Software Engineering For Data Science (SEDS)

Class: 2 Year 2nd Cycle
Branch: AIDS

Dr. Belkacem KHALDI| ESI-SBA

Lecture 04:

Advanced Concepts for Python Software Engineering: Unit Testing, Git, and Github

Advanced Concepts for Python Software Engineering: Unit Testing, Git, and Github

1. Unit Testing
2. Git
3. GitHub

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Unit Testing

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

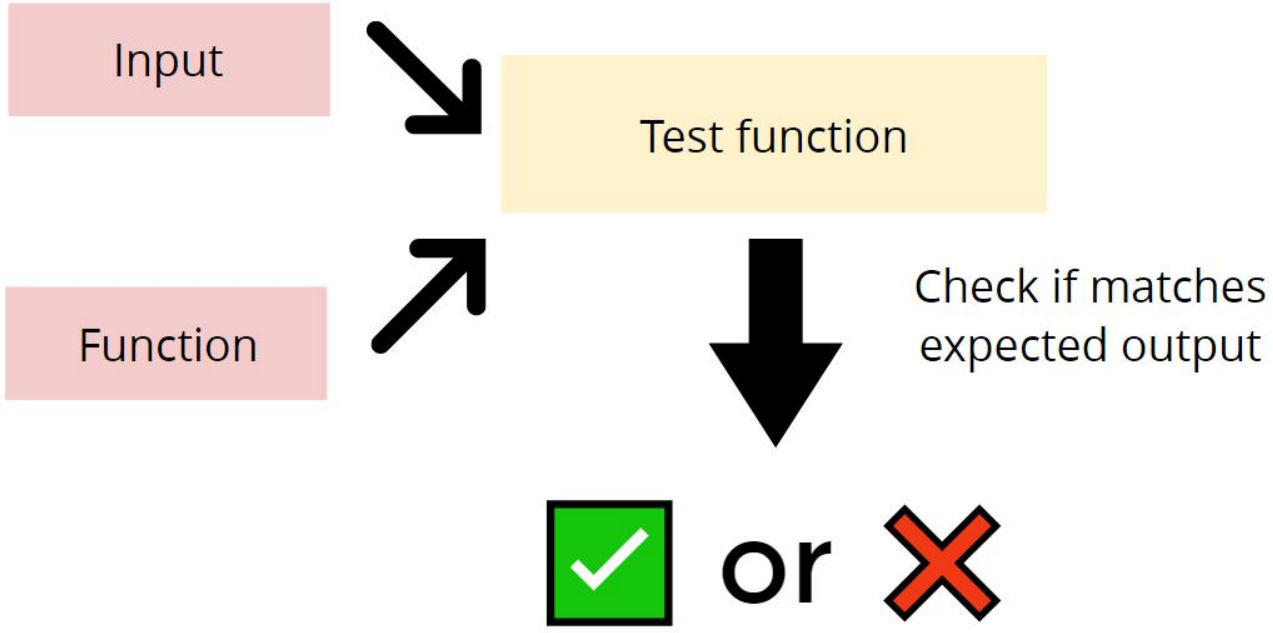
Unit Testing

What is Unit Testing?

- You have a program structured in **units**: e.g. **functions, classes, modules**.

- You want to make sure a **unit** matches the expected outputs.

- **Unit Testing** ⇒ A **Software Testing Method** by which individual units of **source code**… are tested to determine whether they are fit for use. en.wikipedia.org/wiki/Unit_testing



Unit Testing

How can we test an implementation?

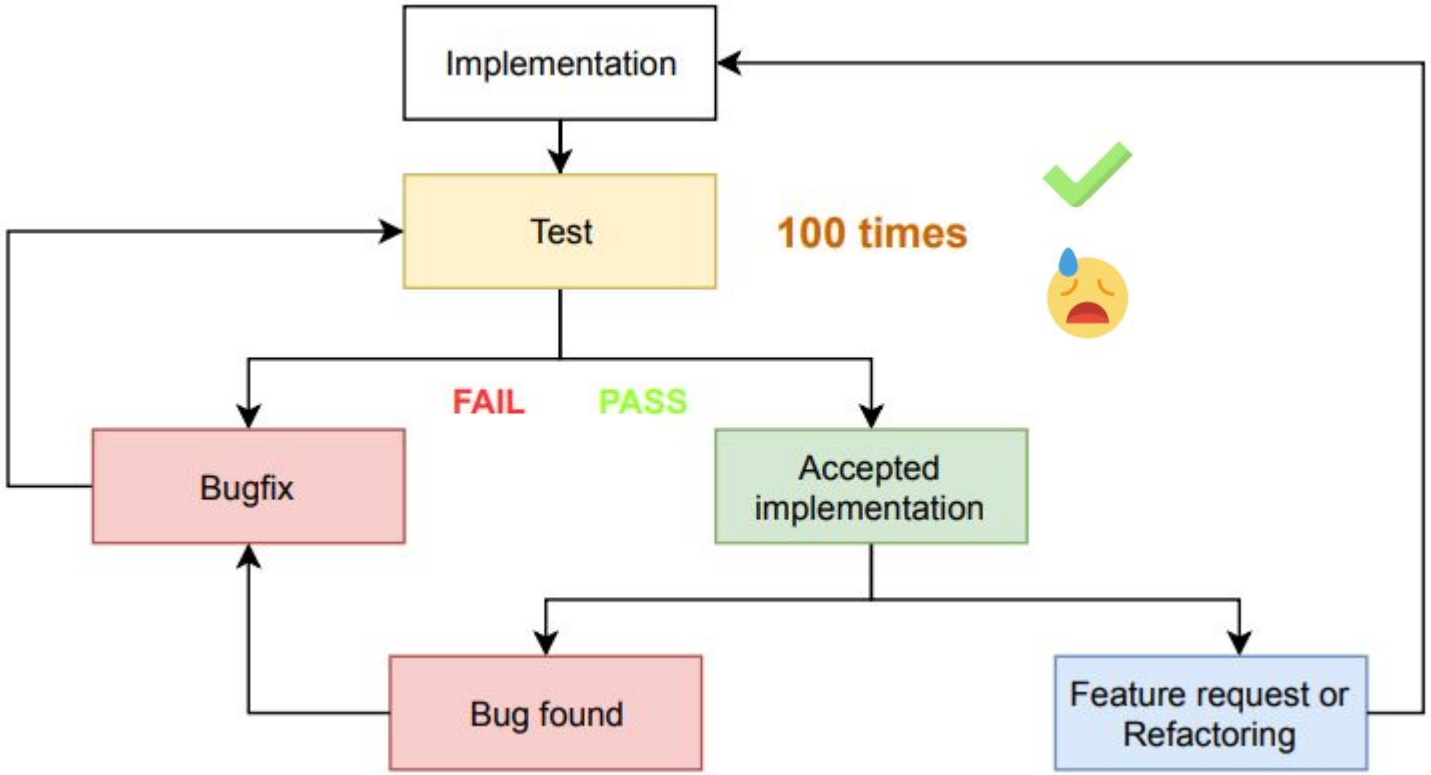
Implementation

```
def my_function(argument):
    " do something and return result"
```

Test, Bugfix, and Acceptance

```
my_function(argument1)
return_value_1

my_function(argument2)
return_value_2

my_function(argument3)
return_value_3
```

But How many tests ?



Unit Testing

Time spent in testing a Function

Manual Testing

5 mins x 100 ~

8 hours

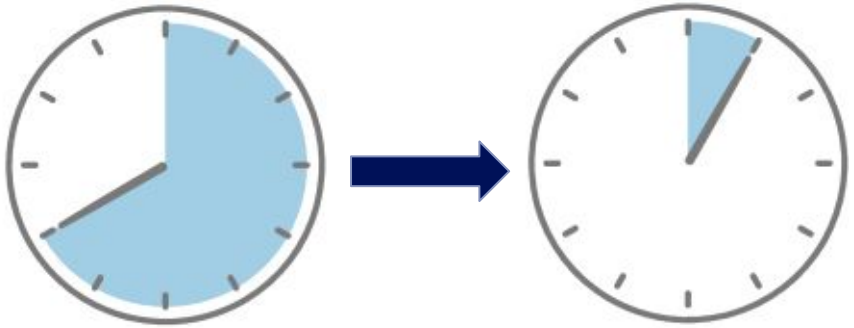
Thanks to Python Community

Automatic Testing

tox

pytest

unittest



8 hours

1 hour

Unit Testing

Pytest

We will use **pytest**

- Has all essential features.
- Easiest to use.
- Most popular.

https://docs.pytest.org/en/7.2.x/

Installation using Conda

```
conda install pytest
```

Installation using pip

```
pip install  pytest
```

Installation Confirmation

```
pytest --version
```

Unit Testing

Pytest

Basic Test functions

my_test_file.py

```python
import pytest

def serve_beer(age):
    if (age is None) or (age<18):
        return "No beer"
    else:
        return "Have beer"

def test_serve_beer_legal():
    adult = 25
    assert serve_beer(adult) == "Have beer"

def test_serve_beer_illegal():
    child = 10
    assert serve_beer(child) == "No beer"
```

Method to test

Test Methods

Basic Pytest Commands

Run tests in a module

```
pytest <test_file_name>
```

```
========================================== test session starts ==========================================
platform win32 -- Python 3.11.0, pytest-7.2.0, pluggy-1.0.0
rootdir: C:\Users\user\.spyder-py3
collected 2 items

temp.py ..                                                                                        [100%]

========================================== 2 passed in 0.02s ===========================================
```

Run a specific test within a module

```
pytest <test_file_name>::<test_func_name>
```

```
========================================== test session starts ==========================================
platform win32 -- Python 3.11.0, pytest-7.2.0, pluggy-1.0.0
rootdir: C:\Users\user\.spyder-py3
collected 1 item

temp.py .                                                                                         [100%]

========================================== 1 passed in 0.02s ===========================================
```

Run tests in a folder

```
pytest <folder_name>/
```

More details on how to invoke Pytest commands are available in:
https://docs.pytest.org/en/7.1.x/how-to/usage.html

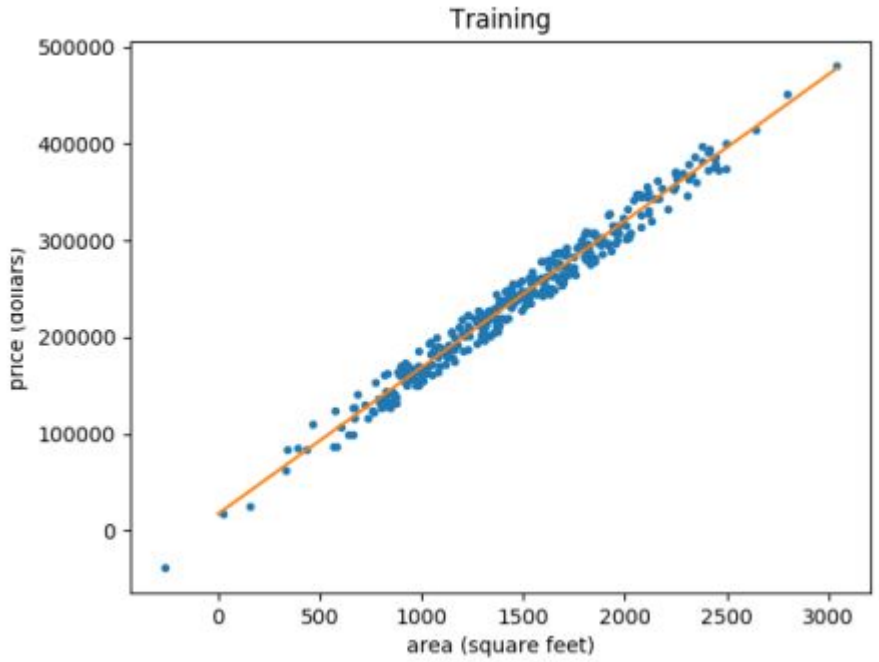
Unit Testing

Pytest

Learn Unit Testing - with a data science spin

Price	SqFt	Bedrooms	Bathrooms
114300	1790	2	2
114200	2030	4	2
114800		3	2
94700	1980	3	2
119800	2130	3	3
114600	1780	3	2
151600	1830	3	3
150700	2160	4	2
119200	2110	4	2
104000		3	3
132500	2030	3	2
123000	1870	2	2
102600	1910	3	2
126300	2150	3	3
176800	2590	4	3
145800	1780	4	2
147100	2190	3	3
83600	1990	3	3
111400	1700	2	2
167200	1920	3	3
116200	1790	3	2
113800	2000	3	2
	1690	3	2

Dataset Not Clean



Linear regression of housing price against area

Modularity: Object-Oriented Programming

Develop a complete unit test suite

```
src/
|-- data/
|-- features/
|-- models/
|-- visualization/
```

Recommended to Create a Full copy of src/ as tests/

```
tests/ # Test suite
|-- data/
|-- features/
|-- models/
|-- visualization/
```

Write unit tests for your own projects.

Cookiecutter Data Science Project Structure

```
$ cookiecutter https://github.com/drivendata/cookiecutter-data-science
```

Ce PC › Disque local (C:) › Utilisateurs › user › Cookiecutter-ds-prjct

Nom	Modifié le	Type
data	01/11/2022 9:34 AM	Dossier de fichiers
docs	01/11/2022 9:34 AM	Dossier de fichiers
models	01/11/2022 9:34 AM	Dossier de fichiers
notebooks	01/11/2022 9:34 AM	Dossier de fichiers
references	01/11/2022 9:34 AM	Dossier de fichiers
reports	01/11/2022 9:34 AM	Dossier de fichiers
src	01/11/2022 9:34 AM	Dossier de fichiers
.env	01/11/2022 9:34 AM	Fichier ENV
.gitignore	01/11/2022 9:34 AM	Document texte
LICENSE	01/11/2022 9:34 AM	Fichier
Makefile	01/11/2022 9:34 AM	Fichier
README	01/11/2022 9:34 AM	Fichier source Mar...
requirements	01/11/2022 9:34 AM	Document texte
setup	01/11/2022 9:34 AM	Fichier source Pyt...
test_environment	01/11/2022 9:34 AM	Fichier source Pyt...
tox	01/11/2022 9:34 AM	Paramètres de co...

Unit Testing

Unit Test Example

Module: test_row_to_list.py

```python
import pytest
import row_to_list

def test_for_clean_row():
    assert row_to_list("2,081\t314,942\n") == \
                        ["2,081", "314,942"]

def test_for_missing_area():
    assert row_to_list("\t293,410\n") is None

def test_for_missing_tab():
    assert row_to_list("1,463238,765\n") is None
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`
`"\t293,410\n"`	Invalid	None
`"1,463238,765\n"`	Invalid	None

Module: row_to_list.py

```python
def row_to_list(s):
    return list(s.split())
```

Running unit tests

```
pytest test_row_to_list.py
```

Unit Testing

Understanding test result report

```
============================= test session starts ==============================
platform win32 -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: C:\Users\user
plugins: anyio-3.5.0
collected 3 items

test_row_to_list.py .FF                                                  [100%]

================================= FAILURES =================================
___________________________ test_for_missing_area ___________________________

    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
E       AssertionError: assert ['293,410'] is None
E        +  where ['293,410'] = row_to_list('\t293,410\n')

test_row_to_list.py:14: AssertionError
___________________________ test_for_missing_tab ____________________________

    def test_for_missing_tab():
>       assert row_to_list("1,463238,765\n") is None
E       AssertionError: assert ['1,463238,765'] is None
E        +  where ['1,463238,765'] = row_to_list('1,463238,765\n')

test_row_to_list.py:17: AssertionError
========================== short test summary info ==========================
FAILED test_row_to_list.py::test_for_missing_area - AssertionError: assert ['...
FAILED test_row_to_list.py::test_for_missing_tab - AssertionError: assert ['1...
========================== 2 failed, 1 passed in 0.22s ======================
```

General Information

Test Summary Result

Test Detailed Failures Results

Short Test Summary Info

Unit Testing

Understanding test result report – Test Summary Result

```
collected 3 items

test_row_to_list.py .FF                                                  [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.
.	Passed	No exception raised when running unit test.	Everything is fine. Be Happy!

Unit Testing

Understanding test result report – Test Detailed Failures Results

```
_____________________________ test_for_missing_area _____________________________

    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
E       AssertionError: assert ['293,410'] is None
E        +  where ['293,410'] = row_to_list('\t293,410\n')
```

- The line raising the exception is marked by >
- The exception is an **AssertionError**
- The line containing **where** displays **return values.**

Unit Testing

Understanding test result report – Short Test Summary Info

```
========================== short test summary info ==========================
FAILED test_row_to_list.py::test_for_missing_area - AssertionError: assert ['...
FAILED test_row_to_list.py::test_for_missing_tab - AssertionError: assert ['1...
========================== 2 failed, 1 passed in 0.22s ==========================
```

- Result summary from all unit tests that ran: **2 failed, 1 passed** tests.
- Total time for running tests: **0.22** seconds.
  - Much faster than testing on the **interpreter**!

Unit Testing

Running multiple test functions

```
@pytest.mark.parametrize('n', range(5))
def test_for_clean_row(n):
    assert row_to_list("2,081\t314,942\n") == ["2,081","314,942"]
```

```
In [8]: !pytest test_row_to_list.py::test_for_clean_row
============================= test session starts ==============================
platform win32 -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: C:\Users\user
plugins: anyio-3.5.0
collected 5 items

test_row_to_list.py .....                                                [100%]

============================== 5 passed in 0.12s ===============================
```

Git

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Git

- **What is Git?**
  - A version-control system for tracking changes in your code.
  - Developed in 2005 by Linus Torvalds
  - Used for coordinating work on files among multiple people.
    - Who wrote this module?
    - When was this function edited? By whom? Why was it edited?
    - Over the last 1000 revisions, when/why did a particular unit test stop working?

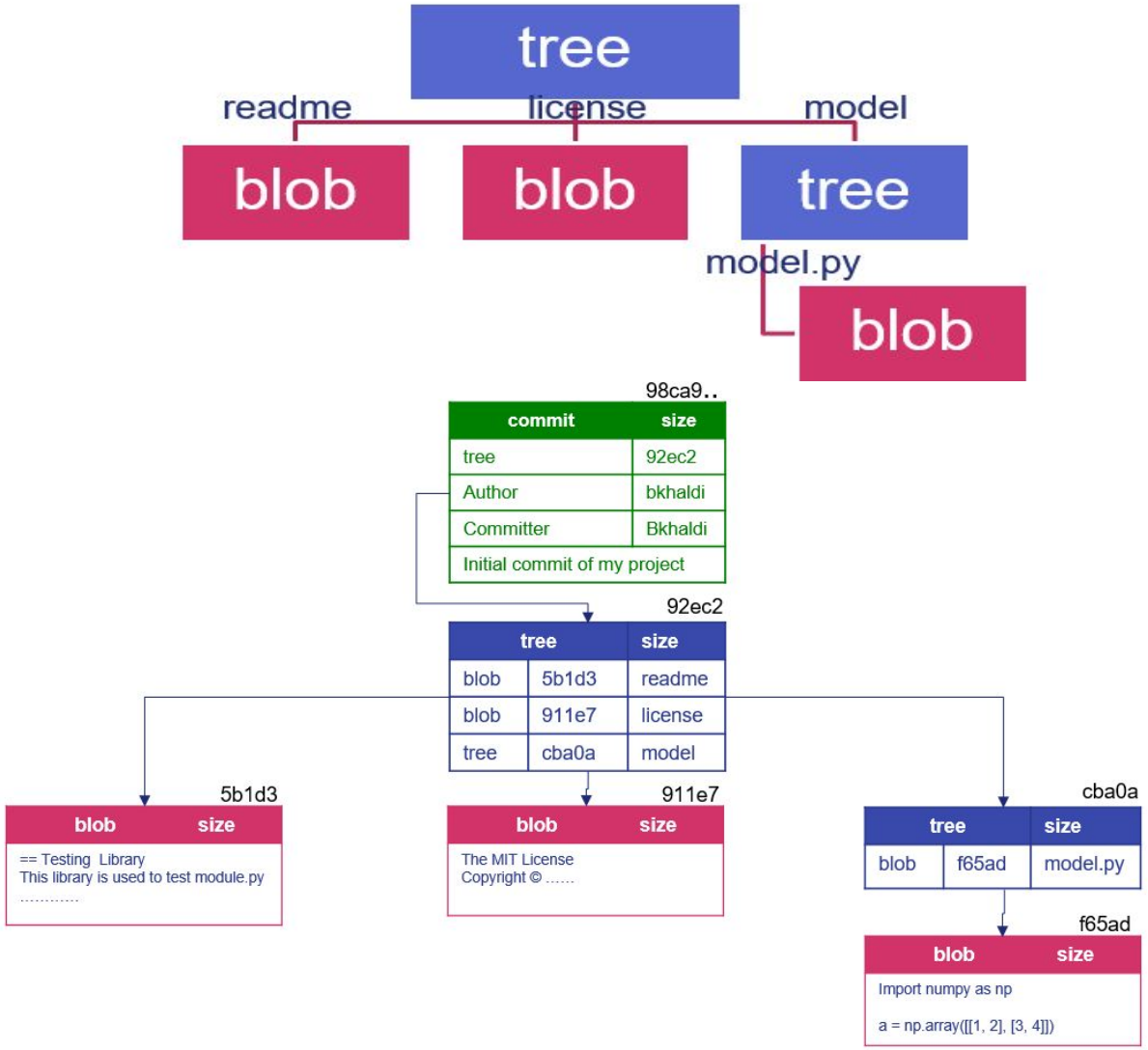
- **Why Git?**
  - Great for coordinating changes on a project among multiple contributors
  - Great for debugging purposes
  - Extremely fast version control
  - Cloud storage of your code.

Git

- **Git Basics**
  - Git thinks of its data as a set of **snapshots (commit)** of a miniature filesystem.
  - Every time a project state is saved (commited), Git stores a reference to that **snapshot**.
    - File have not changed ⇒ Git doesn't store the file again, just a link to the previously stored identical file.

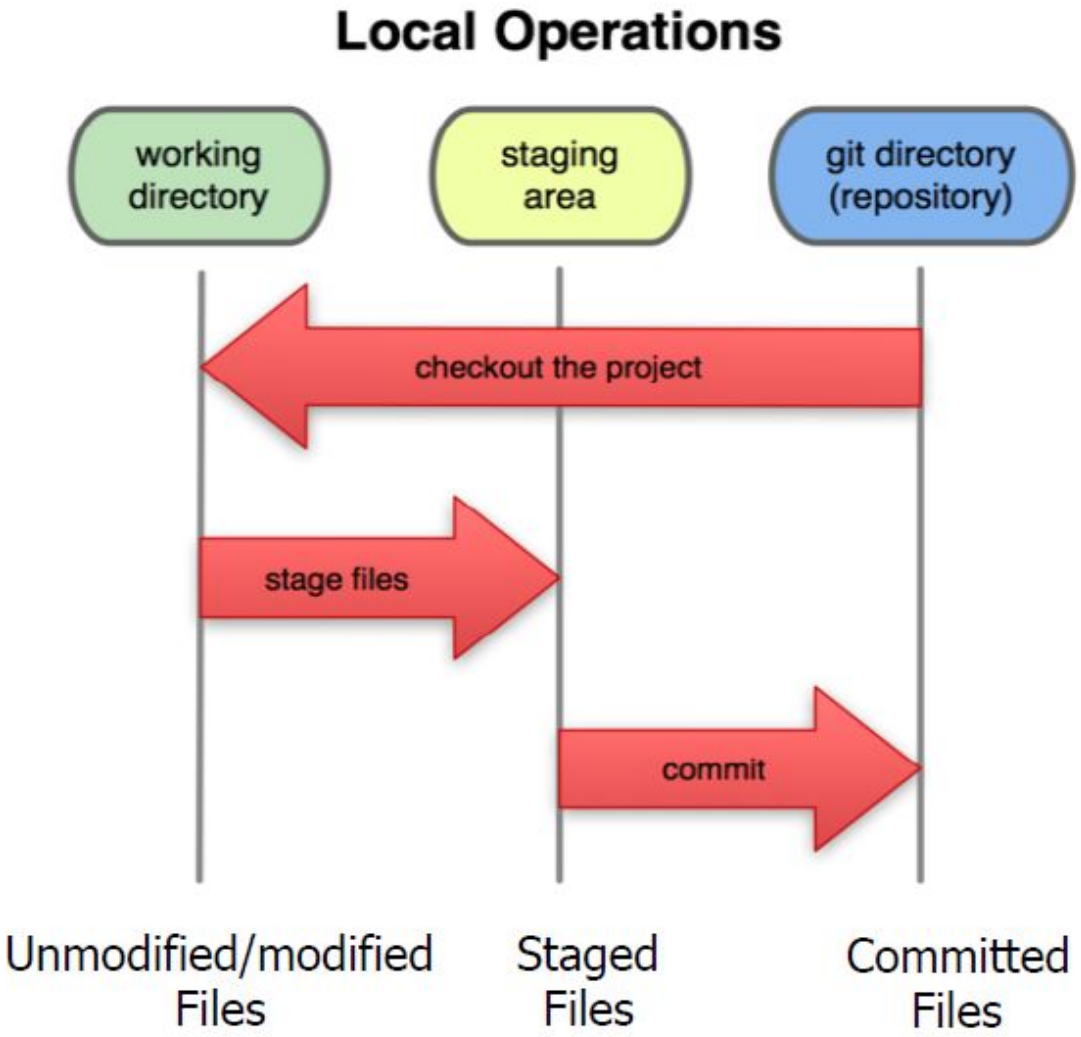
Git

- **Git Basics – Snapshot**
  - Internally, Git stores, mainly, 3 types of objects:
    - **Blob Objects (files)**: A snapshot of a file at a given moment.
    - **Tree Objects (folders)**: A tree has one or more entries, each of which is the SHA-1 hash of a blob (a reference to a file) or subtree.
    - **Commit Objects:** The main contents of a commit object are a reference to a tree, a reference of the parent commit and some metadata.



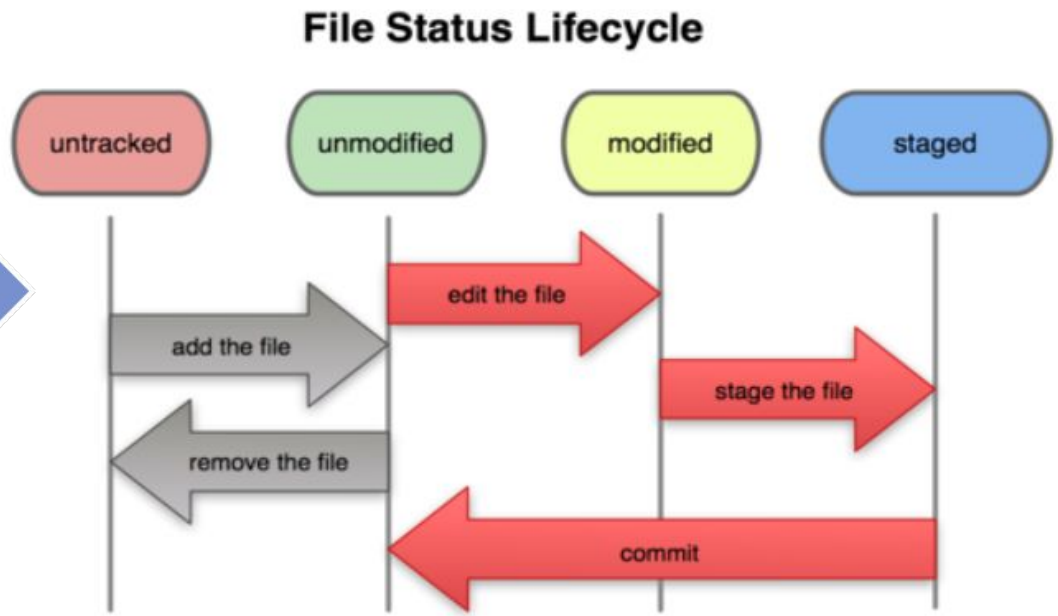
Git

- **Git Basics – Local Operations**
  - Three states that your files can be in: **committed, modified, and staged.**
    - **Modified:** file changed but not committed to database yet
    - **Staged:** current version of modified file marked to go into next commit snapshot
    - **Committed:** data stored in local database



Git

- **Git Basics – Basic Git Workflow**
  - **Workflow**



Git

- **Basic Local Git Commands**
  - **Git Configuration**

```
git config --global user.name "<user_name>"
git config --global user.email "<email@domain>"
```

  - **Creating a repository**

```
$ mkdir <folder_name>
$ cd <folder_name>
```

```
$ git init
```

> Initialized empty Git repository in <folder_name>/.git/

```
$ git status
```

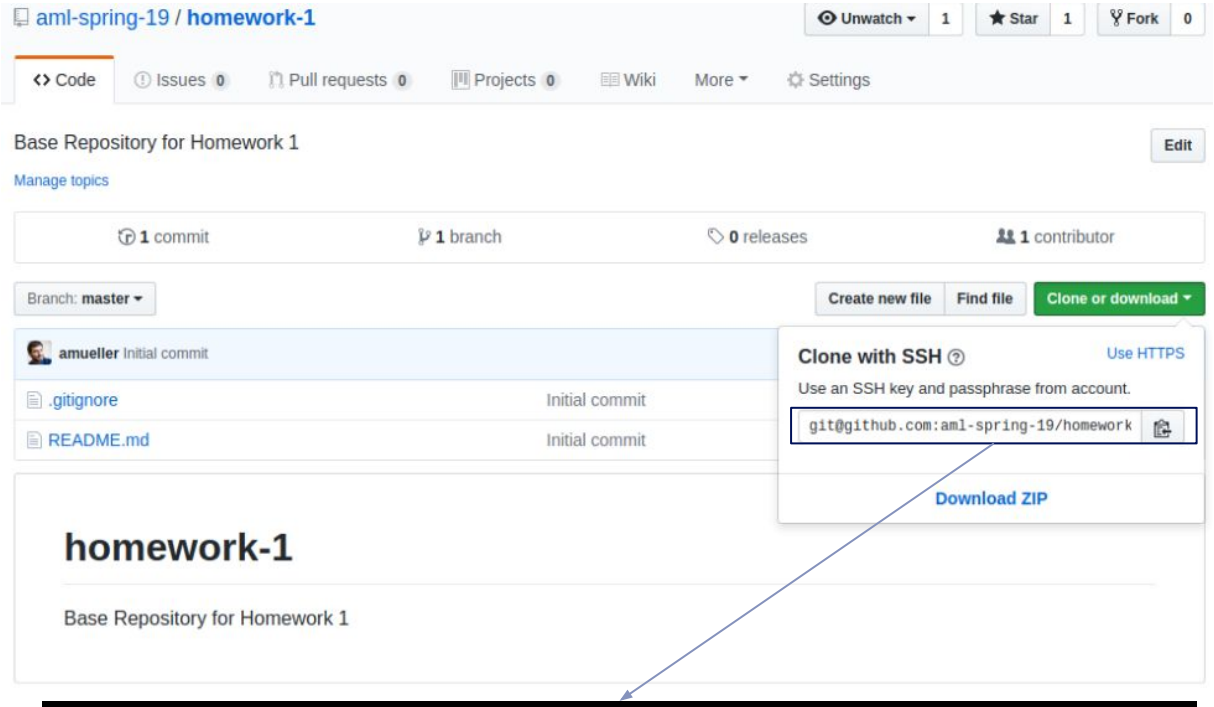
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

  - **Cloning a repository**

```
$ git clone <remote_reposotery_fom_github>
```




```
$ git clone git@github.com:aml-sprint-19/homework-1
```

Git

- **Basic Local Git Commands**
  - **Creating and staging a file**

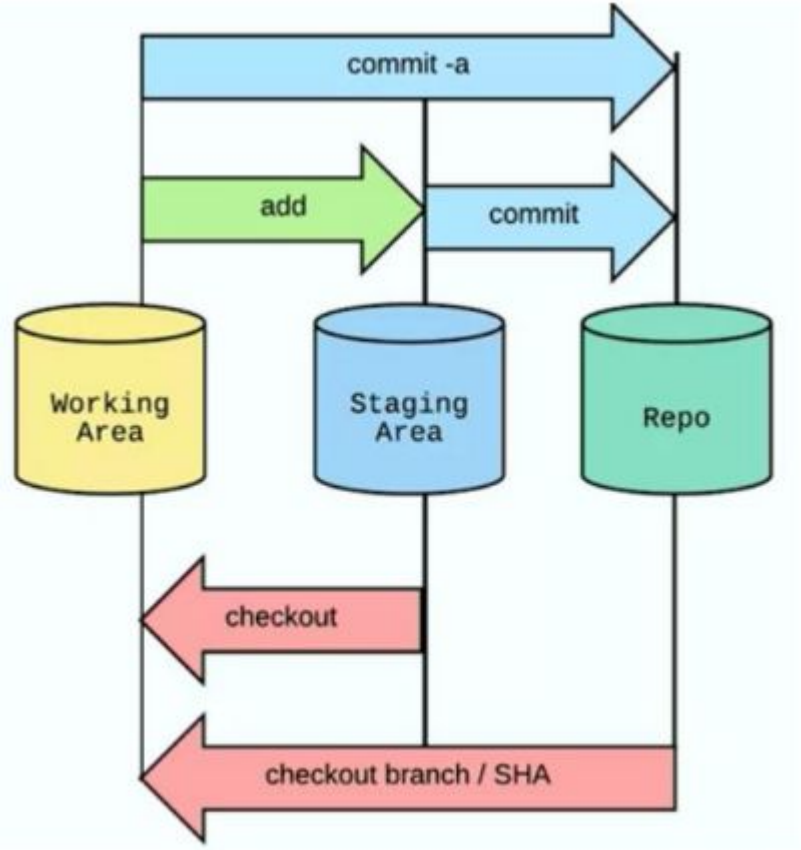
```
$ echo "print('Hello world!')" >> task1.py
```

```
$ git status
```

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        task1.py

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

```
$ git add task1.py
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   task1.py
```



Git

- **Basic Local Git Commands**
  - **Viewing History**

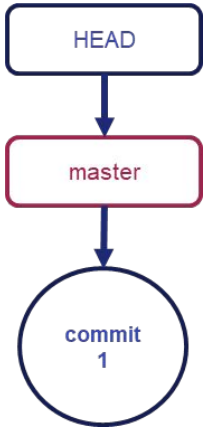
```
$ git commit -m "1st version: say hello"
```

```
[master (root-commit) 0380c57] 1st version: say hello
 1 file changed, 1 insertion(+)
 create mode 100644 task1.py
```

```
$ git log
```

```
commit 0380c57713d2cfff59bfadda316a06ddfc540388 (HEAD -> master)
Author: belkacem khaldi <b.khaldi@esi-sba.dz>
Date:   Tue Nov 1 20:47:07 2022 +0100

    1st version: say hello
```

- A snapshot is identified by **SHA-1 hash**.
- Git assigns **references** for hashes
- For example:
  - "**master**" (or main) ⇒ points to the latest **commit** in the main branch of development.
  - "**Head**" ⇒ points to "where we currently are"



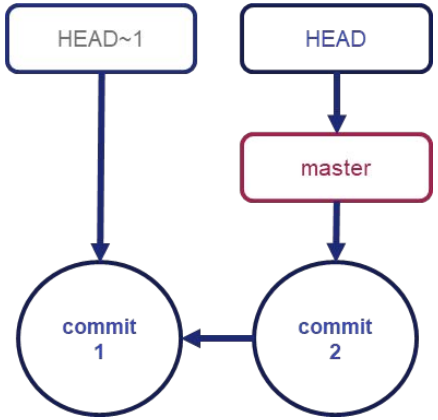

```
$ echo "print('module 1 added')" >> module1.py
```

```
$ git add .
```

```
$ git commit -m "module.py added"
```

```
$ git log
```

```
commit 05a89ac93838137fbae1e4f8af02c90ed257ee1e (HEAD -> master)
Author: belkacem khaldi <b.khaldi@esi-sba.dz>
Date:   Tue Nov 1 21:19:16 2022 +0100

    module.py added

commit 0380c57713d2cfff59bfadda316a06ddfc540388
Author: belkacem khaldi <b.khaldi@esi-sba.dz>
Date:   Tue Nov 1 20:47:07 2022 +0100

    1st version: say hello
```



Git

- **Basic Local Git Commands**
  - **Exploring History**

```
$ git diff HEAD~1
```

  - **Or**

```
$ git diff 0380c57713d2cfff59bfadda316a06ddfc540388
```

```
diff --git a/module.py b/module.py
new file mode 100644
index 0000000..7ce1959
--- /dev/null
+++ b/module.py
@@ -0,0 +1 @@
+"print('Second module')"
```

  - **Recovering Older Version**

```
$ git checkout HEAD~1
```

```
HEAD is now at 0380c57 1st version: say hello
```

```
$ git log
```

```
commit 0380c57713d2cfff59bfadda316a06ddfc540388 (HEAD)
Author: belkacem khaldi <b.khaldi@esi-sba.dz>
Date:   Tue Nov 1 20:47:07 2022 +0100

    1st version: say hello
```

Git

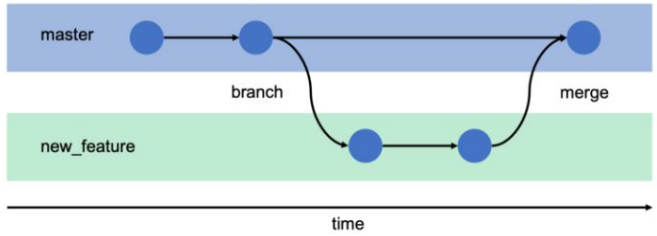
- **Basic Local Git Commands**
  - **Branches**

```
$ git checkout -b "new_feature"
```

```
Switched to a new branch 'new_feature'
```

Make some changes, add, commit…

- Branches are used to copy an existing repository and create a new development "branch"



  - **Moving between branches:**

```
$ git checkout master
```

```
Switched to branch 'master'
```

  - **Merge Branches**

```
$ git checkout  "new_feature"
```

```
$ git merge master
```

```
Updating 0380c57..23270ee
Fast-forward
 module.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 module.py
```

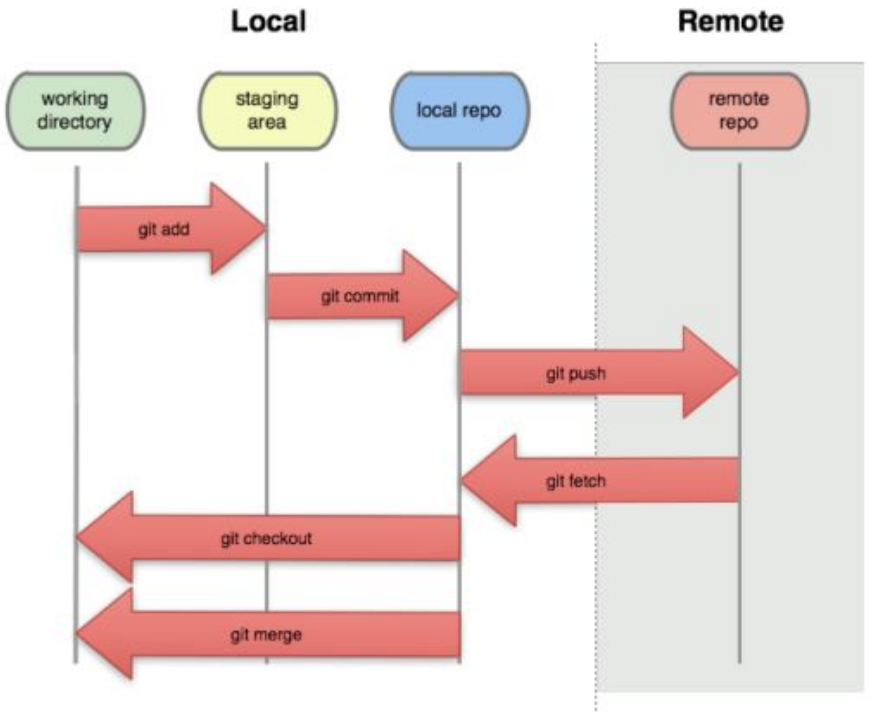
```
$ git log --all --graph --decorate
```

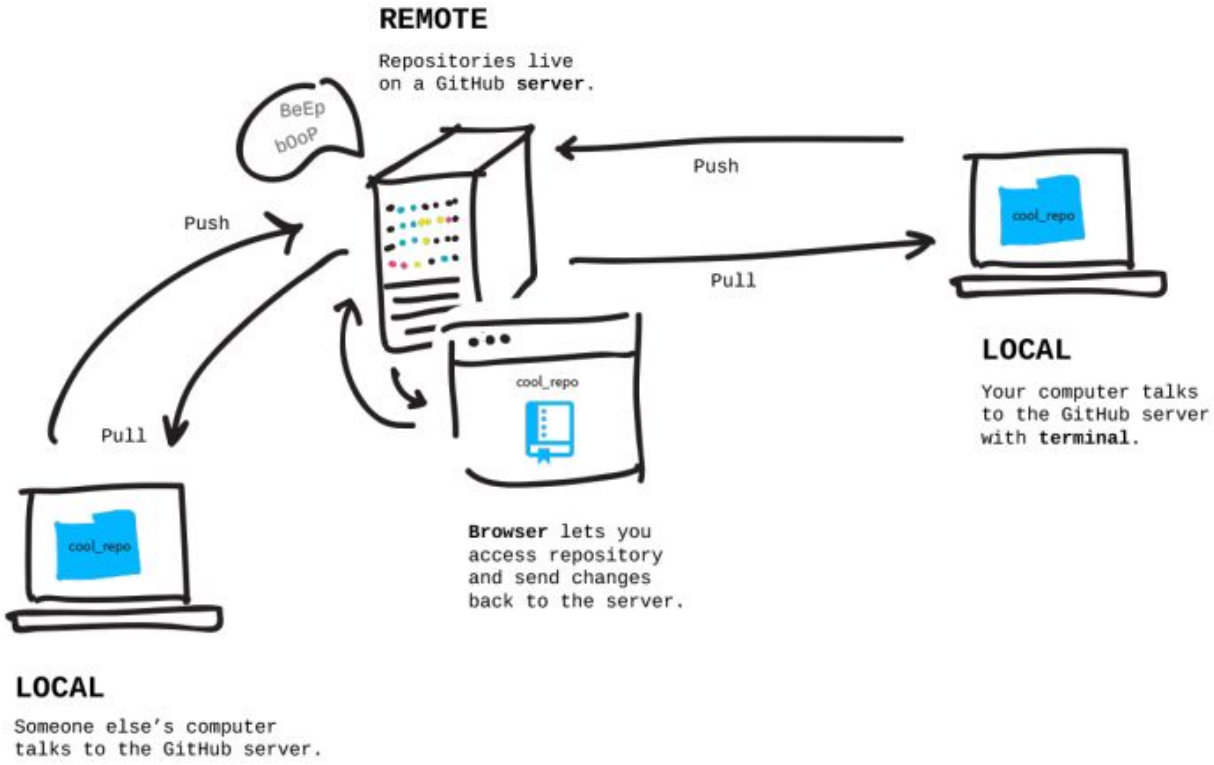
```
* commit 23270ee3d6c34d65bc32e67eec814164cc41b065 (HEAD -> new_feature, master)
| Author: belkacem khaldi <b.khaldi@esi-sba.dz>
| Date:   Tue Nov 1 22:18:18 2022 +0100
|
|     updated module.py
|
* commit 05a89ac93838137fbae1e4f8af02c90ed257ee1e
| Author: belkacem khaldi <b.khaldi@esi-sba.dz>
| Date:   Tue Nov 1 21:19:16 2022 +0100
|
|     module.py added
|
* commit 0380c57713d2cfff59bfadda316a06ddfc540388
  Author: belkacem khaldi <b.khaldi@esi-sba.dz>
  Date:   Tue Nov 1 20:47:07 2022 +0100

      1st version: say hello
```

GitHub

GitHub

- **What is GitHub?**
  - GitHub ⇒ A web-based hosting service for software development projects that use the **Git** revision control system.





GitHub

- **SSH Keys**
  - **SSH** keys are special, unique files that allow the user to access secure data (in this case our code)
  - **Git** implements a similar level of security and requires user to **ssh** with **ssh keys**.
  - If a user does not have a key, they will not gain access to anything

- Generate your **SSH keys**
  - Enter the following command with the email associated with your **GitHub** account:

```
ssh-keygen
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\user/.ssh/id_rsa):
Created directory 'C:\Users\user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\user/.ssh/id_rsa.
Your public key has been saved in C:\Users\user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:sFdi2/kARUK7EsYhXLqVIJWGwcZKj5H8rQNifHZZoa8 user@DESKTOP-C6MVBVN
The key's randomart image is:
+---[RSA 3072]----+
|.o+=+ooo+.o      |
| ==.+=.+ +       |
|oo=.o.X = .      |
|o= = B.= B .     |
|o + + o.S +      |
|   o  .o   o     |
|    .E      .    |
|                 |
|                 |
+----[SHA256]-----+
```

To Copy the SSH Key

Windows (OS) →

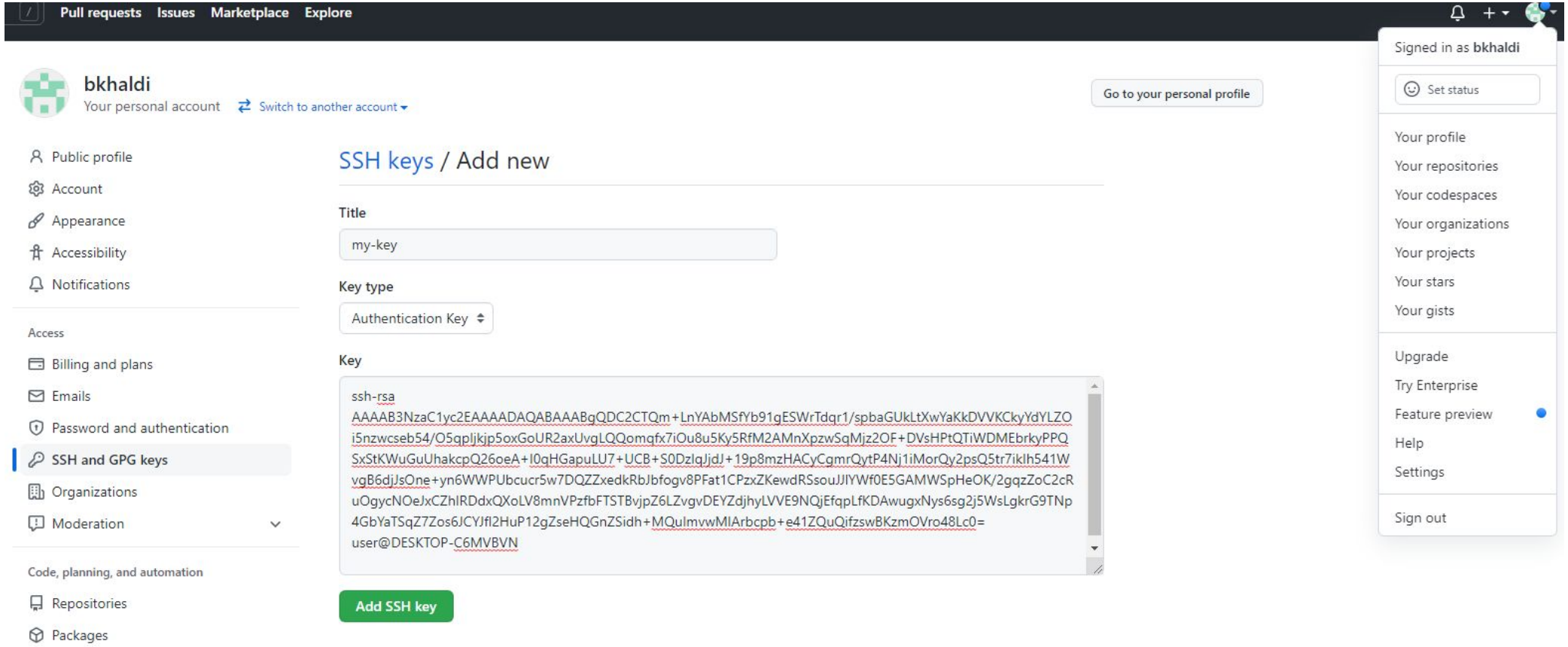
```
clip < C:\Users\user/.ssh/id_rsa.pub
```

Linux/Mac (OS) →

```
pbcopy < C:\Users\user/.ssh/id_rsa.pub
```

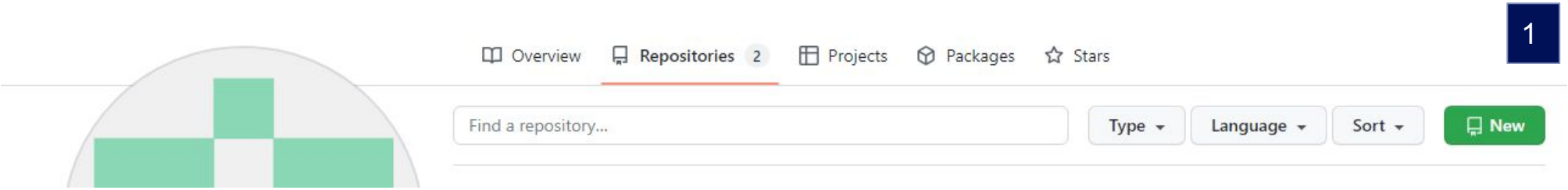
GitHub

- **SSH Keys – GitHub Settings**



GitHub

- **GitHub Repository**



- Find the above search bar, and look for the button labeled new.
- In the top right corner, press on your icon and then in the menu bar, select Your repositories

GitHub

- **GitHub Repository**

3

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

Set up in Desktop or HTTPS SSH https://github.com/bkhaldi/git_folder.git

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 "# git_folder" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/bkhaldi/git_folder.git
git push -u origin main
```

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

```
git remote add origin https://github.com/bkhaldi/git_folder.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.

Import code

2

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * bkhaldi / Repository name * git_folder

Great repository names are short and memorable. Need inspiration? How about urban-potato?

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.
.gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None

You are creating a private repository in your personal account.

Create repository

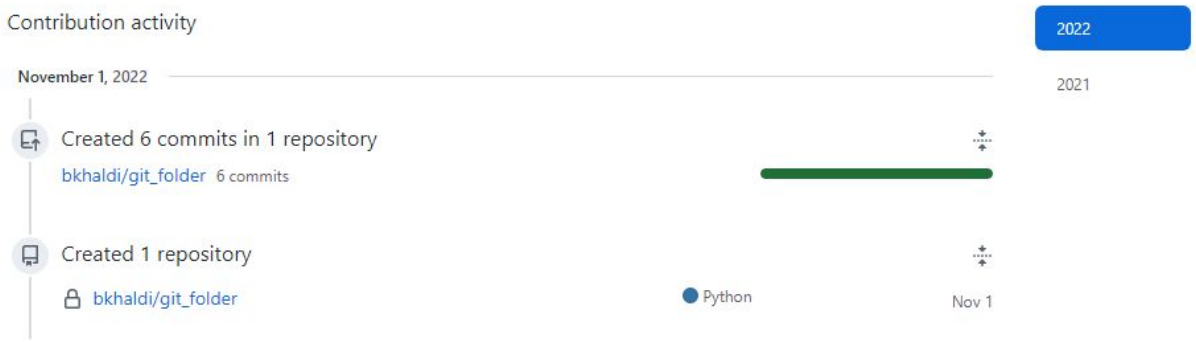
GitHub

- **GitHub Repository**

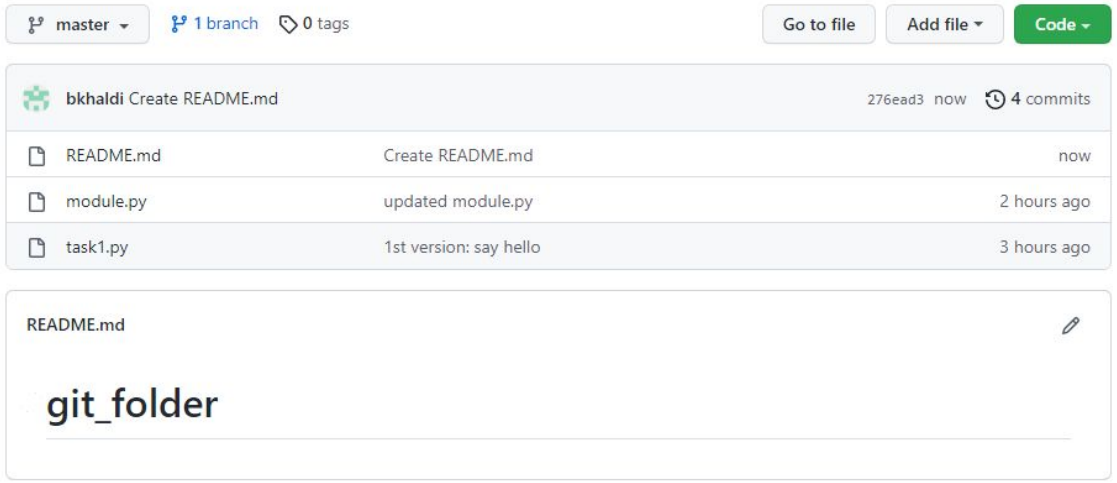
4

```
git push -u origin master
```

```
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), 810 bytes | 270.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bkhaldi/git_folder.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```



Let's make a change in the remote repo.



5

```
git pull
```

```
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 693 bytes | 7.00 KiB/s, done.
From https://github.com/bkhaldi/git_folder
   23270ee..276ead3  master     -> origin/master
Updating 23270ee..276ead3
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Thanks for your Listening

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz