Introduction to the GIT

Imane Fouad, UM6P

1 Objectives

- Learn how to use a Version Control Software.
- Master versioning of a software project.
- Share a project and work as a team.

Notice

Read carefully the output of each git command. If you are not familiar with a command, consult the documentation.

2 Configuring Your Git Account

In a project managed by multiple people, it is important to know who did what. Git therefore needs a configuration to identify your future commits. You can also specify certain behaviors.

The following commands will modify your `.gitconfig` file. Type them one by one while observing the changes in the file:

```
git config --global user.name "Your Name"
git config --global user.email your.name@um6p.com
```

Windows tip: You can use the following command to check the updates in your `.gitconfig` file:

```
type .gitconfig
```

After running these commands, Git will associate your commits with the specified name and email.

Reminder

Git configurations need to be done only once for each user account.

3 Initializing a Git Repository

We will work on a project called `tpgit`. Using the course materials and the documentation, you should:

- Create a directory named `tpgit`
- Initialize a Git repository in this directory
- Inspect the files created for the repository; they will appear in the `.git` folder in terminal.

- Create a file named `fruits.txt`.
- Make a commit by adding 2 fruits to `fruits.txt`. The usual sequence of commands is: **status, add, status, commit, status, log** .

  Read the output of each command to understand its function.
- Make 3 additional commits, adding new fruits each time.

Tip

Use `git status`, `git diff <filename>`, and `git log` to understand the state of your repository. For a detailed global history with all branches, tags, and commits, use:

```
git log --graph --oneline --decorate --all
```

4 Git Branches

Branch: `vegetables`

- Create a branch named `vegetables` and switch to it
- Verify that you are on the `vegetables` branch (either by looking at the graph or running): *git branch*
- Add a file `vegetables.txt` on this branch.
- Make 3 separate commits by adding vegetables to `vegetables.txt`.
- Check the commit history using `git log`

Branch: `sauces`

- Create a branch named `sauces`
- Add a file `sauces.txt` on this branch.
- Make 2 separate commits by adding content to `sauces.txt`.
- Verify the commit history and confirm the existence of 3 branches (`master`, `vegetables`, `sauces`) using `git branch` and `git log`.

5 Git Merges

You have decided that your commits in the `vegetables` and `sauces` branches are relevant. Now you need to integrate them into `master`.

- Go to the `vegetables` branch
- Check the state of your working directory (for example, note that `sauces.txt` does not exist yet).

- Merge `master` into `vegetables`
- If everything went well, merge `vegetables` back into `master`
- Verify the commit history and the appearance of `vegetables.txt` in the `master` branch
- Now merge the `sauces` branch into `master` by following the same process
- Confirm that all branches are up to date.

6 Remote Repository

We will now create your accounts on the GitHub platform. Go to the GitHub sign-in page `https://github.com/signup`. Create your account using your email address.

Notice!

There are two network protocols for communicating with GitHub: HTTPS and SSH.
We recommend using HTTPS at first. In the future, you may choose SSH, which is more powerful and flexible, but you will need to configure your private and public keys.
For guidance, follow GitHub's instructions on generating a new SSH key pair.

1. Log in to https://github.com.
2. Click on **New Repository** (top right) to create a new project.
3. Give your project a name (e.g., `tpgit`) and click **Create repository**. **Do not check** "Add README."
4. You are now in your newly created (empty) repository. GitHub will show you instructions for different ways of initializing it. Read them carefully. If you do not use SSH keys, make sure to copy the HTTPS URL instead of SSH.
5. Follow the instructions under **...or push an existing repository from the command line**. Run the following commands (replace `<user>` and `<project>` with your actual GitHub username and repository name):

   ```
   git remote add origin https://github.com/<user>/<project>.git
   git remote -v
   git push -u origin master
   ```

   The first command adds a remote connection named `origin`, the second Shows the URLs of the remote repositories connected to your local repository, and the third pushes your local `master` branch to GitHub.
6. Refresh your GitHub repository page. Your project `tpgit` should now be visible online.
7. Explore your project on GitHub: check the commit history, browse your files, and view graphs.

Managing Project Members Add your teammate to your GitHub repository. Make sure your teammate has created an account and signed in at least once so they appear in the list.

1. Go to your repository page on GitHub.
2. Click on **Settings** (top-right menu).
3. In the left menu, click **Collaborators**.
4. Click **Add people**.
5. Search for your teammate's GitHub username and add them to your repository.

7 Collaborative Work

You will work in pairs on the repository of Pair 1. Pair 2 must clone Pair 1's repository into a folder named `tp-git-binome`.
Steps:

1. Pair 2 should go to Pair 1's repository on GitHub.
2. Click on **Code** and select **HTTPS**.
3. Clone the repository locally using the following command:

   ```
   git clone <repository-HTTPS-URL> tp-git-binome
   ```

4. Navigate into the cloned directory and examine the commit history

8 Merge Without Conflict

Work for student 1

1. Create a branch `spices`
2. Add a file `spices.txt` on this branch.
3. Make 2 separate commits on `spices.txt`
4. Merge your branch into `master`
5. Pull any changes from the remote repository: (git pull)
6. Push your changes to GitHub:(git push)

Work for student 2

1. Create a branch `herbs`

2. Add a file `herbs.txt` on this branch.
3. Make 2 separate commits on `herbs.txt`.
4. Merge your branch into `master`
5. Pull any changes from the remote repository: git pull
6. Push your changes to GitHub: git push

Both pairs should:

1. Verify that `spices.txt` and `herbs.txt` exist in both local repositories.
2. Check the commit history in both repositories: Are all commit IDs the same? Can you identify the last merge commit? Can you see your partner's commits?

One pair pushed their changes first. The other pair had to pull and merge these changes before pushing their own work. Because each pair worked on **different files**, Git automatically resolved the merge. Next, we will intentionally create conflicts and learn how to resolve them.

9 Merge with Conflict

Generating a conflict

1. Both pairs edit same lines in `fruits.txt` in their local repositories:Delete some fruits and add new ones, Make sure each pair makes **different changes**.
2. Commit your changes locally, but do **not push yet**:
3. Pair 1 pushes their changes first
4. Pair 2 tries to push, but first must pull Pair 1's changes Git will report a **merge conflict**.
5. Check the status and understand the conflict
6. Using your course material, resolve the conflict in `fruits.txt`.
7. Once resolved, commit the merge and push to GitHub
8. Pair 1 pulls the latest changes and verifies the state of `fruits.txt`.

Generating a second conflict (switch roles)

1. Repeat the same steps, but this time use `vegetables.txt` and **reverse roles**: Pair 2 pushes first, Pair 1 pulls and encounters the conflict.
2. During the conflict (after `git pull`, you can **abort the merge** if needed:

   ```
   git merge --abort
   ```

   - Verify that the repository returns to its previous state (check with `git status`).
3. Redo the pull to encounter the conflict again, then resolve it properly, commit, and push.

Tip

To reduce conflicts, make sure everyone keeps their repository up to date. Regularly run `git pull` on your branches and merge changes from the remote into your branch frequently. Resolve small conflicts immediately while they are still simple. Keeping branches synchronized helps avoid larger, more complicated conflicts later.

10 Document Your Project Using Markdown

Markdown is a lightweight markup language that is faster to write than HTML and very flexible. It is automatically rendered by GitHub (and many other platforms) and can be used to document your project or even write reports.

Create a `README.md` file at the root of your repository. Use Markdown syntax to include titles, lists, and code snippets. At a minimum, your README should include:

- A description of your project.
- A section **Authors** listing the contributors.
- A section **Objectives** describing the goals of your project.

You can refer to this example on GitHub for inspiration: https://gist.github.com/PurpleBooth/109311bb0361f32d87a2

Click on **RAW** to see the source. Verify that your README displays correctly on GitHub.

11 Continue Learning

Tutorials:

- https://crypto.stanford.edu/~blynn/gitmagic/intl/fr/book.pdf
- https://learngitbranching.js.org/
- https://try.github.io/
- https://git-scm.com/book/en/v2

Videos:

- https://www.youtube.com/watch?v=OqmSzXDrJBk
- https://www.youtube.com/watch?v=uR6G2v_WsRA
- https://www.youtube.com/watch?v=3a2x1iJFJWc
- https://www.youtube.com/watch?v=1ffBJ4sVUb4
- https://www.youtube.com/watch?v=duqBHik7nRo