



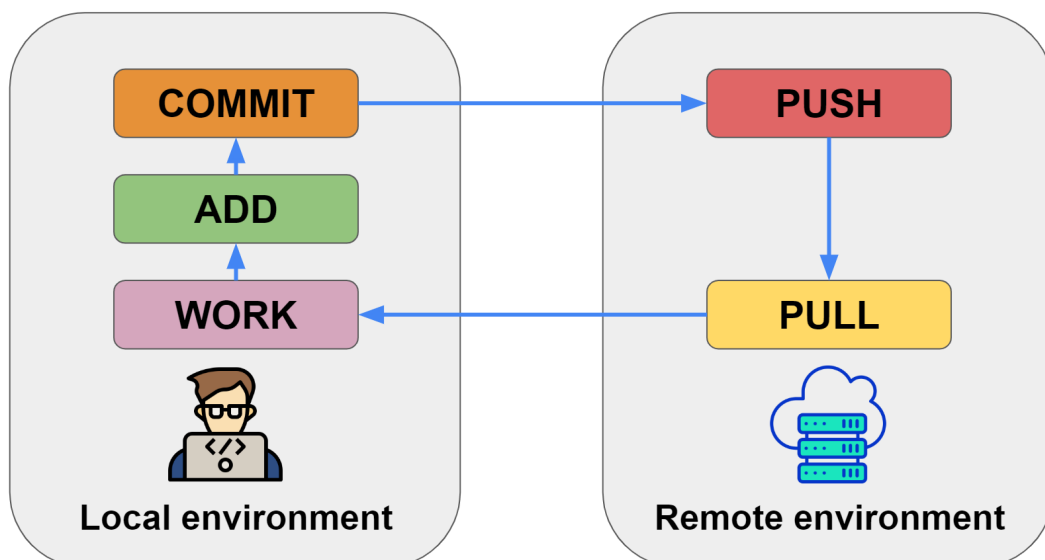
# The Git basic workflow

Now is the time to get our hands dirty with Git. At first, the Git workflow might seem a little daunting with a lot of steps but do not worry, it is only a matter of practice.

**Note:** we have introduced no Python concepts for now. Therefore, we will use Git to manipulate text files only in the following sections. Please keep in mind, that the same concepts will be applied to source code files later.

## Basic workflow

The basic Git workflow can be boiled down to five steps, as shown below: Pull, Work, Add, Commit, and Push:



Developer icon created by surang - Flaticon (<https://www.flaticon.com/free-icons/web-development>)

Cloud created by lakonicon - Flaticon (<https://www.flaticon.com/free-icons/seo-and-web>)

## Pull

Pulling means retrieving the latest version of the project from the remote repository. It is necessary to start by performing this task first before starting to work. This ensures that our work is compatible with the latest version of the code and minimizes the risk of version conflicts (a concept that will be introduced later).

To pull changes from the remote repository, go to your project folder and use the following command:

```
git pull
```

If you try it on your current project, you will get an “*Already up to date*” message since no one has modified your project in the meantime.

## Work

Now that we have the latest version of our files, we can start performing our modifications.

- Add the sentence “This is the first modification in this project.” to the **README.md** file. Also create two files with some text of your choice: **file1.txt** and **file2.txt**.
- ▼ Before saving any modifications in the Git repository, you should ALWAYS start by visualizing your local changes. Use the following command:

```
git status
```

- ▼ Observe the Git response:

```
C:\Users\33611\Documents\Formations\Cap4Lab\git-workflow\git-course>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

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

Git has detected that you modified the file README.md. As far as the two other text files are concerned, they are still unknown to your repository, hence, they are under the **Untracked files** section. It is also highly recommended to review your changes inside the **modified** files in a fine-grained fashion.

▼ Use the following command to visualize what changed in README.md:

```
git diff README.md
```

▼ Observe the output. Here you can see that an empty line, as well as a new sentence, were added:

```
C:\Users\33611\Documents\Formations\Cap4Lab\git-workflow\git-course>git diff README.md
diff --git a/README.md b/README.md
index 78ed57c..2908565 100644
--- a/README.md
+++ b/README.md
@@ -90,3 +90,5 @@ For open source projects, say how it is licensed.

## Project status
If you have run out of energy or time for your project, put a note at the top of the README saying that development has
slowed down or stopped completely. Someone may choose to fork your project or volunteer to step in as a maintainer or c
owner, allowing your project to keep going. You can also make an explicit request for maintainers.
+
+This is the first modification in this project.
\ No newline at end of file
```

In some cases, we will want to ignore local modifications and go back to the version stored in the repository.

▼ Retrieve the version of README.md stored in the repository using the following command:

```
git checkout README.md
```

If you use **git status** again, README.md will no longer appear in the output. You can also open the file to check that the sentence you have added has disappeared.

**Important:** when doing so, you will lose your local modifications! If you want to save them for later, while going back to the current version in the repository, you can use git stash.

Before going further, make sure to add the sentence back to README.md!

## Add

Once you have worked on the project and tested your work, it is time to select which files and modifications you want to save into the repository. In the Git lingo, it is said that you are **staging** your modifications, meaning that you are preparing them to be saved in the version history of your repository.

This phase is important since you might not want all your modifications to be taken into account. You can stage all your modifications at once using the following command:

```
git add --all
```

Or you can choose to stage all the modifications in a subfolder of your project using:

```
git add subfolder_path
```

Nevertheless, WE DO NOT RECOMMEND using such commands unless you really know what you are doing. Instead, try to add your modifications file by file after performing a **git diff** on each one (except untracked files). The command to add a single file is the following:

```
git add file_path
```

▼ Suppose we want to ignore file2.txt. Add the two files we want to stage one by one:

```
git add README.md
git add file1.txt
```

▼ Visualize the staged files using **git status** (notice how they appear in green and no longer in red):

```
C:\Users\33611\Documents\Formations\Cap4Lab\git-workflow\git-course>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt
```

If you want to undo your staging, you can use the following command:

```
git reset
```

You can also unstage a single file:

```
git reset file_path
```

▼ Unstage README.md and stage it again:

```
git reset README.md
git status
git add README.md
```

**Note:** you can also stage chunks of a file only using **git add file\_path -p**. However, this is an advanced feature and it is out of the scope of this course.

## Commit

So far, we have used the word “save” to denote the fact that we want to add changes to the history of the modifications in our repository. In the Git lingo, we say that we **commit** our work. Also, every commit should be associated with a message that clearly describes the associated modifications.

Here is the command to commit the staged changes:

```
git commit -m "your_message"
```

We encourage you to adopt the following Semantic Commit Messages to write your messages.

▼ Commit the changes staged so far using the message “docs: enrich README.md and add text file”:

```
git commit -m "docs: enrich README.md and add text file"
```

**Important:** so far, your work has been committed to your local repository, NOT THE REMOTE ONE. You still get a chance to change the last commit if needed using git commit amend.

## Push

▼ Visualize the status of your repository using **git status**:

```
C:\Users\33611\Documents\Formations\Cap4Lab\git-workflow\git-course>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
       file2.txt

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


Git tells you that your local repository is ahead of the remote version by one commit. You are invited to “push” your work to publish it, for others to be able to see it.

▼ Push your work using the following command:


```
git push
```







At this point, you can view your changes on GitLab.


▼ Go to the page associated with your project on GitLab (or reload it). You can see your published changes:






docs: enrich README.md and add text file  
Hatim Khouzaimi authored 7 minutes ago

9ad0efa3 

 README
  Add LICENSE
  Add CHANGELOG
  Add CONTRIBUTING
  Add Kubernetes cluster
  Set up CI/CD

 Configure Integrations

Name	Last commit	Last update
 README.md	docs: enrich README.md and add text file	7 minutes ago
 file1.txt	docs: enrich README.md and add text file	7 minutes ago

 README.md

## Git Course

---

### Getting started

---

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

## The .gitignore file

Notice that we are not interested in committing nor tracking the modifications happening inside `file2.txt`. However, it is still here polluting our **git status** output. Moreover, if we stage it by accident, it will get committed. There exists a feature in git that makes it possible to declare these kinds of files so that they can be ignored by Git.

To do so, you should create a file called `.gitignore` inside the folder of your project. Then fill it with the list of files you want to ignore. You can also use an advanced syntax to ignore files that respect a certain pattern and to organize your file. More about this [here](#).

- Create a `.gitignore` file with “`file2.txt`” as a content.
  - Check the output of **git status**
  - Notice how **file2.txt** disappeared from the list of untracked files.
- ▼ However, `.gitignore` has to be committed itself. Commit it using the message “chore: add `.gitignore`”. Also, don’t forget to push this new modification:

```
git add .gitignore
git commit -m "chore: add .gitignore"
git push
```