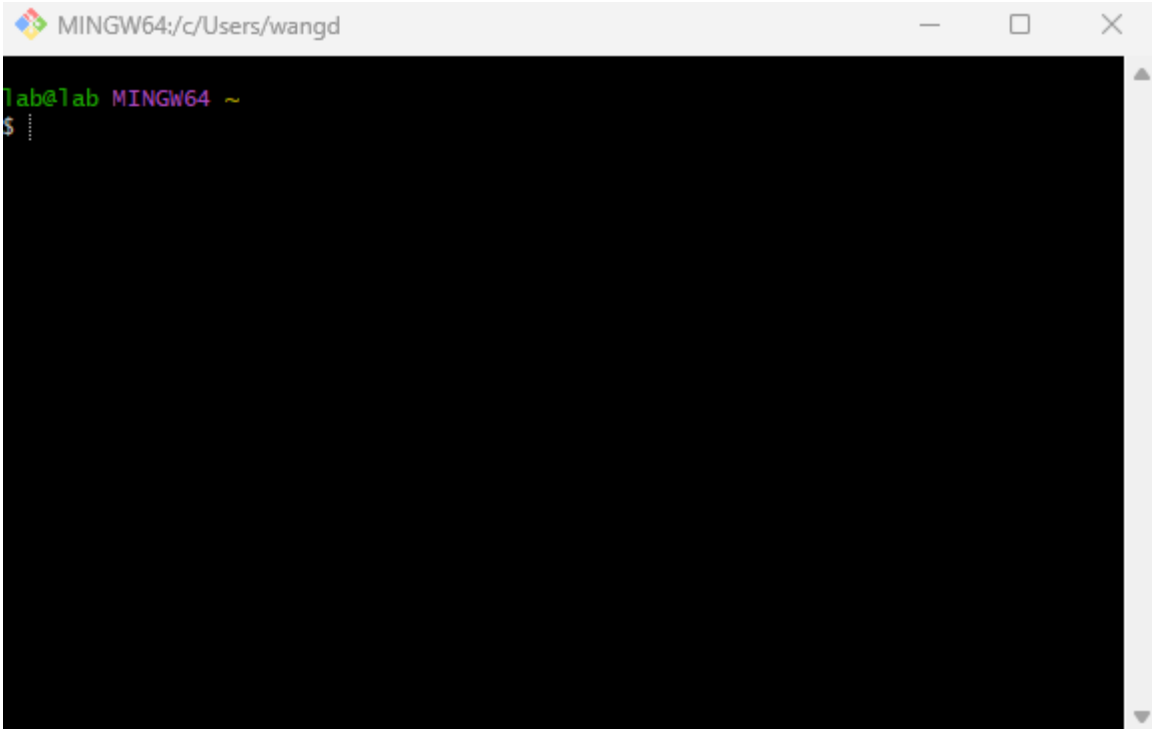
[CS304] Tutorial 4 - Introduction of Git

Using Git

This lab tutorial is about how to use Git to manage our projects.

Open a terminal

After successfully installing git from https://git-scm.com, you should be able to start a "git bash" from your start up menu if you are using windows:



If you are using linux then you only need to open a terminal.

Create a project

Now create a project with `git init`:

```
git init
```

After that the ".git" directory should be created:

```
MINGW64:/c/Users/wangd/Desktop/hello

lab@lab MINGW64 ~
$ cd Desktop && mkdir hello && cd hello

lab@lab MINGW64 ~/Desktop/hello
$ git init
Initialized empty Git repository in C:/Users/wangd/Desktop/hello/.git/

lab@lab MINGW64 ~/Desktop/hello (master)
$ ls

lab@lab MINGW64 ~/Desktop/hello (master)
$ ls -al
total 8
drwxr-xr-x 1 lab 197121 0 Feb  7 11:01 ./
drwxr-xr-x 1 lab 197121 0 Feb  7 11:01 ../
drwxr-xr-x 1 lab 197121 0 Feb  7 11:01 .git/
```

The ".git" directory shows that this directory is a git project.

Add and commit

Let's first create a new file in the project:

```
lab@lab MINGW64 ~/Desktop/hello (master)
$ ls -al
total 9
drwxr-xr-x 1 lab 197121   0 Feb  7 14:25 ./
drwxr-xr-x 1 lab 197121   0 Feb  7 11:01 ../
drwxr-xr-x 1 lab 197121   0 Feb  7 11:01 .git/
-rw-r--r-- 1 lab 197121 116 Feb  7 14:30 Hello.java

lab@lab MINGW64 ~/Desktop/hello (master)
$ cat Hello.java
public class Hello {
    public static void main( String[] args ) {
        System.out.println("Hello");
    }
}
```

Then you need to take a snapshot to the staging area:

```
git add .
```

You can use `git diff --cached` to see what is added to your project:

```
MINGW64:/c/Users/wangd/Desktop/hello

lab@lab MINGW64 ~/Desktop/hello (main)
$ git add .

lab@lab MINGW64 ~/Desktop/hello (main)
$ git diff --cached
diff --git a/Hello.java b/Hello.java
new file mode 100644
index 0000000..f027015
--- /dev/null
+++ b/Hello.java
@@ -0,0 +1,5 @@
+public class Hello {
+    public static void main( String[] args ) {
+        System.out.println("Hello");
+    }
+}
\ No newline at end of file
```

Note that the `git diff` command with or without the `--cached` argument is different. See git document https://git-scm.com/docs/git-diff for more details.

This is not over, you must commit the changes to finish this step with:

```
git commit
```

```
lab@lab MINGW64 ~/Desktop/hello (main)
$ git commit
[main (root-commit) 2973aed] Create Hello.java
 1 file changed, 5 insertions(+)
 create mode 100644 Hello.java
```

The terminal will prompt for you to enter the "commit message", it is better if you know how to use "vim".

You can also specify the commit message when doing the commit, so that you don't need to type the commit message in the next step:

```
MINGW64:/c/Users/wangd/Desktop/hello

lab@lab MINGW64 ~/Desktop/hello (main)
$ git add .

lab@lab MINGW64 ~/Desktop/hello (main)
$ git commit -m "modified Hello.java"
[main f649003] modified Hello.java
 1 file changed, 1 insertion(+), 1 deletion(-)
```

See log

Now we have done one or two commits. If there are many commits in the repository, you may want to see the log to know the history.

You can choose one of the following command or refer to https://www.git-scm.com/docs/git-log for a detailed demostration.

```
git log
git log -p
git log --stat --summary
```

Making branches

You can have multiple branches in your project.

```
lab@lab MINGW64 ~/Desktop/hello (main)
$ git branch b1

lab@lab MINGW64 ~/Desktop/hello (main)
$ git branch
  b1
* main

lab@lab MINGW64 ~/Desktop/hello (main)
```

In the above example, we created a new branch named "b1" with command `git branch b1`. Then we use `git branch` to show the branches and we see two branches there: "b1" and "main". The "main" branch is the default one. We also see that the current selected branch is "main".

You can switch between branches:

```
lab@lab MINGW64 ~/Desktop/hello (main)
$ git switch b1
Switched to branch 'b1'

lab@lab MINGW64 ~/Desktop/hello (b1)
$ git branch
* b1
  main
```

Now let's do some changes and commit on branch "b1":

Hello.java - Notepad

File Edit View

```
public class Hello {
    public static void main( String[] args ) {
        System.out.println("Hello");
    }
}
// modify on branch b1
```

```
lab@lab MINGW64 ~/Desktop/hello (main)
$ git switch b1
Switched to branch 'b1'

lab@lab MINGW64 ~/Desktop/hello (b1)
$ git branch
* b1
  main

lab@lab MINGW64 ~/Desktop/hello (b1)
$ git add . && git commit -m "add branch b1 comment in Hello.java"
[b1 f9e8e10] add branch b1 comment in Hello.java
 1 file changed, 1 insertion(+)

lab@lab MINGW64 ~/Desktop/hello (b1)
```

Go back to "main" and make some changes:

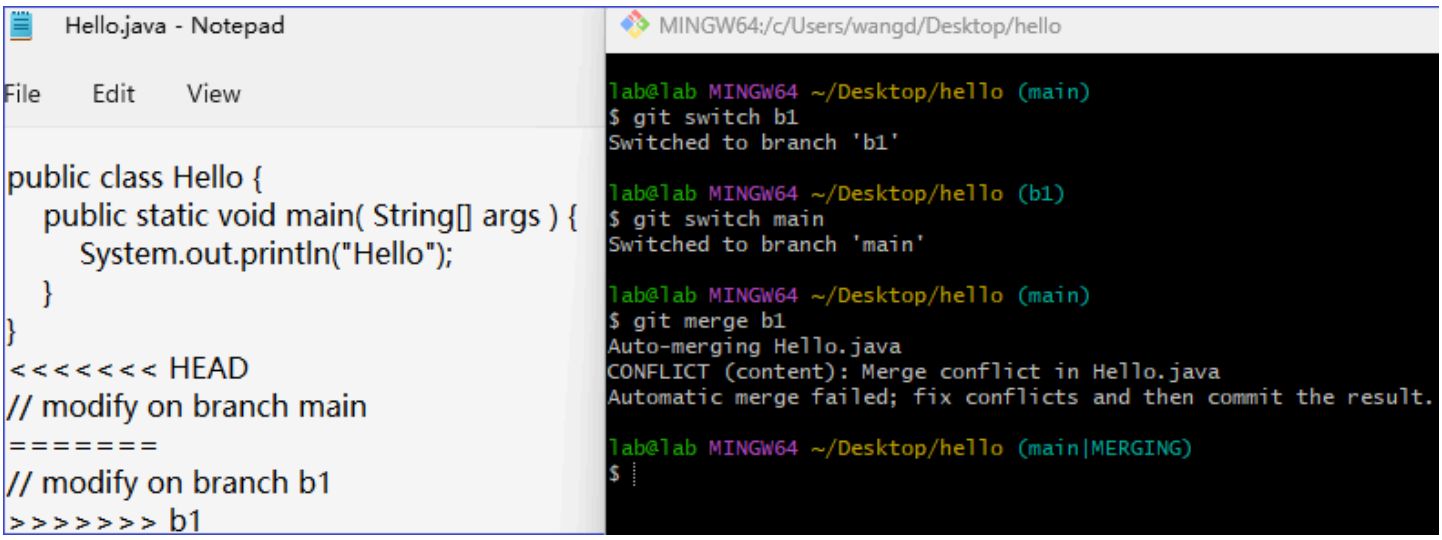
Hello.java - Notepad

File Edit View

```
public class Hello {
    public static void main( String[] args ) {
        System.out.println("Hello");
    }
}
// modify on branch main
```

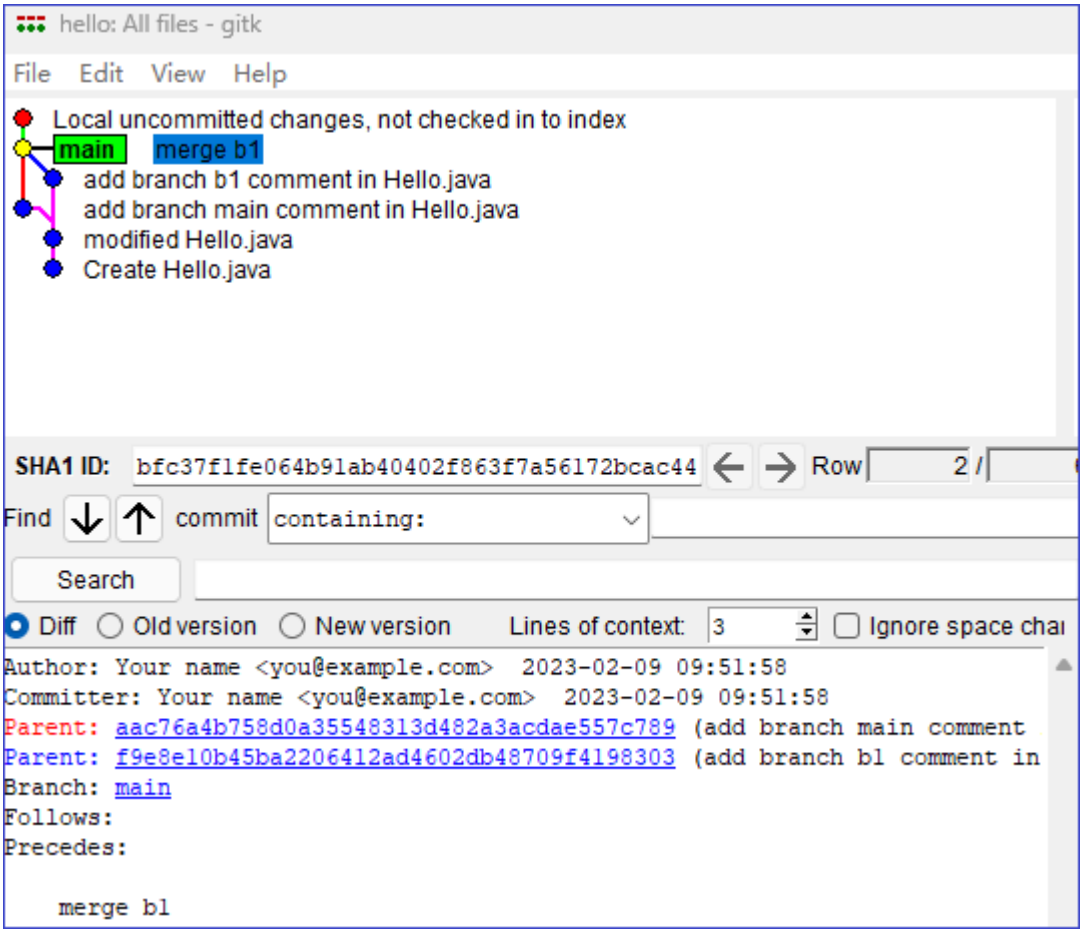
```
lab@lab MINGW64 ~/Desktop/hello (b1)
$ git switch main
Switched to branch 'main'

lab@lab MINGW64 ~/Desktop/hello (main)
$ git add . && git commit -m "add branch main comment in Hello.java"
[main aac76a4] add branch main comment in Hello.java
 1 file changed, 1 insertion(+)

lab@lab MINGW64 ~/Desktop/hello (main)
$
```

Merge branch "b1" from branch "main" with `git merge b1`. If there is a conflict, as follows:



you should manually solve the conflict and do a commit. Then everything is fine and you can remove branch "b1" if it is not useful anymore with `git branch -d b1`. Argument "-d" and "-D" are different, refer to git document for more details.

Now everything is fine and you can execute `gitk` to see a graphical view of the change log:



Git configurations

When you do commit for the first time, git bash may tell you to config your username and email for the first time. You may do it like following:

```
git config --global user.name 'Your Name Comes Here'
git config --global user.email 'you@yourdomain.example.com'
```

Here you are making global changes in your Git.

Apart from the global configs, there are also local configs in project. Open Git bash in Teedy repo and run `git config --list`, Git will show the Git configs and your project's configs. If you add `--show-origin` argument, like:

```
git config --list --show-origin
```

This command show result like this:

MINGW64:/c/Users/wangd/Desktop/Teedy

```
lab@Thinkpad MINGW64 ~/Desktop/Teedy (master)
$ git config --list --show-origin
file:D:/C/Git/etc/gitconfig     diff.astextplain.textconv=astextplain
file:D:/C/Git/etc/gitconfig     filter.lfs.clean=git-lfs clean -- %f
file:D:/C/Git/etc/gitconfig     filter.lfs.smudge=git-lfs smudge -- %f
file:D:/C/Git/etc/gitconfig     filter.lfs.process=git-lfs filter-process
file:D:/C/Git/etc/gitconfig     filter.lfs.required=true
file:D:/C/Git/etc/gitconfig     http.sslbackend=openssl
file:D:/C/Git/etc/gitconfig     http.sslcainfo=D:/C/Git/mingw64/etc/ssl/certs/ca-bundle.crt
file:D:/C/Git/etc/gitconfig     core.autocrlf=true
file:D:/C/Git/etc/gitconfig     core.fscache=true
file:D:/C/Git/etc/gitconfig     core.symlinks=false
file:D:/C/Git/etc/gitconfig     pull.rebase=false
file:D:/C/Git/etc/gitconfig     credential.helper=manager
file:D:/C/Git/etc/gitconfig     credential.https://dev.azure.com.usehttppath=true
file:D:/C/Git/etc/gitconfig     init.defaultbranch=main
file:C:/Users/wangd/.gitconfig  user.email=you@example.com
file:C:/Users/wangd/.gitconfig  user.name=Your Name
file:.git/config        core.repositoryformatversion=0
file:.git/config        core.filemode=false
file:.git/config        core.bare=false
file:.git/config        core.logallrefupdates=true
file:.git/config        core.symlinks=false
file:.git/config        core.ignorecase=true
file:.git/config        remote.origin.url=https://github.com/sustech-cs304/Teedy.git
file:.git/config        remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
file:.git/config        branch.master.remote=origin
file:.git/config        branch.master.merge=refs/heads/master
```

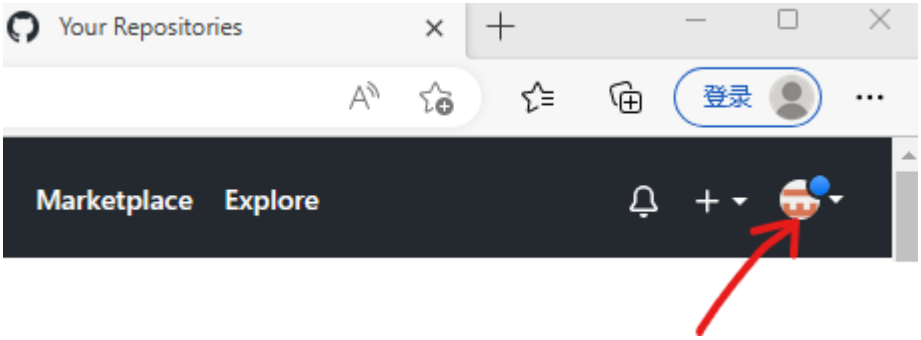
In the first few lines are global configs of your Git. What follows are the local configs of your Teedy repo. You can see they are stored in .git/config file.

There are plenty of useful configs you can edit like default editor and color. You can explore them in Git Configuration.

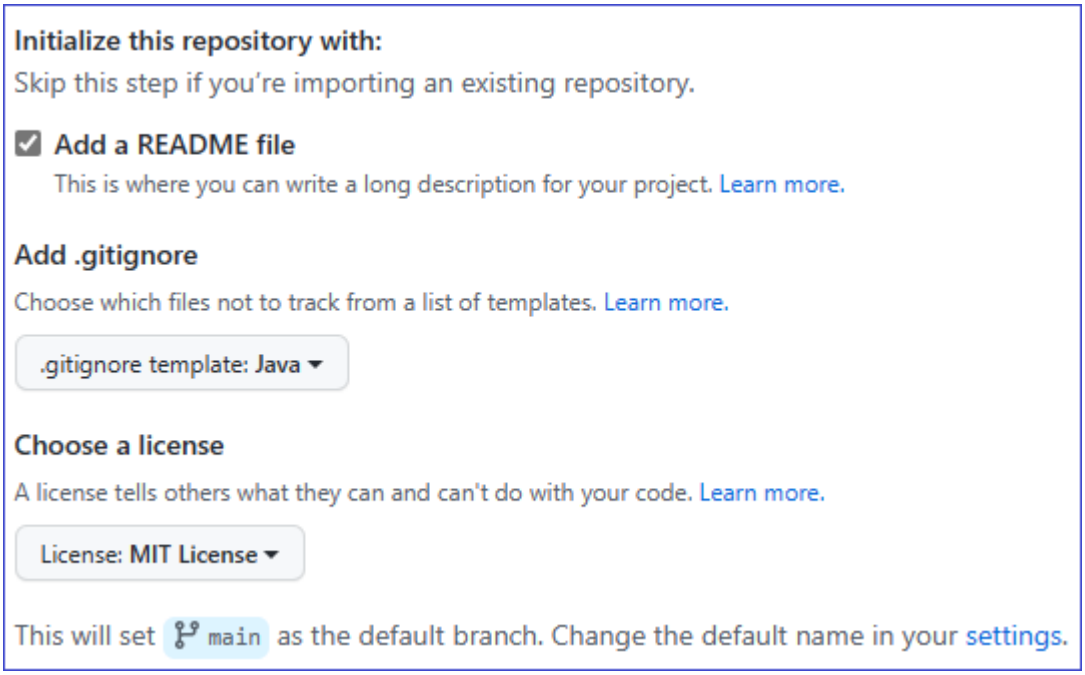
Using Github

Go to https://github.com and register an account if you haven't already done it.

Create Github repo



Click the button above, click "repositories", "new" to create a new repository. When github ask you to set the initial configuration, you can choose to add "readme", ".gitignore" and "licence" file to you repo:

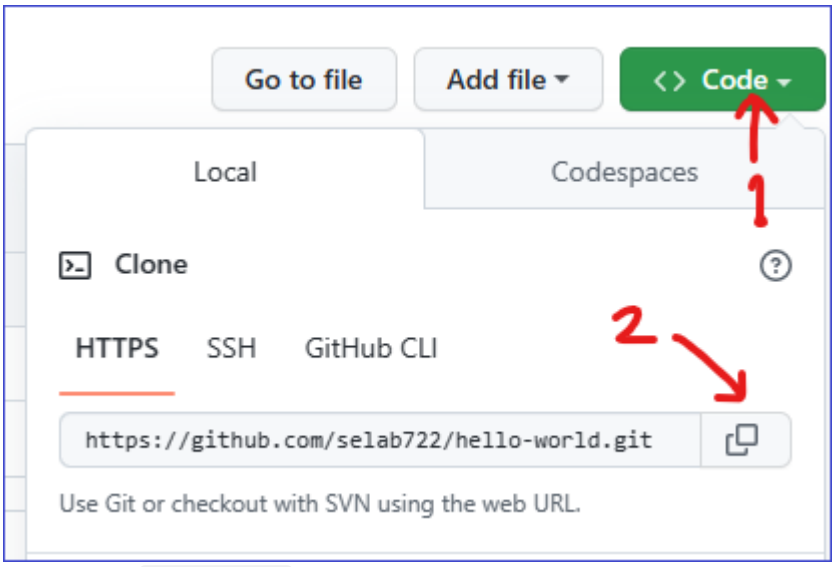


Click "Create repository" and your repository will be created.

You can directly edit some files using github but sometimes you still want to edit them locally, specially when you need an IDE.

Clone repo to local computer

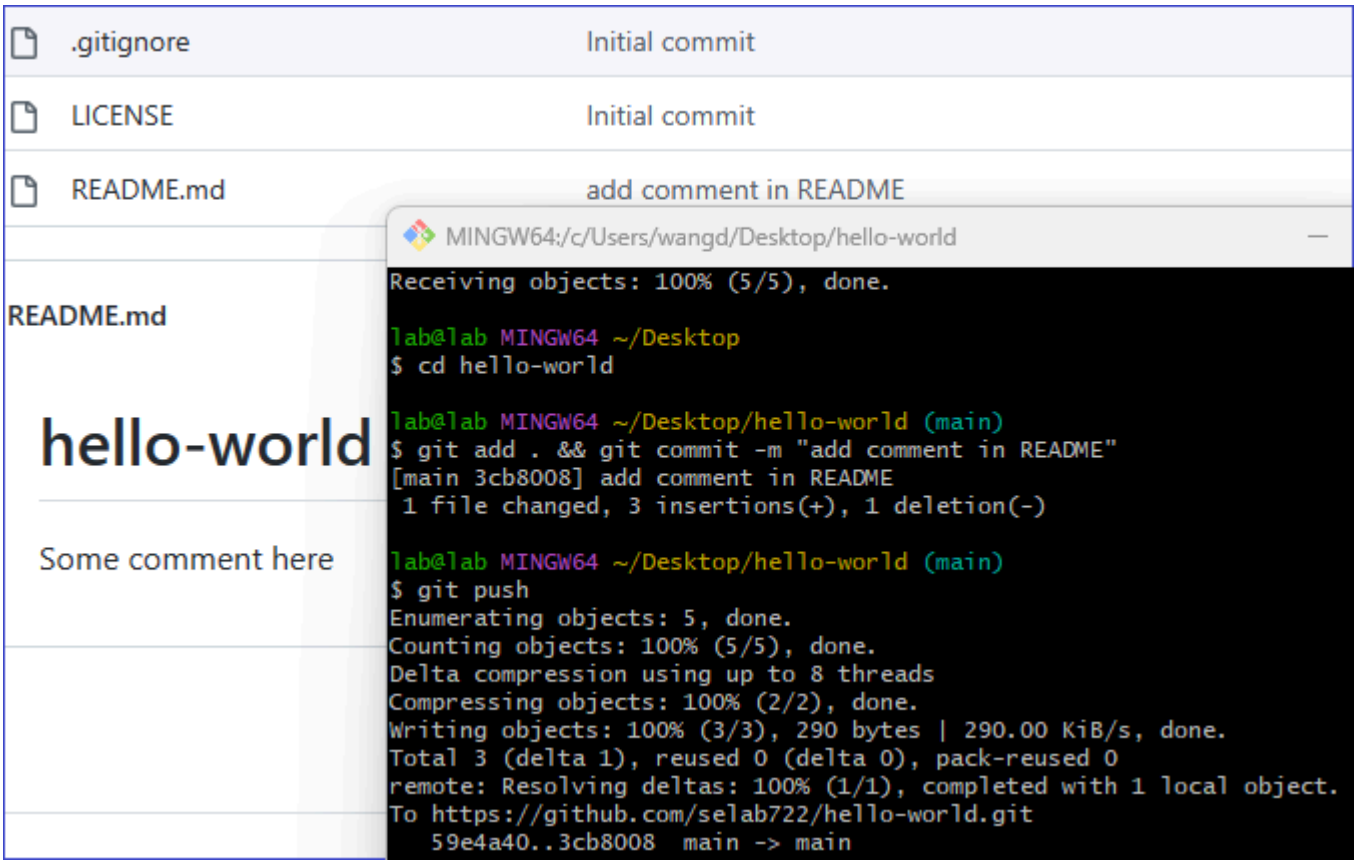
Click "code" and "clone" to copy the link (like https://github.com/username/hello-world.git) to the click board:



Then run `git clone` locally to clone the repository to your local disk:

Pull and push

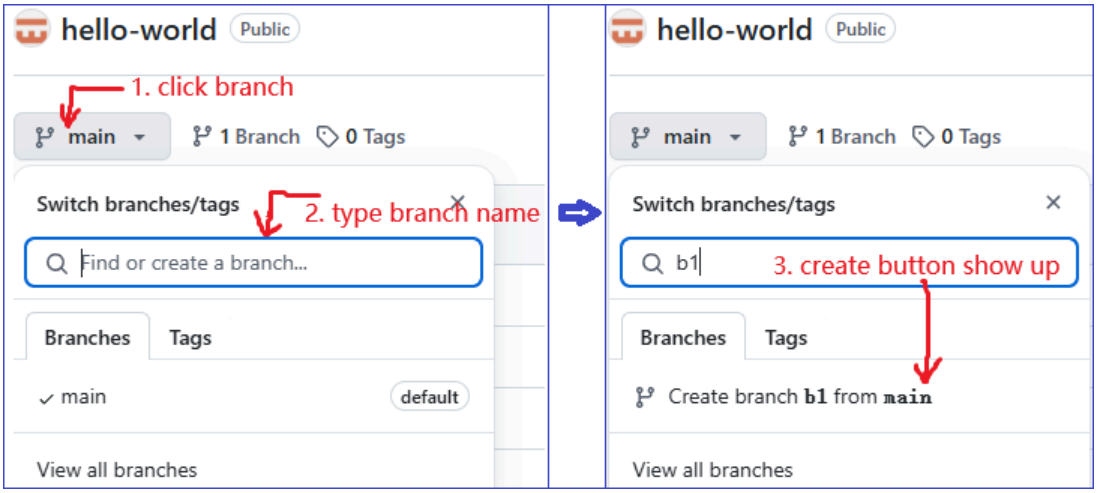
You can now do any modification as a local project (remember to "cd" into the project directory). You can use `git pull` to "pull" updates from the github repo to your local. You can use `git push` to "push" local updates to the github repo.



Pull request

Creating new branches, making commits on new branches, merging changes from new branches, and deleting new branches are possible through a Github repo as well as a local repo.

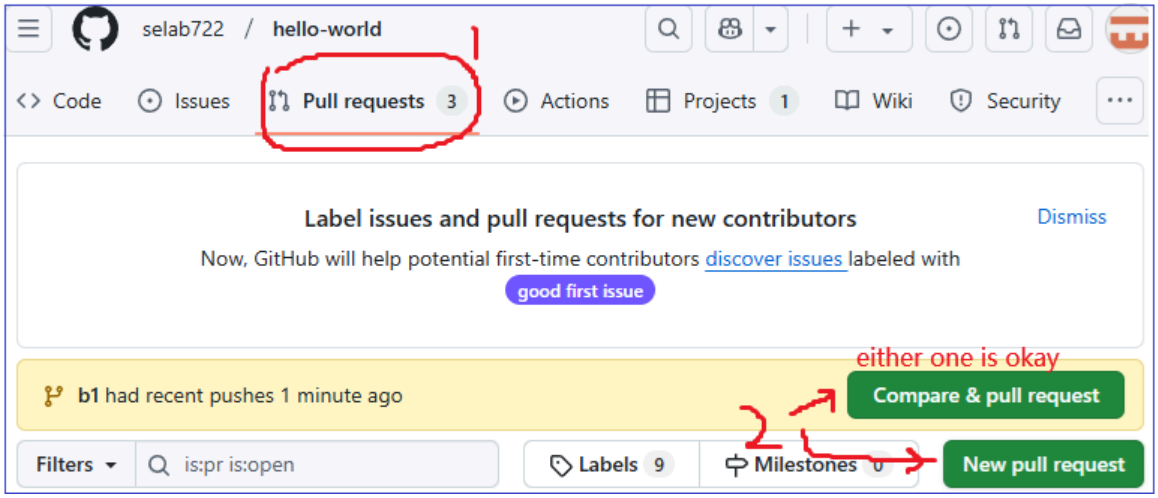
Creating a new branch on a Github repo is simple:



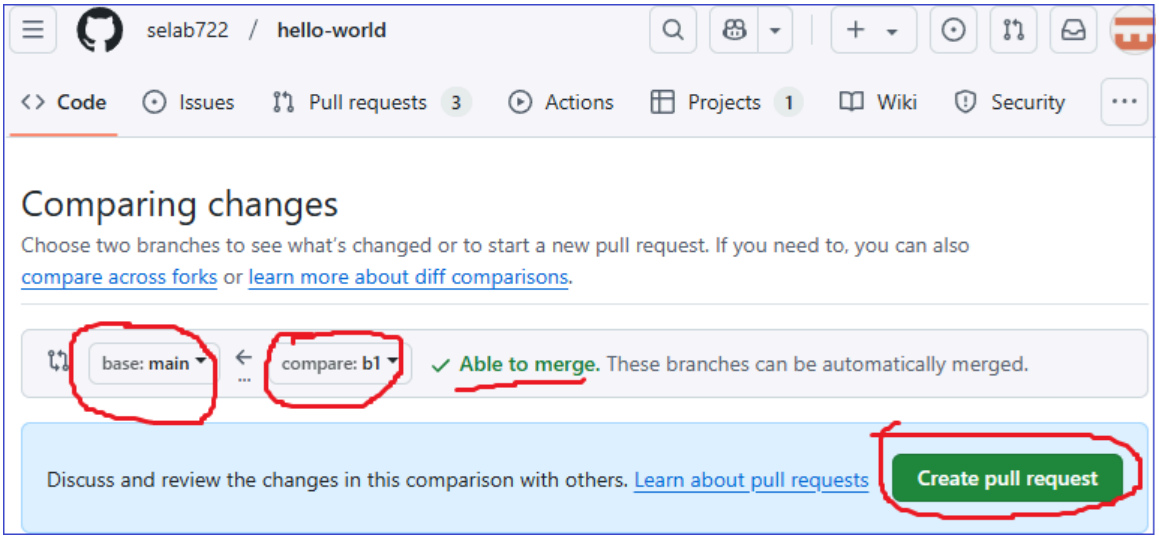
Then you can easily clone the repo to local, switch to the newly created branch, do commits on the branch, then push the new branch.

Another way is to just clone repo to local, then create new branch locally, then push this new branch (not exist on Github yet) to Github. See lab exercise for detail.

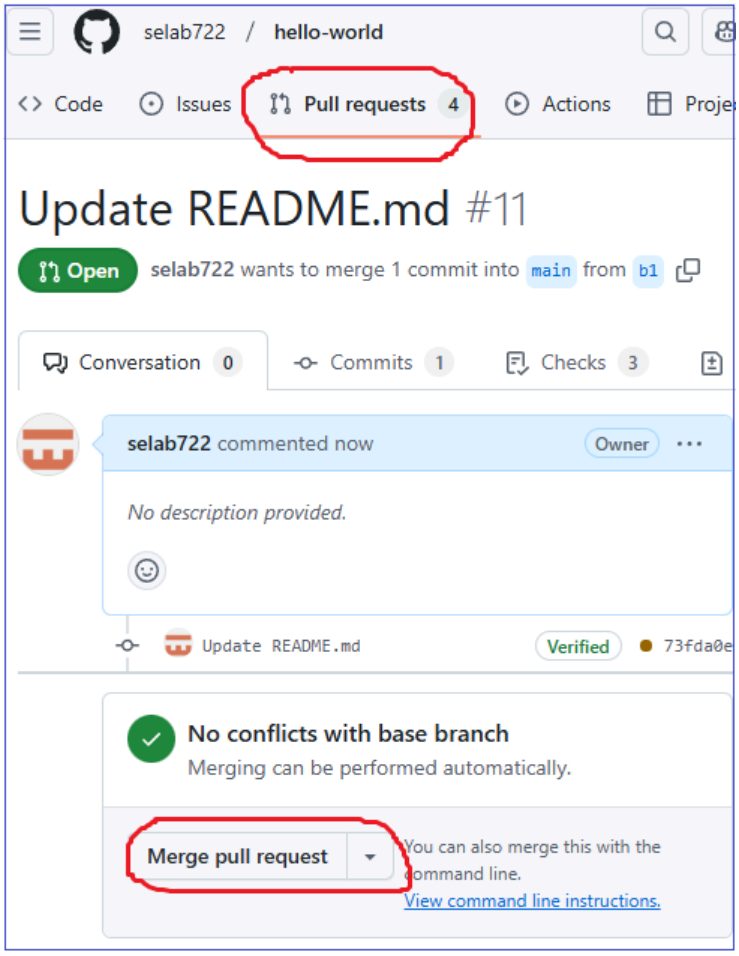
Either way there's a new branch on your Github repo with some new updated commits not existing in main branch. In order to "merge" new branch, create a **pull request**:



Then you need to specify which branch should pull which branch. If no problem, create the **pull request**:



Finally you view the "Pull requests" tab, choose the pull request you just created, then merge it:



Now your merge is successful, your main branch has the new updates and you can delete the new branch.

One problem here is "Why we call it a 'pull request' instead of just call it a 'merge'". Think of this scenario: the original repo is created by someone else and you have no write access to that repo. After you forked their repo, you make changes on your own repo, and you want to "push" the changes to the original repo. But you cannot push since you have no write access. So you can only "request" them to "pull" your changes from your repo to their own repo, if they agree. By creating a "pull request", you request, and they approve.

Git remote add

Let's think about this situation such that there are three repos you need to worry about:

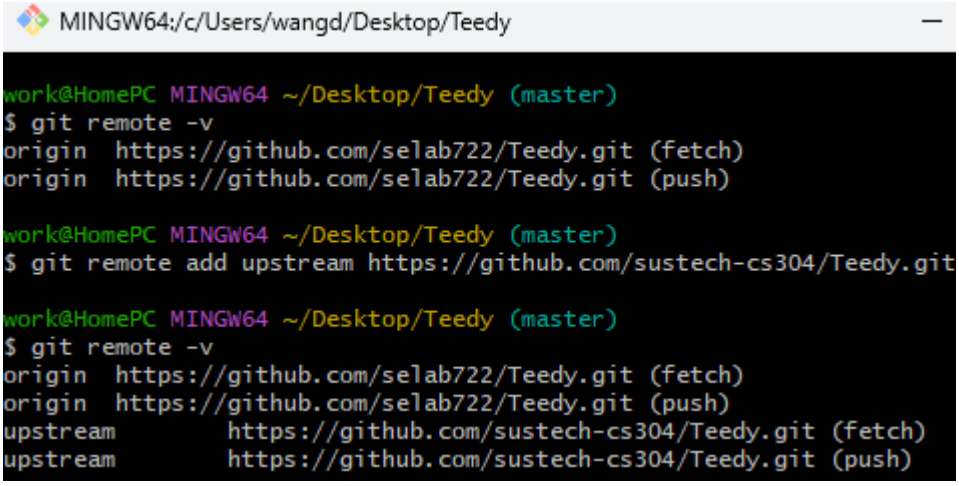
- The original Teedy repo at https://github.com/sustech-cs304/Teedy.
- Your Teedy repo at https://github.com/your-id/Teedy, either forked or "created from template" from the first one.
- Your local repo on your computer, cloned from the second one.

Now there are some new update in the first repo (original Teedy), but not in the second one. You want to **pull** those changes to your local repo, so that you can re-run them to verify the changes, and **push** them to the second repo (your Teedy) later. How to sync changes from original repo to your forked repo?

First, you need to use `git remote add` to configure your repo:

```
git remote add upstream https://github.com/sustech-cs304/Teedy.git
```

You can check the result of this command by running `git remote -v`:



Now you can "pull" from the remote upstream repo (the original repo):

```
git fetch upstream
git switch master
git merge upstream/master
```

This will keep you up to date with the original repo, and let you easily sync your forked repo.

References

- https://git-scm.com/docs/gittutorial
- https://git-scm.com/docs/user-manual
- https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration