# Python Project Tutorial 2

## An introduction to Git

*Git* is a popular form of version control which offers a simple way to alter source code when working on collaborative projects. Once you have installed Git, you can check the version of Git on your system by executing this line of code in the command line:

```
In [ ]: git --version
```

You can create a local Git repository on your system, where changes to a project can be easily tracked and saved (the data will be stored in a .git directory). There are two types of Git repositories:

- Bare repositories: developers share project changes to a wider team, where individual users are unable to modify or create new versions of the repository.
- Non-bare repositories: users are allowed to modify or create new versions of the repository.

To create a git repository, you can open the *Git Bash terminal* and execute the following:

```
In [ ]: cd ~/Desktop # change directory to Desktop (for example)
        mkdir myproject # make new project directory here
        cd myproject/ # move to the project directory
```

In addition to this, you can make a git repository in the project folder with the command:

```
In [ ]: git init [repository-name]
```

## An introduction to Github

*Github* is a cloud based repository where developers can manage and track changes to source codes they deposit. This can be very useful in events where you want to make changes to source code that is being used by your collaborators, but you do not have access to your personal device. Once you have made a Github account, you can either create a Github repository directly, or you can *push* a repository from your personal device to Github. To do this, Git refers to different stages of development of the personal repository as follows:

- A *committed* state refers to a file with all changes saved to the local repository.
- A *modified* state refers to a file with changes made, but they have yet to be saved (this is an alteration of a previous committed state).
- A file in a *staged* state means that it is ready to become a committed state.

To track changes to a file using Git, we can use:

```
In [ ]: git add . # applies to all files in the repository
        git add file_name # applies to a single file
        git status # checks the current status
```

If in a staged state, then you can commit the file (if in a modified state, this cannot be done):

```
In [ ]: git commit -m "comment something here" # -m means message
```

Then, the local repository can be pushed to Github by executing the lines of code:

```
In [ ]: # Connect local repository with remote Github repository:
        git remote add origin https://github.com/username/repository_name.git

        # Change default branch name to 'main':
        git branch -M main

        # Push repository from local device to Github:
        git push -u origin main
```

## Branches in Git

Branches are a way to copy a file and work on it without making changes to the original. This gives you the option to either merge the chanages to the original file, or keep the copied file as an independent version. To create a new branch, simply execute the line:

```
In [ ]: git checkout -b new_branch # checkout tells Git to switch to a new branch
                                   # -b tells Git to create a new branch
                                   # new_branch is the name of the created branch to switch to

        # Alternatively, you can use:
        git branch new_branch
```

You can check all the existing branches in your repository by running:

```
In [ ]: git branch # Lists all branches in repository, with *
                   # telling you the one you are currently in
```

To move from new_branch to the main branch, run:

```
In [ ]: git checkout main

        # Alternatively, you can use:
        git switch main
```

And to merge changes in new_branch to the main branch, write the following:

```
In [ ]: git merge new_branch
```

If there are not problems, then this will be achieved, however, if there are conflicts, you can run the following to see:

```
In [ ]: git diff
```

Once the conflicts have been resolved, you can merge again. To see a graphical history of the sequence of events you have carried out using Git, you can run:

In [ ]:
```
gitk
```

Once the branches have been merged, you can delete new_branch as follows and push the main repository to Github:

In [ ]:
```
git branch -d new_branch
git switch main
git remote add origin https://github.com/username/repository_name.git
git push -u origin main
```

## Pull requests

The *pull* command in Git clones the current state of a repository onto a local repository on your personal device. For example, if you have a clone a Github repository called repository_name, then this will be the name of the repository on your local device. This can be achieved as follows:

In [ ]:
```
git clone Github_url # Github_url refers to the URL of the repository on Github
```

Once you have done this, you can modify and commit changes to the cloned repository, and then push the repository back to Github. This could look something like:

In [ ]:
```
# Track changes to file1, for example:
git add file1

# Following this, modify one of the files, then:
git commit -m "some text here describing changes to file1"

# Push respository onto Github:
git push -m origin main
```

From this, you can create a pull request, where you get people to review your changes. To do this, you must make them a collaborator for the repository (go to repository settings for this), and then add people as reviewers for the pull request. Once done, this will allow a discussion to form, and when agreement is achieved, the pull request can be merged.

In [ ]:
```

```