork/learning-git] at ʘ master ✓ ^
→ git pull upstream master
warning: Pulling without specifying how to reconcile divergent branches is
discouraged. You can squelch this message by running one of the following
commands sometime before your next pull:

    git config pull.rebase false  # merge (the default strategy)
    git config pull.rebase true   # rebase
    git config pull.ff only       # fast-forward only

You can replace "git config" with "git config --global" to set a default
preference for all repositories. You can also pass --rebase, --no-rebase,
or --ff-only on the command line to override the configured default per
invocation.

Username for 'https://github.com': programmingperspective
Password for 'https://programmingperspective@github.com':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 250 bytes | 250.00 KiB/s, done.
From https://github.com/woojiahao/learning-git
 * branch                master      -> FETCH_HEAD
   f1bd871..fc844c1      master      -> upstream/master
Auto-merging names.txt
CONFLICT (content): Merge conflict in names.txt
Automatic merge failed; fix conflicts and then commit the result.

λ chill [Projects/git-guide-fork/learning-git] at ʘ master ^#
→ █

```

Figure 20: git pull merge conflict warning

6. `git status` can be used to view the status of the merge conflict

```
git status
```

```

λ chill [Projects/git-guide-fork/learning-git] at 1 master ^#
→ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

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:   names.txt

no changes added to commit (use "git add" and/or "git commit -a")

```

Figure 21: git status merge conflict

7. Open `names.txt` with a text editor, there will be a unique notation marking the area where the merge conflict is present

This notation is further elaborated on in chapter 5 but this is the gist of what it describes

```

<<<<<<< HEAD (Member A's changes)
Red
=====
Blue
>>>>>>> fc844c137e0f6b7ead645f11c0b24f08c51b5202 (Owner's changes)
Andrew Ng

```

8. The member will remove their change and only keep the text “Blue” from the owner

```

Blue
Andrew Ng

```

9. The member will add and commit the resolution of the merge conflict

```

git add names.txt
git commit -m "Fix merge conflict - choose owner edit"
git push origin master

```

Once this round is performed, another member can try to simulate a merge conflict with the original repository.

Conclusion

This practical goes over a very brief version of the exercises described in each chapter. This practical is designed as a quick reference to understand the applications of Git.

It is highly recommended that you reference the formal chapters in this guide to gain a full understanding of the concepts described in this practical.

For more details on each topic, you can visit the GitHub repository for this guide [here](#). More chapters and erratas will be updated there.