Lab 13-1: Using Git for Version Control and Collaboration

Prerequisites

- Git installed on your machine
- A github account
- (optional) SSH key configured in your github account
- VS Code installed (we recommend completing the lab using VS code)

Part 1: Create a local repository

1. Create a directory `lab13-1-git`

```
mkdir lab13-1-git
```

2. create a file `first.txt`:

Using a command or menu to create `first.txt`:

```
# on windows, in a cmd terminal
echo "" > first.txt
# In a Git Bash terminal, or on mac or linux
touch first.txt
```

Add content to the file

```
My first file
```

3. Make the directory a git repository using

```
git init
```

4. Add `first.txt` to the stage area using

```
git add -A
# or
git add .
```

5. Commit the changes to the local repository with a message `added first.txt`

```
git commit -m "added first.txt"
```

6. Check the status of the local repository

```
git status
```

Part 2: Create a github.com remote repository and push the local repository to the remote repository

1. Create a **private** repository on **`github.com`** called `lab13-1-git`.

2. Follow the instructions to push the existing local repository to the remote repository.

Since you have already created the local repository, what you need to do to sync your local repository with the remote repository is:

```
git remote add origin git@github.com:xxx/lab13-1-git.git
git branch -M main
git push -u origin main
```

> The first line estalbishes the **origin** (i.e. the remote repository) that the local repository is tracking. The second command names the local repository's current branch as **main**. The third command pushes the local repository to the origin (remote repository)'s **main** branch.

If you have not set up the SSH key in your github account, you will need to use the https URL instead of the ssh URL.

3. Open a new (bash) terminal. Make sure you're in the parent directory of the local repository.

If you're currently in the `lab13-1-git` directory, you can go to the parent directory using:

```
cd ..
```

4. Clone the remote repository into `lab13-1-git-clone`:

```
git clone git@github.com:xxx/lab13-1-git.git lab13-1-git-clone
```

> The remote repo address (`git@github.com:xxx/lab13-1-git.git`) can be copied from **github.com**.

Again, if you have not set up the SSH key in your github account, you will need to use the https URL instead of the ssh URL.

We will use `lab13-1-git-clone` to simulate Person 2 who is independently working on the same repository as Person 1.

Part 3: Person 2 made changes

In the following, Person 2 has introduced a second file to the repo.

Using Person 2's terminal.

1. In the `lab13-1-git-clone` directory, add a file `second.txt`, with the following content:

```
My second file
```

2. Stage, commit, and push the changes.

```
git add -A
git commit -m "adding a second file"
git push
```

3. Check the status of the local repository

```
git status
```

Part 4: Person 1 pulls the changes and works on a new branch

In the following, we assume that Person 1 will pull the latest changes before working on a new branch.

1. Person 1 can use a `git pull` to get the latest version.

```
git pull
```

As you can see, `second.txt` has been added to the repo.

2. Create a new branch `develop` in `lab13-1-git` and switch to this branch.

```
git checkout -b develop
```

At this point, the branch `develop` is identical to `main`.

We now let Person 1 make changes to the `develop` branch.

3. Person 1 enhances `second.txt` in the `develop` branch.

Make changes to `second.txt` by adding a second line, like the following:

```
my second file
Exciting stuff added by Person 1
```

4. Commit the changes to the local repository:

```
git commit -am "exciting stuff added by Person 1"
```

> `-am` will add changes to **existing** files to the staging area and commit those changes.

5. Push the changes to the remote repository:

```
git push
```

Note that this push fails because the remote does not have a develop branch. The following recommended command will establish a develop branch in the remote repository and push the local changes to the remote:

```
git push --set-upstream origin develop
```

6. Verify that the main branch is intact.

We first checkout the main branch

```
git checkout main
```

Verify that the `second.txt` still has one line in the main branch

Part 5: Person 1 merges the `develop` branch into the `main` branch

In the following, we assume that Person 1 is happy with changes made in the `develop` branch and decide to merge it back into the main branch.

1. first verify that the `main` branch is the current branch

```
git status
```

Verify that the current branch is `main`. If not, run the previous step to switch to the `main` branch.

2. Merge the `develop` branch into the `main` branch

```
git merge develop
```

The merging should succeed, since the main branch has not changed.

Verify that the `second.txt` in the **main** branch has two lines.

3. Push the changes to the main branch to the remote

```
git push
```

Part 6: Person 2 made a conflicting change.

In the following, we simulate that Person 2, without knowing changes made by Person 1, has also made changes to `second.txt` in the main branch.

1. Go back to Person 2's terminal.

2. Make changes to `second.txt` in the main branch.

```
My second file
Exciting stuff added by Person 2
```

3. Add, commit, and push the changes

```
git add .
git commit -am "new stuff added to second.txt"
```

Then, Person 2 pushes the changes:

```
git push
```

The push will fail because Person 1 has made changes to the `second.txt` on the same line.

4. Resolve the conflict manually.

You may use VSCode's conflict resolution feature to resolve the conflict.

Specifically, we will accept both Person 1 and Person 2's changes, letting Person 1's change appear on the second line.

5. Add, commit, and push the merged changes

We need to add, commit, and push the merging actions

```
git add .
git commit -am "resolve a conflict Person 1 and Person 2's changes to second.txt"
git push
```

6. Go back to Person 1's terminal and let Person 1 pull the changes

```
git pull
```

Verify that you have three lines in `second.txt`.

End of the Lab

submission requirements

Run the following two commands and submit the screenshots of the commands and output (image files) to Canvas.

```
git log --oneline --graph --all --pretty=format:"%h %ad %s"
```

```
ls -l
```