

First steps with Git

Jakob Willforss

2020-10-11

Introduction

This tutorial aims to get your feet wet using the Git terminal to manage your files locally and understand how to send these to GitHub.

To illustrate each step, I use the visualizations provided by the web page Visualizing Git. Highly useful for understanding Git and what it does!

Downloading Git

NOTE: This is only required if you don't already have git installed on your computer. You can check this by running `git --version`. If installed properly, this should give you a version message.

```
git --version  
> git version 2.17.1
```

If installed properly, proceed to **Creating a new repository**.

If not installed, navigate to <https://git-scm.com/> and find the download for the system you are using. Carry through the installation.

Creating a new repository

When we start a new project, we start creating a new folder (`mkdir PythonHelloProject`). At this point, Git is not involved - we have only made a new folder in our file tree.

We navigate into the folder, and then initialize a new repository using the command `git init`. This action creates a hidden folder `.git` inside the `PythonHelloProject`. Now Git will track what is happening within this folder.

```
mkdir PythonHelloProject  
cd PythonHelloProject  
git init  
  
> Initialized empty Git repository in /home/jakob/PythonHelloProject/.git/
```

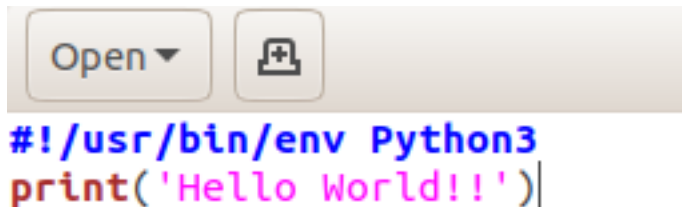
Creating and adding your first two files

Here, we will go through the steps of adding new files to our repository. This is a three-step procedure:

1. Create a new file in the file tree
2. Add the newly created files to the stage, selecting those to include in the next commit
3. Commit the staged files, creating a snapshot where these files have been included

Including the first file in the history

First, we create a Python hello-world script. You can open the file in any text editor and add the lines. The file is made from the terminal here for demonstration purposes.



We can inspect how the file looks. At this point, we have created a new file on the computer. Git is not involved yet.

```
cat hello_python.py
```

```
> #!/usr/bin/env Python3
> print('Hello World!!')
```

We can always check with `git` what is the status of the repository. Here, we see that Git notices that there are new files added, which are not part of the current history.

```
git status
```

```
> On branch master
>
> No commits yet
>
> Untracked files:
>   (use "git add <file>..." to include in what will be committed)
>
>   hello_python.py
>
> nothing added to commit but untracked files present (use "git add" to track)
```

We add this file to the **staging area** using the command `git add`. This command is used to select what file or files to be included in the next commit.

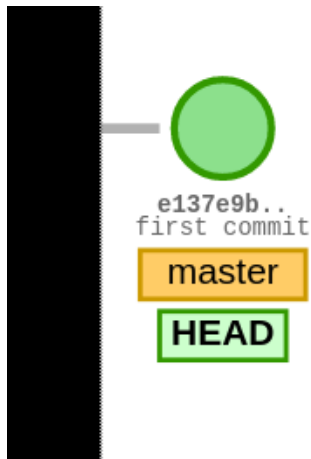
```
git add hello_python.py
```

We are ready to create a commit and make new history.

```
git commit -m "first commit"
```

```
> [master (root-commit) ee89d19] first commit
> 1 file changed, 2 insertions(+)
> create mode 100644 hello_python.py
```

At this point, our Git history looks the following way:

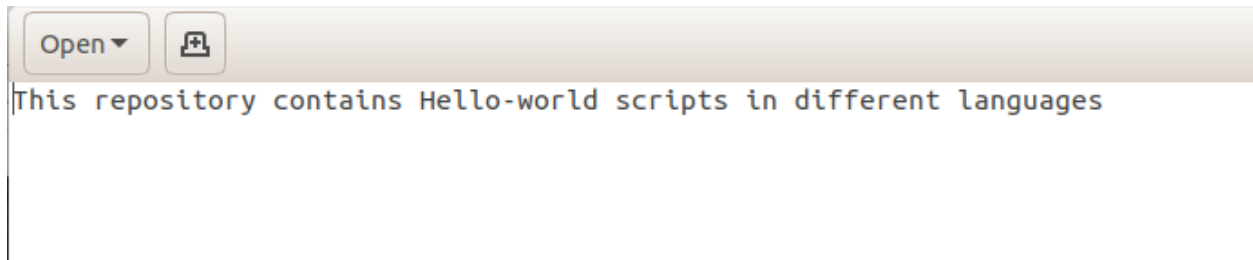


Creating and including the second file

Secondly, we also include a README file.

We create a new file, either as the command below or through a text editor.

```
echo "This repository contains Hello-world scripts in different languages" > README.md
```



Next, we add the second file to the stage, preparing it to be committed.

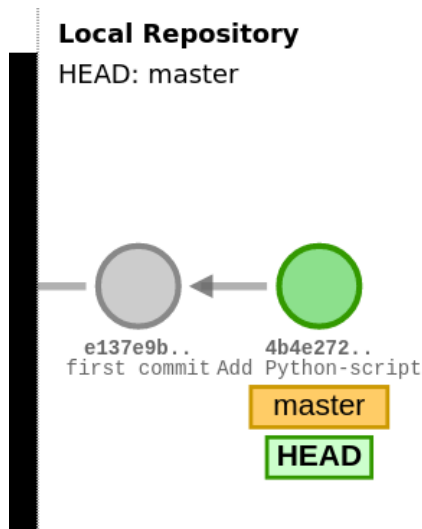
```
git add README.md
```

We create a second commit, including this file in the Git history.

```
git commit -m "Add README"
```

```
> [master 5724d25] Add README
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
```

The repository at this stage can be illustrated in the following way. Here, the **HEAD** points to the version we are looking into, and the **master** (or **main**) points to our main branch.



We can inspect the history at this point using the command `git log`. Compare the output to what is shown in the figure above.

```

git log
> commit 5724d2592a13b32ca7256d8cd4cb31a275f9af22 (HEAD -> master)
> Author: Jakob Willforss <jakob.willforss@hotmail.com>
> Date: Sun Oct 11 15:57:33 2020 +0200
>
> Add README
>
> commit ee89d19ee6eab220f92f37f2016ca9cf34f7b08f
> Author: Jakob Willforss <jakob.willforss@hotmail.com>
> Date: Sun Oct 11 15:56:59 2020 +0200
>
> first commit
  
```

Updating existing files

Now we know how to add newly created files to the Git repository. Beyond this, we can also commit *modifications* to the history and the *removal* of files. Here, I will demonstrate how we can commit changes made to a file to the history.

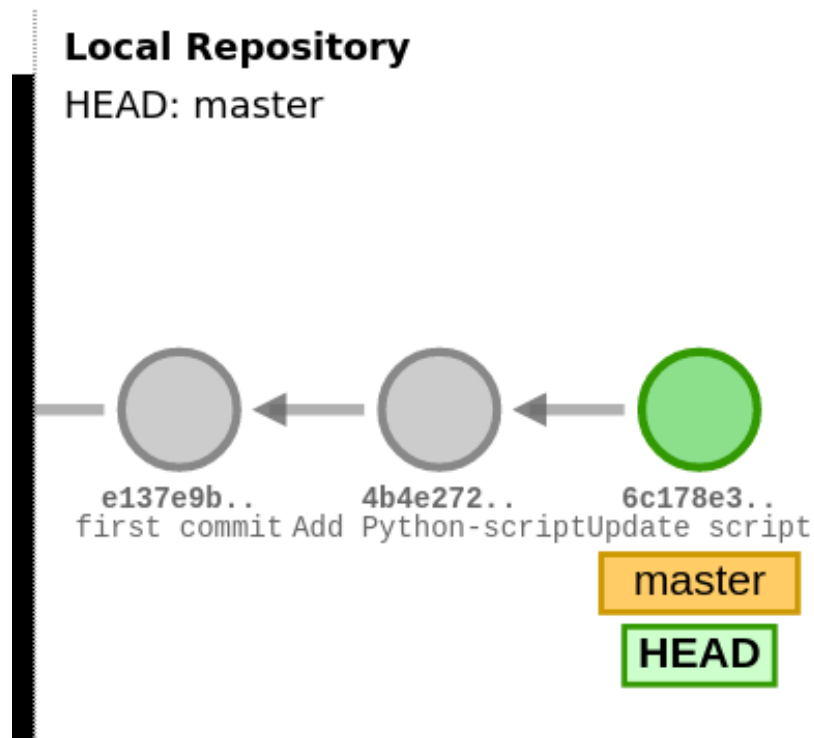
We update the existing Python-script by appending a second line - saying “bye”.

```
echo "print('That's all, bye!')" >> hello_python.py
```

```
#!/usr/bin/env Python3
print('Hello World!!')
print('That's all, bye!')
```

As before, check the status.

```
git status
> On branch master
> Changes not staged for commit:
>   (use "git add <file>..." to update what will be committed)
>   (use "git checkout -- <file>..." to discard changes in working directory)
>
>   modified:   hello_python.py
>
> no changes added to commit (use "git add" and/or "git commit -a")
git add hello_python.py
```



Moving back and forth in history

One of the powerful features of Git is that we, at any point, can go back to any previous snapshots. By doing this, the **file tree** (the files we see on our computer) changes into the state the files were in at that point. This allows us to, for instance, explore how our scripts looked before introducing bugs.

By using the command `git log` we can see the history, and the commit IDs related to each step.

```
git log
> commit 5724d2592a13b32ca7256d8cd4cb31a275f9af22 (HEAD -> master)
> Author: Jakob Willforss <jakob.willforss@hotmail.com>
> Date:   Sun Oct 11 15:57:33 2020 +0200
>
>   Add README
>
> commit ee89d19ee6eab220f92f37f2016ca9cf34f7b08f
```

```
> Author: Jakob Willforss <jakob.willforss@hotmail.com>
> Date: Sun Oct 11 15:56:59 2020 +0200
>
> first commit
```

Using the first part of the ID is one way to access that commit (for instance 5724d25, normally a few letters is enough, but if using too little, for instance 572, Git may not be able to uniquely identify the commit).

Next, you can checkout the previous commit.

```
git checkout ee89d19e
> M hello_python.py
> Previous HEAD position was 5724d25 Add README
> HEAD is now at ee89d19 first commit
```

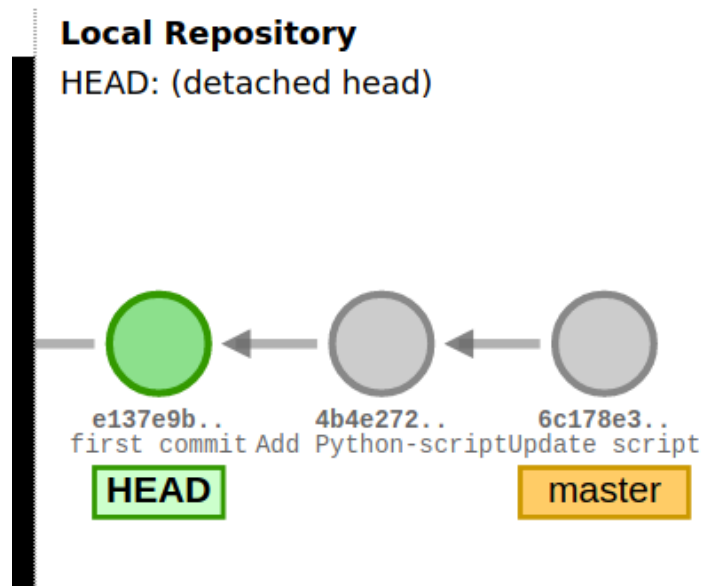


Figure 1: Git checkout

If we check the log from here, the history looks different. This is because each commit only is aware of what has happened before it.

```
> commit ee89d19ee6eab220f92f37f2016ca9cf34f7b08f (HEAD)
> Author: Jakob Willforss <jakob.willforss@hotmail.com>
> Date: Sun Oct 11 15:56:59 2020 +0200
>
> first commit
```

At any point, we can navigate back to the latest commit by using the command `git checkout` and specifying the head of the branch (either `master` or `main`).

```
git checkout master
> M hello_python.py
> Previous HEAD position was ee89d19 first commit
> Switched to branch 'master'
```

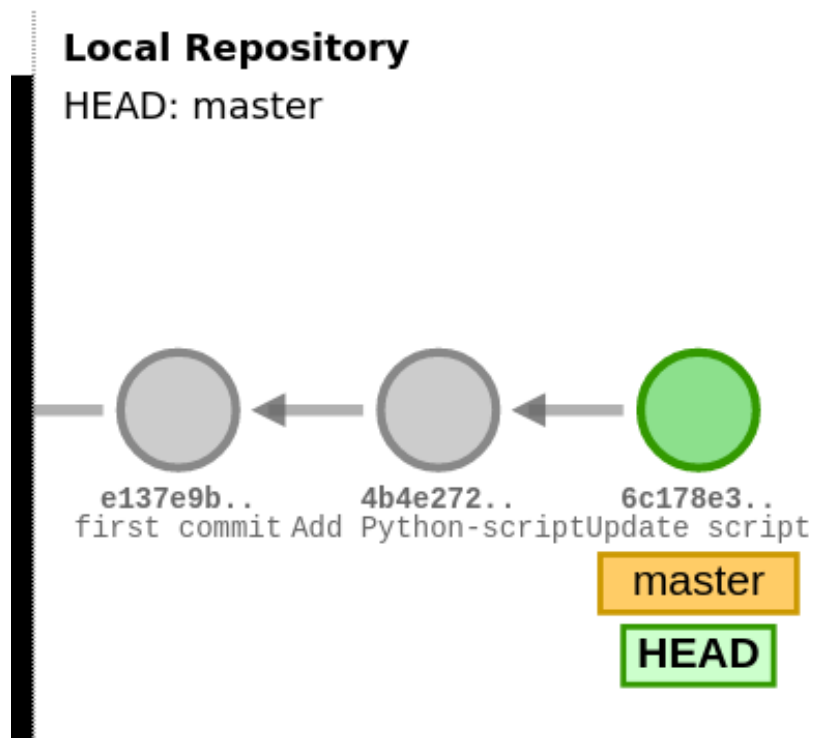


Figure 2: Commit 3


Sending a repository to GitHub

Setup a new repository on GitHub. Give it a name, and assign it as public. Don't add the README this time! (As you already have created one.)

Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * **Repository name ***

 Jakob37 ▾ / HelloPython ✓

Great repository names are short and memorable. Need inspiration? How about **silver-waddle?**

Description (optional)

-
- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.
-

Initialize this repository with:

Skip this step if you're importing an existing repository.

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)
- ☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
- ☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)
-

Create repository

You will come to a page similar to the following. Follow the instructions for 'push an existing repository from the command line'

Quick setup — if you’ve done this kind of thing before

or

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# HelloPython" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:Jakob37/HelloPython.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:Jakob37/HelloPython.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Note that there are two protocols for linking GitHub repositories - https and ssh. Using ssh allows you to interact with GitHub without using a password every time, but requires you to set it up properly. Instructions for this are found [here](#). Using the https protocol works well, so don't worry about it.

Adapt the command below accordingly - Replace “Jakob37” with your user and “HelloPython” with the name of your repository!

```
git remote add origin https://github.com/Jakob37/HelloPython.git
git branch -M main
git push -u origin main
```

Concluding words

This tutorial only provides a teaser for the functionality of Git. By putting some more time into working through a workshop, you will have a powerful and useful tool at your disposal. I would strongly encourage you to start using Git in your next coding- or analysis-project, and gradually get used to it. When proficient, you will have the full benefit of Git with very little effort.

Some resources for further reading:

- My materials
- Free and interactive online tutorial to get started with Git: [link](#)
- Non-free, but great and comprehensive tutorial at DataCamp: [link](#)
- For a deeper understanding of what goes on ‘under the hood’, I recommend this post: [link](#)