Github

Monika Avila Marquez

2025-06-15

Table of contents

Preface

Notes on Git and Github use

Part I

Basics

1 Git and Github

1.1 Version control

Version control allows to control any changes done to a code. There are different types of version control systems, but the most useful ones are the distributed ones.

1.2 Git

What is Git:

- It is an free and open-source software distributed under the GNU license.
- It is a distributed version control system (DVCS): this means that any user can have a copy of a project, make changes locally and then sync changes with the main remote project.

What does it do: - It supports branching strategy.

Brackground: was developed in 2005 to satisfy Linux needs of version-control.

1.3 Github

It is a host service for Git repositories.

1.4 Important terms to learn

- SSH (secure shell) protocol: method to secure remote login.
- Repository: contains the project folders that are set uo for version control.
- Fork: copy of a repository
- Pull request: is a request to someone to review and approve your changes before they become final.
- Working directory: contains files associated to a git repository
- Commit: a snapshot of the current state of a project

- Branch: is a separate line of development that allows you to work independently on your changes.
- Merging: combines changes from one branch to another.
- Cloning: creates a local copy of a remote git repository

2 Configurar Git

1. Install Git in your computer (Set the initial configuration in your computer):

- In Rstudio:
  - `install.packages("usethis")`
  - Set your username and email: `usethis::use_git_config(user.name = "x",user.email = "x")`
  - Configure your personal access token:
    - Create the token with `usethis::create_github_token()`
    - Save the token in your computer with `gitcreds::gitcreds_set()`
  - Vaccinate your Git installation with `usethis::git_vaccinate()`

2. Check the initial configuration of Git in your computer.

- Terminal: `git config --list`
- R: `usethis::git_sitprep()` You will need to set your user name and provide a token. The token is obtained from GitHub > Settings > Developer settings > Personal access tokens > Tokens (Classic) > Generate new token.

3. Different ways to interact with Git:

- Terminal
- GitHub desktop
- R

3 Git branches

Git branches store all files in Github. It can be thought as parallel copy of a repository.

3.1 Main branch

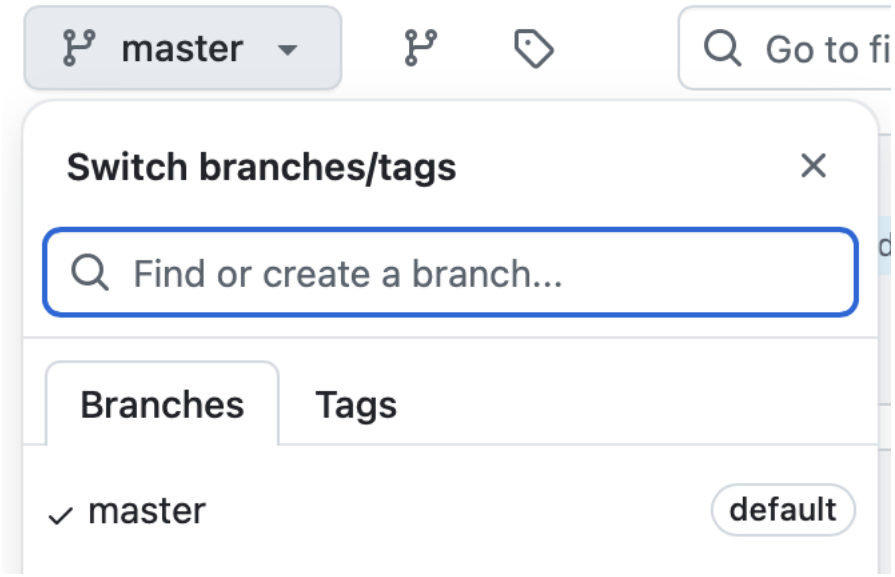
The main branch stores the deployable code.

3.2 Other branches

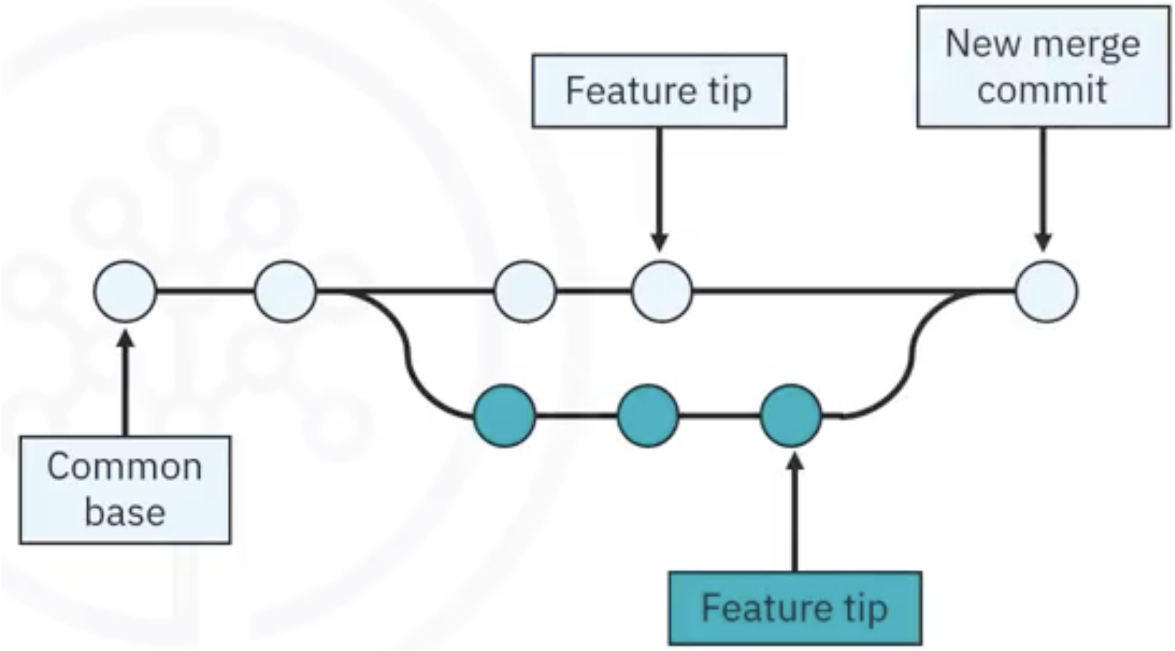
When you plan to change the code:

1. Create a new branch:

- Command line: in the local repository of interest (previously you have changed to the local repository using cd and make sure that the local repository is a git repository which means that you have used `git init` in the directory of interest), use `git checkout -b "name of branch"`. This new branch is a exact copy of the original branch.
- Github: in the repository of interest, go to tab “master”, and fill in the space that indicates “Find or create a branch”



2. Work on the new branch. Once the new code is finished, we have two branches: the original and the copy.
3. When both branches are ready to join, the code of each branch is called the tip of each branch.
4. Merge the branches: the two tips of the branch are merged into a new tip that belongs to the original branch that contains the deployable branch.



Git Branching Workflow

1. Clone the repository `git clone URL`
2. Create branch `git branch BRANCH_NAME`
3. Switch to the new branch using `git checkout BRANCH_NAME`
4. Modify the targeted file
5. Check modifications using `git status`
6. Stage changes using `git add FILE_NAME`
7. Commit changes using `git commit -m "Message"`
8. Switch back to the main branch using `git checkout main`
9. Merge branches using `git merge`
10. Make the changes accessible to the remote repository using `git push -u origin main`
11. Check if changes have been done using `git status`

4 Main branch

In RStudio, we might get instead of main master. To change it:

```
usethis::git_default_branch_rename(from = "master", to = "main")
```

To configure it as default:

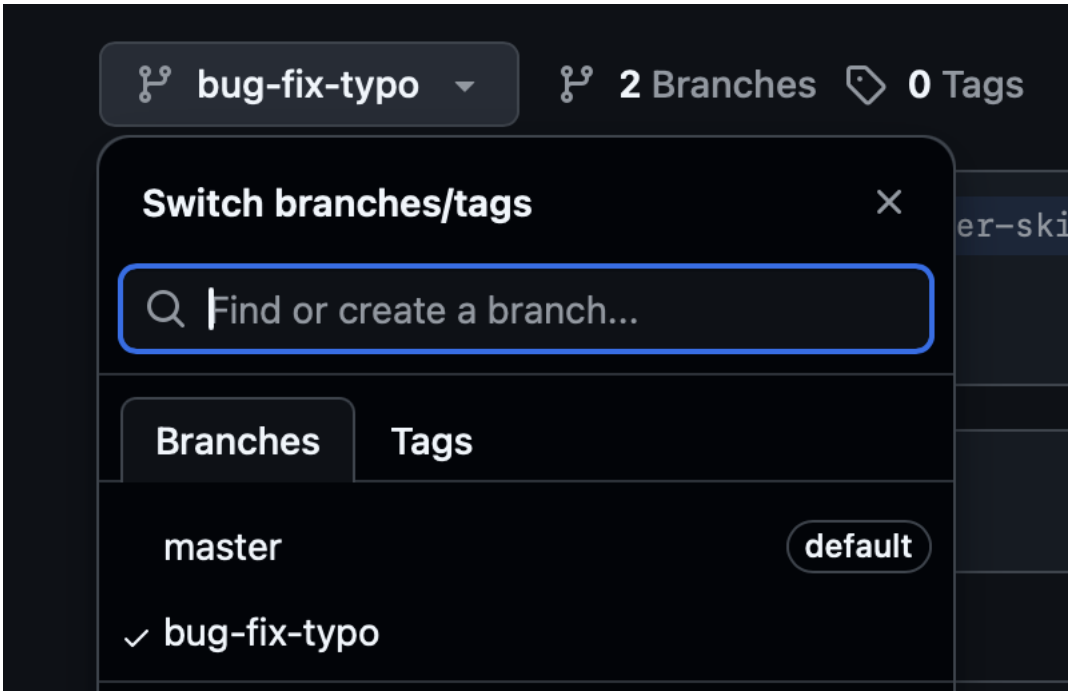
```
usethis::git_default_branch_configure(name ="main")
```

5 Changes to code

To change code or a file, one needs to:

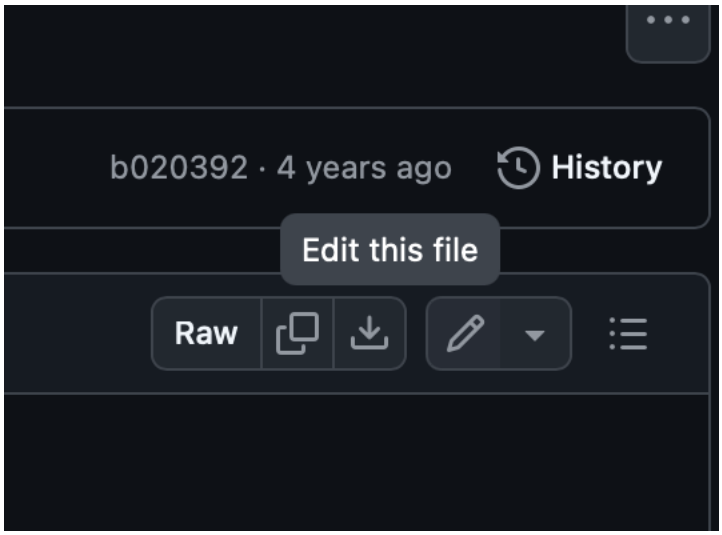
1. Go to the branch used for development. Important: not the main branch containing the deployed code!

- In the command line:
  - `cd "name of local git repository"`
  - `git checkout "name of branch"`
  - `nano "file name to change"`
- In Github:
  - Choose the branch for development:



  - Select file

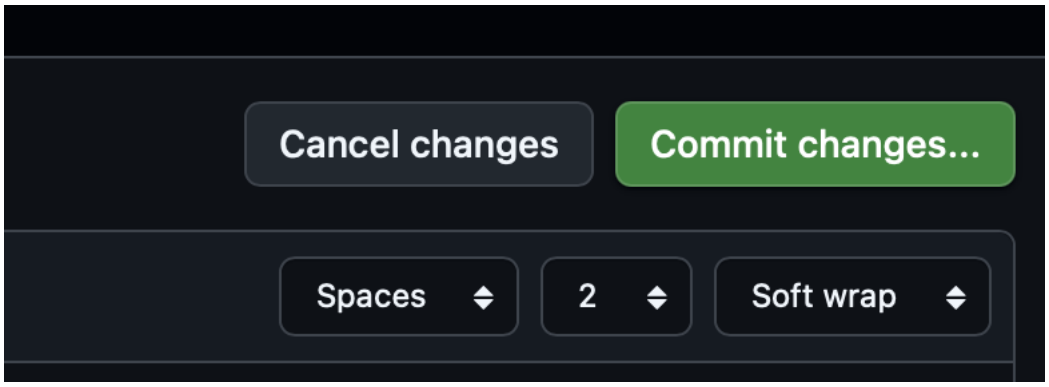
– Click pencil icon



– Make changes to the file.

6 Git commit

- A commit is a saved change of files. For instance, we changed a file and we want to save it. In word, we would click save. In Git, we need to use commit.
- Important: we need to add a commit message which has to describe correctly the changes done.
- Commit changes to the file:
  - In the terminal (assuming you are already in the directory of the local git rpository):
    * `git add .` (This allows to upload the file to a staging area)
    * `git commit -m "message"`
  - In Github



6.1 Commit message best practices

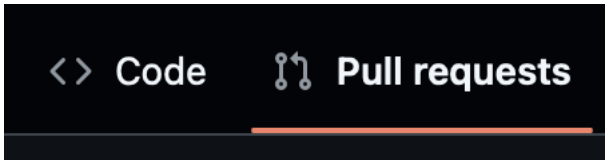
- Don't end with a period
- Less than 50 words
- Active voice

7 Pull request

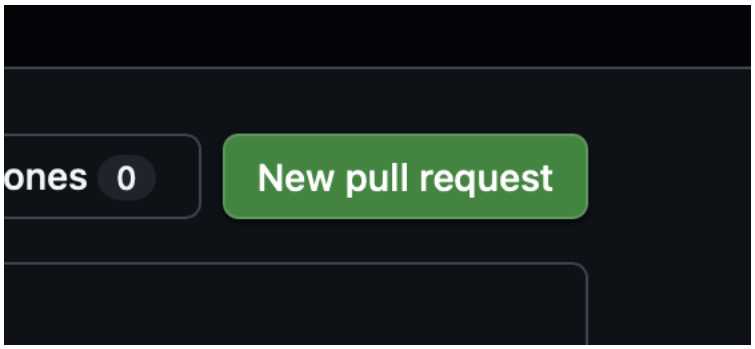
- A pull request makes available the committed changes available to others for review and use.
- Can follow any commit.
- Can target any user.
- There is always a log file of the pull that registers the user who approved the pull.
- It allows to merge a new branch with the main one.

To make a pull request: - On command line: -

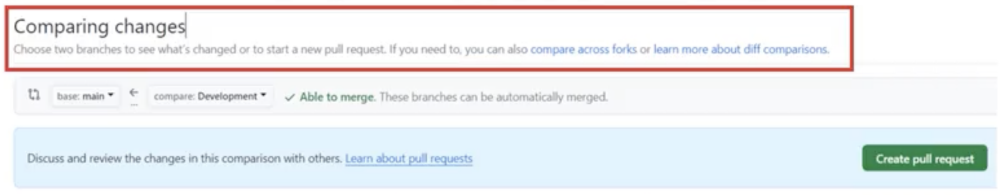
- On Github:
  - Go to pull request



  - Select new pull request



  - Select the new branch from the compare box



Source: IBM Git coursera course

  - Confirm that the changes are the ones that you want to asses
  - Add title and description of request
  - Click Create pull request

- Then code is reviewed and approved
- Approved code in merged in the main branch

8 Merge a pull request

To merge a committed code into the main branch:

- On Github:
  - Click Merge Pull Request
  - Click Confirm merge
  - Delete obsolete branch
- On command line:

Part II

Workflow

Existing project

1. Clone the remote Git repository on your local computer.
2. Create a new branch to make changes in the targeted file. This is done to avoid to make direct changes in the main branch that contains the deployable code.
3. Add the modified files to a staging area.
4. Commit the modified files to the new branch.
5. Pushes the changes in the branch to the remote repository (Push commits to the remote repository).
6. To merge the new branch to the main branch in the remote repository, creates pull request.
7. The maintainer reviews the pull request and merges it with the main branch.
8. Maintainer creates Release1 branch
9. Users pull changes in Release1 to local repositories
10. Users perform testing locally
11. Push commits to remote repository
12. Create pull request
13. Maintainer aproves pull requests and merge changes

New project: you start from scratch

Option 1

1. In the terminal, go to the desired directory where you want to host your local repository using `cd`.
2. Initialize a local Git repository `git init` . This allows Git to track changes.
3. If there are files you do not want to keep track of, create a .gitignore file for your repository using `touch .gitignore` . You can modify this file on the terminal with `nano .gitignore` . This step is important.
4. Move selected files to an Staging Area using `git add .`
5. Make an initial commit `git commit -m "message"` .
6. Create a blank remote repository on Github.

7. Link local repository to the remote repository `git remote add origin URL_github_repository` .
8. **Push** files from local repository to the remote repository `git push --all origin` .
9. Other users can now clone the remote repository.
10. Others users start working on the repository by creating branches

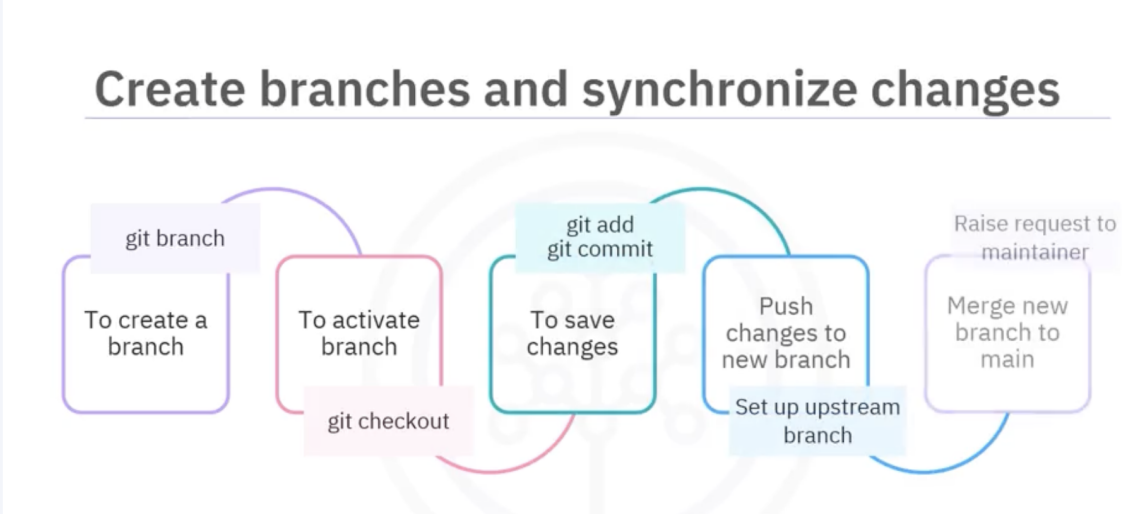
Option 2 (My idea)

1. Initialize a remote Git repository
2. Clone the remote Git repository
3. Select files that you want to keep tracked
4. Move selected files to an Staging Area
5. Make an initial commit
6. Push files from local repository to the remote repository

9 Workflow using commands

9.1 New Project

1. Create a local directory using `mkdir`
2. Navigate to the directory `cd`
3. Initialize a local repository using `git init` in the target local directory
4. Add the files that you want to control version to the local repository
5. Add the files to the staging are using `git add name_files`
6. Commit the files to the main branch using `git commit -m "message describing the commit"`
7. Review the commit history using `git log`
8. To make changes without causing problems in the main branch, create a new branch using `git branch name_child_branch`
9. To switch to the new branch use `git checkout name_child_branch`
10. Make changes on the targeted file in the new branch.
11. Add the changed files to the staging area using `git add name_file_to_change` .
12. Commit the changes using `git commit -m "commit_message"` . Here it is important to specify in the commit message that changes are done in the name of the branch
13. Verify the commit using `commit log`
14. To see the status of changes `commit status`
15. Push commit `git push` ???
16. Merge the changes using `git merge name_child_branch`
17. Delete the child branch using `git branch -d name_child_branch`



Source: IBM coursera course on Introduction to Git and Github

9.2 Existing Project

1. Clone the remote repository using `git clone url_direction`
2. Change to the directory that was cloned using `cd`
3. Create a new branch to work on a target change `git branch name_new_branch`
4. Change of branch using `git checkout name_branch`
5. Then go to the file to modify. Modify it
6. Add changes to the staging are using `git add filename`
7. Commit changes using `git commit -m message`
8. Merge branches
9. Switch back to main `git checkout main`
10. Merge changes using `git merge childbranch`
11. Make changes accessible in the remote repository `git push -u origin main`

10 Git Branching Workflow

10.1 Workflow 1: merge branches first locally and then push to the remote

1. Clone the repository `git clone URL`
2. Create a new branch, you can use `git branch BRANCH_NAME`. But you can better use `git checkout -b BRANC_NAME`, this will create and change automatially to the new branch.
3. Switch to the new branch using `git checkout BRANCH_NAME`
4. Modify the targeted file
5. Check modifications using `git status`
6. Stage changes using `git add FILE_NAME`
7. Commit changes using `git commit -m "Message"`

7.5 Push the changes to the target branch `git push -u origin Target_branch_name`. This step is important to push the branch that was created to make the changes to GitHub.

8. Switch back to the main branch using `git checkout main`
9. Merge branches using `git merge BRANCH_FORCHANGES_NAME`
10. Make the changes accessible to the remote repository using `git push -u origin main`
11. Check if changes have been done using `git status`
12. Now you can go to GitHub and check that you have three branches.
13. In GitHub you can ask for a pull request

10.2 Workflow 2: merge branches first remotely and then pull: recommened

1. Clone the repository `git clone URL`
2. Create a new branch, you can use `git branch BRANCH_NAME`. But you can better use `git checkout -b BRANC_NAME`, this will create and change automatially to the new branch.
3. Switch to the new branch using `git checkout BRANCH_NAME`
4. Modify the targeted file
5. Check modifications using `git status`
6. Stage changes using `git add FILE_NAME`
7. Commit changes using `git commit -m "Message"`

7.5 Push the changes to the target branch `git push -u origin Target_branch_name`. This step is important to push the branch that was created to make the changes to GitHub.

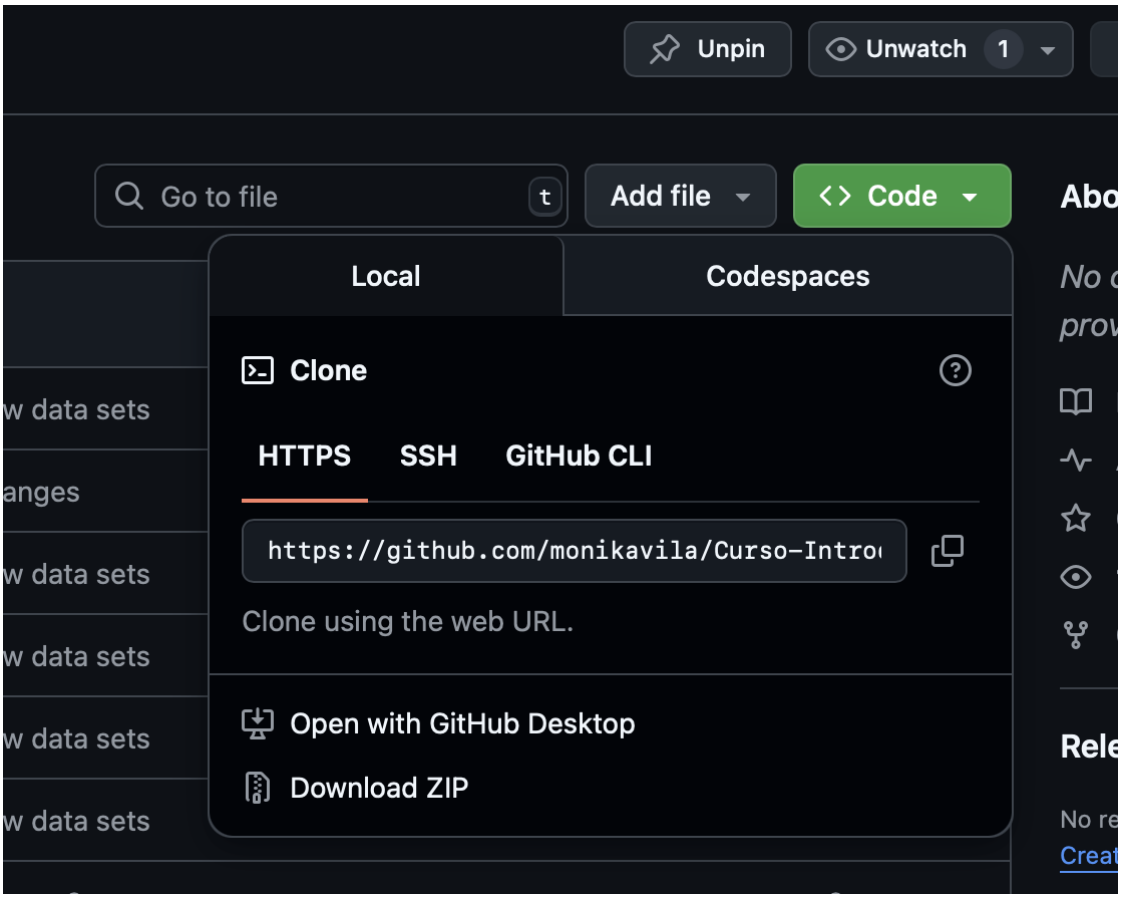
8. In Github make a merge pull request
9. Approve the merge
10. Pull the merge into your local repository using `git checkout main` and the `git pull`

It is better if we use continous integration, this means use Github actions to test.

11 Cloning

Cloning allows you to copy an existing remote repository in your local computer. This can be done by following the process:

1. In Github, go to the repository of interest and click Code, and copy the HTTPS link.



2. On the terminal, go to the desired directory where you want to host the cloned repository.
    1. `git clone "the copied HTTPS"`

12 Commands

12.1 Line commands

Basic line and Linux commands are:

- `mkdir` : create a new directory
- `cd` : change of working directory
- `touch` : create a file
- `nano file_name` : to modify a file on the terminal

12.2 Git commands

- `git add .`: adds all files to the staging area
- `git add file_ _name` : adds a specific file to the staging area
- `git commit -m "message"`
- `git log`: shows history of commits
- `git branch branch_name` : creates a new branch
- `git checkout branch_name`: switches from branch to branch
- `git status`: to see the status of changes
- `git merge`: allows to merge the new branch to the main branch

13 Git revert

To revert a commit:

1. Create a new branch `git checkout -b BRANCH_NAME`
2. Revert commit `git revert BRANCH_NAME`
3. Push branch to GitHub `git push - -u origin BRANCH_NAME` You will need a personal access token that is generated in Settings > Developer settings > Personal access tokens > Tokens (classic) > Generate new token
4. Go to GitHub, there you will see the new branch.

5. There you can create a pull request.

14 Creating a new repository in Git and Github

1. Navigate to the direcory that will host the local repository.
2. In the terminal tab `git init -b main`
3. Add a gitignore file using `touch .gitignore`
4. Stage files added to the repo `git add .`
5. Commit the files `git commit -m "Initial commit"`
6. Create a GitHub repository using on GitHub without README and .gitignore files.
7. Link the local repository to the remote repository using `git remote add origin URL_GITHUB_REPOSITORY_TO_LINK`
8. Verify that the remote was set up correctly using `git remote -v`
9. Push the changes in the local repository to the remote one by running the comand `git push -u origin main`
10. Verify that your local branch is up to date with the remote one by `git fetch` `git status`
11. To create a new branch `git checkout -b NAME_NEW_BRANCH`
12. To push the local branch to the remote `git push -u origin NAME_NEW_BRANCH`
13. To pull the remote branch to the local one -u origin
14. Merge branches in Github using Pull Requests.
15. To delete a local branch use `git branch -d NAME_BRANCH`

15 Workflow for an existing local repository that is synchronized with a remote repository on Github

15.1 Only one branch: main

15.1.1 Solo work

15.1.1.1 Repository that was not synchronized and not updated using a proper Git wrokflow

15.1.1.1.1 Part 1: Verify that the local repository is connected to Github, and the structure of the repository

1. Check that the local repository is connected to Github:

   `git remote -v`

   If it is connected, we need to see an output:

   origin https://github.com/monikavila/Github.git (fetch)

   origin https://github.com/monikavila/Github.git (push)

2. Check the current branch

   `git branch`

15.1.1.1.2 Part 2: Update repository locally and remotelly

1. Check status `git status`
2. Commit local changes

   `git add . git commit -m "Updating all the changes in one"`

   Note: Ideally, one should commit each change separately to have meaningful commit messages.

3. Pull changes

   `git pull`

4. Push changes `git push`

15.1.1.2 Repository that is constantly synchronized

15.1.1.2.1 Update repository locally and remotelly

1. Check status `git status`
2. Commit local changes

   `git add . git commit -m "Updating all the changes in one"`

   Note: Ideally, one should commit each change separately to have meaningful commit messages.

3. Pull changes

   `git pull`

4. Push changes `git push`

15.1.2 Team work

15.1.2.1 Repository that was not synchronized and not updated using a proper Git wrokflow

15.1.2.1.1 Part 1: Verify that the local repository is connected to Github, and the structure of the repository

1. Check that the local repository is connected to Github:

   `git remote -v`

   If it is connected, we need to see an output:

   origin https://github.com/monikavila/Github.git (fetch)

   origin https://github.com/monikavila/Github.git (push)

2. Check the current branch

   `git branch`

15.1.2.1.2 Part 2: Update repository locally and remotelly

1. Check status `git status`
2. Pull changes

   `git pull`
3. Commit local changes

   `git add . git commit -m "Updating all the changes in one"`

   Note: Ideally, one should commit each change separately to have meaningful commit messages.
4. Push changes `git push`

15.1.2.2 Repository that is constantly synchronized

15.1.2.2.1 Update repository locally and remotelly

1. Check status `git status`
2. Pull changes

   `git pull`
3. Commit local changes

   `git add . git commit -m "Updating all the changes in one"`

   Note: Ideally, one should commit each change separately to have meaningful commit messages.
4. Push changes `git push`

15.2 Several branches

15.2.1 Option 1: You first updated the main brannch using the workflow as the repository had only the main branch, and then you realize there are more branches

15.2.1.1 The other branch is contained in the main branch

15.2.1.1.1 Case 1: Main is ahead of the other branch

1. List all branches `git branch -a`

2. Check the commits that the other branches have and that main does not have `git log --oneline --graph main..otherbranch` If you do not get any output, then it means that the other branch does not have any commit that it not in the main branch.
3. Check the commits that main has and the other bramches do not have `git log --online --graph otherbranch..main`
4. Confirm the differences between main and other branches `git diff -stat main..otherbranch`

   `git diff -stat otherbranch..main`

   The differences are shown because we compare the files between different commits.
5. Now, let's verify the last common commit between main and the other branch

   `git merge-base main otherbranch`
6. Now, let's verify the tip of the other branch

   `git rev-parse otherbranch`

If they are equal, the other branch is fully contained into the main one and it can be safey deleted.

7. Delete the branch `git branch -d otherbranch`
8. Push the changes = delete remotely `git push origin --delete otherbranch`

15.2.1.1.2 Case 2: The other branch is ahead of main

1. List all branches `git branch -a`
2. Check the commits that the other branches have and that main does not have `git log --oneline --graph main..otherbranch`
3. Check the commits that main has and the other bramches do not have `git log --online --graph otherbranch..main` If you do not get any output, then it means that the other branch is ahead of the main branch (The other branch has commits that are not in main).
4. Confirm the differences between main and other branches `git diff -stat main..otherbranch`

   `git diff -stat otherbranch..main`

   The differences are shown because we compare the files between different commits.
5. Now, let's verify the last common commit between main and the other branch

   `git merge-base main otherbranch`

6. Now, let's verify the tip of the other branch

   ```
   git rev-parse main
   ```

If they are equal, the main is fully contained into the other branch. So we need to merge the other branch with main.

7. Change to the main branch `git checkout main`
8. Pull from remote (to be sure that main is up to date). Skip if the repository is only local. `git pull origin main`
9. Merge branches `git merge otherbranch`

16 Forking and cloning

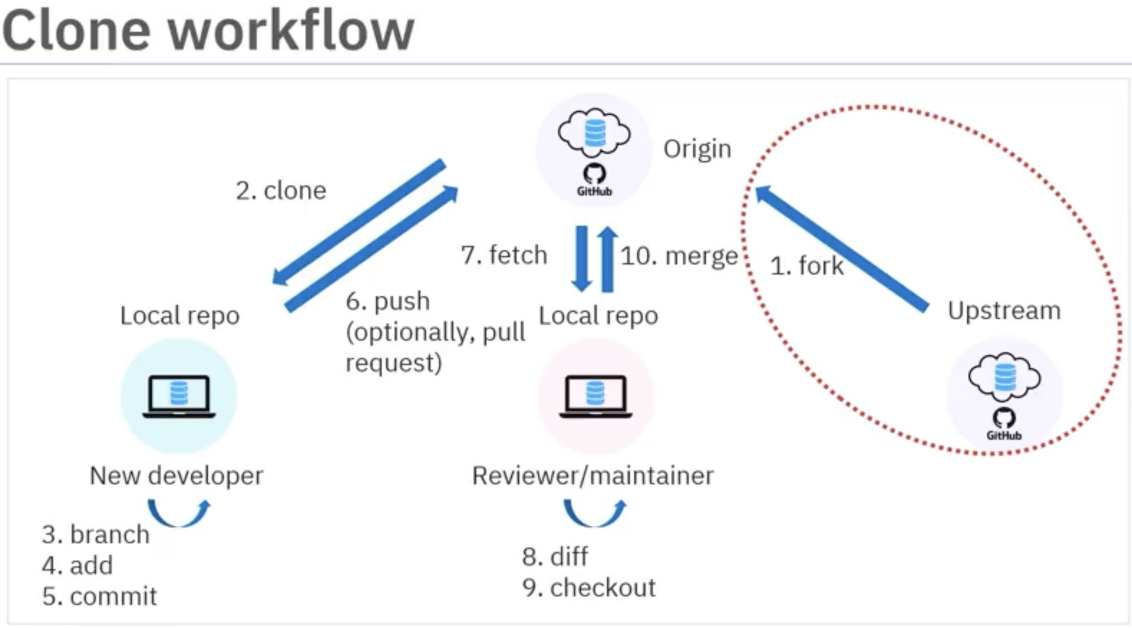
16.1 Cloning

1. Definition

- Copy of the repository in your local machine
- Cloned copies can sync between locations

2. How to clone using command line? `git clone URL`
3. Syncing local changes

- Add files to the staging area `git add files`
- Commit changes `git commit -m message`
- Transfer changes to the remote repository `git push`
- Transfer changes from the remote repo `git fetch`
- Transfer changes from the remote repo to the local repo and merge them to a branch `git pull`
- Upstream: refers to the original work
- Origin refers to your fork



Source: IBM coursera course on Getting started with Git and Github

16.2 Forking

- Creates a copy of a remote repository in GitHub.
  Modifies or extends a project without affecting the original project
- Submit changes to the original repo
- Independently make changes to a project, by submitting a pull request
- Keep a copy of the license

It can be used to:

- Work on a project independently
- Use a repository as the base for a new project

We can also use RStudio following https://happygitwithr.com/fork-and-clone.html

16.2.1 Keep syncing a Fork of a project

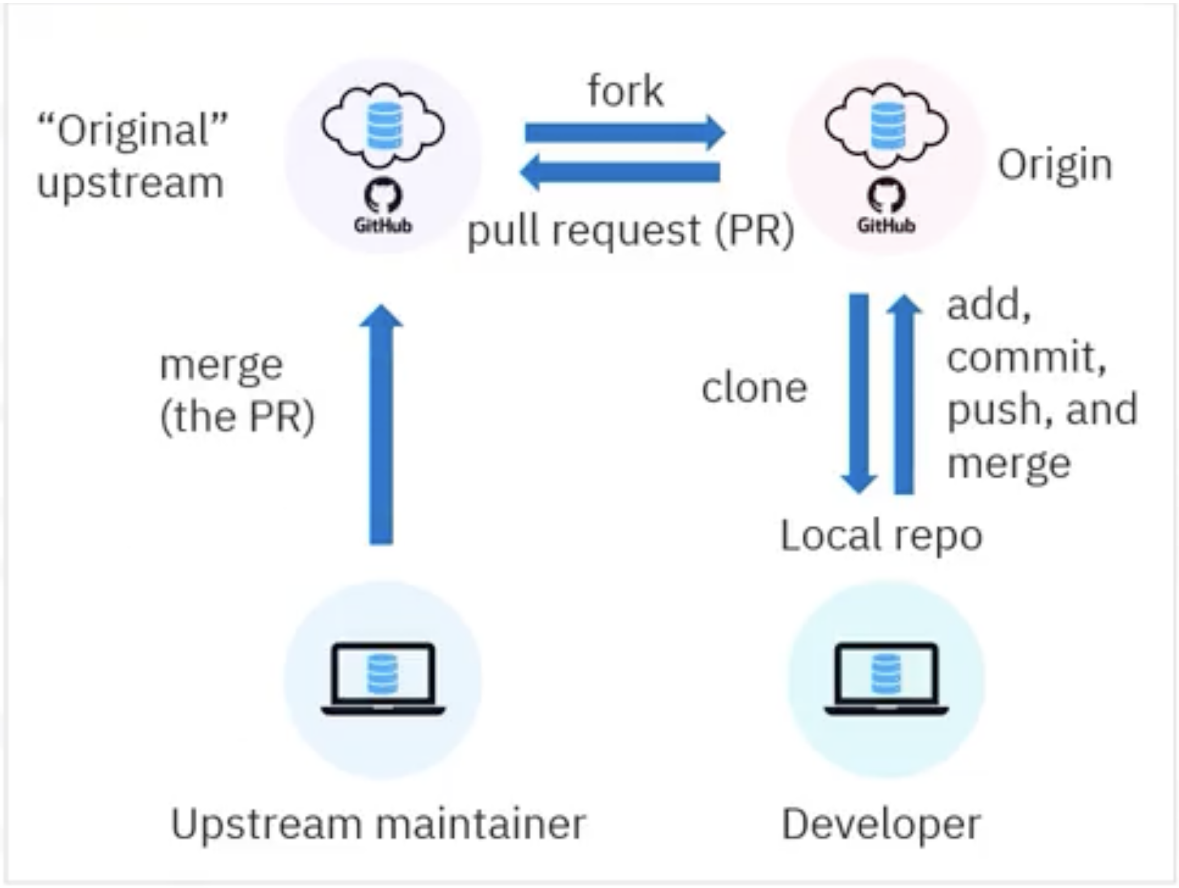
1. Create a local clone of the project `git clone URL`
2. Configure GIT to sync the fork (To understand better)

- Open a terminal and change to the directory containing the local clone
- Access the remote repository `git remote -v`
- Type `git remote add upstream <clone directory>`

16.2.2 Fork workflow

1. Fork the "Original" Github repo on github webpage
2. Clone the new Origin repo to create a local repo
3. Keep syncked the fork (set up a remote for the original)
4. Make changes in your local repo and then push them to the Origin
5. Make a pull request to send changes to the Original Forked repo

A graph of the process is given below:



Source: IBM coursera course on Getting started with Git and Github

https://srivastavayushmaan1347.medium.com/how-to-fork-a-repository-make-changes-and-submit-a-pull-request-on-github-c05b0462403d

17 Github projects

The roles are:

1. Developer
    - Uses the following commands:

    1. git clone
    2. git pull and git fetch
    3. git push

2. Integrator

    Uses the commands:

    1. git clone
    2. git pull
    3. git merge

3. Administrator
    - In charge of:
        - Managing communities
        - Managing servers

          s

18 README file

To improve README files we can see the following material:

- https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/about-readmes
- https://makeareadme.com/
- https://github.com/matiassingers/awesome-readme

19 GitHub Copilot

It is an AI-powered tool that helps writing code.

The features are:

1. Code auto-completion
2. Real time suggestions of code

Workflow with GitHub Copilot:

1. Activate GitHub Copilot.
2. Start writing code and you will get suggestions.
3. Debug code
4. Follow usual process of Git and GitHub.

20 Work strategies

1. pull before starting to work. This guarantees. that local and remote repository are synchronized.
2. Use branches.