AI/ML for prediction of biological properties of molecules

Module 2. Setting up the computational environment

Gemma Turon & Miquel Duran-Frigola
Ersilia Open Source Initiative (www.ersilia.io)
18th - 27th of September, 2023

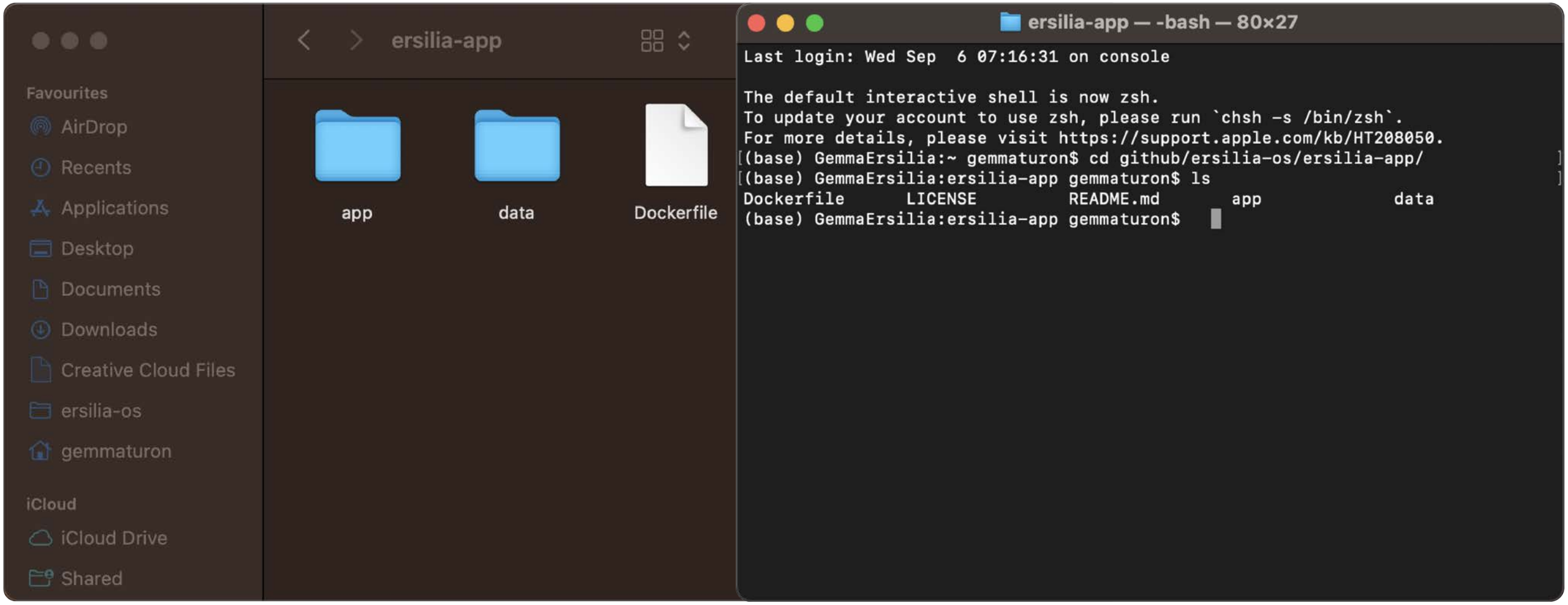
Ersilia

Let’s get started!

Go to menti.org and introduce this code: 123456789

The Command Line Interface (CLI)

Command Line Interface (CLI) vs Graphical User Interface (GUI)



The CLI sends commands in the form of lines of text to the computer, as opposed to the GUI, which has clickable menus

Find the CLI in your system

The programming language of the CLI or Terminal varies between systems

— Windows: ~~command prompt (CMD)~~
— UNIX (MacOS and Linux): bash and zsh

👉 Windows users: download and install PowerShell 7.0
👉 UNIX users: open the Terminal
💡 Use the search function if you can't locate the Terminal

Uses of the CLI

- Automate repetitive tasks
- Pipe several commands, eliminating the need for generating intermediate files and steps
- Use programs that do not offer a GUI:
  - Send API queries to download data from an online repository
  - Access and operate remote computers
- ...

Python and Package managers

Multiple programming languages

Programming languages are sets of rules that convert human-readable text to machine code
Their instructions are precise and only have one single meaning

— Human readable syntax
— Flexibility for several applications
— Lots of pre-built packages

Python packages

```
# Install package via manager (pip or
conda)

!pip install rdkit

# Import modules or functions from the
package

import rdkit
from rdkit import Chem


# Call a specific function to use it
molecule =
Chem.MolFromSmiles("CCCCOCCCC")
```

Python packages are collections of reusable Python modules that provide pre-written code and functionality. They simplify code reuse and maintenance.

Types of Python files: scripts

```python
# Import modules or functions from the package

import rdkit
from rdkit import Chem


# Call a specific function to use it
molecule = Chem.MolFromSmiles("CCCCOCCCC")
inchikey = Chem.inchi.MolToInchiKey(molecule)

# Print the output of the functions
print(inchikey)
```

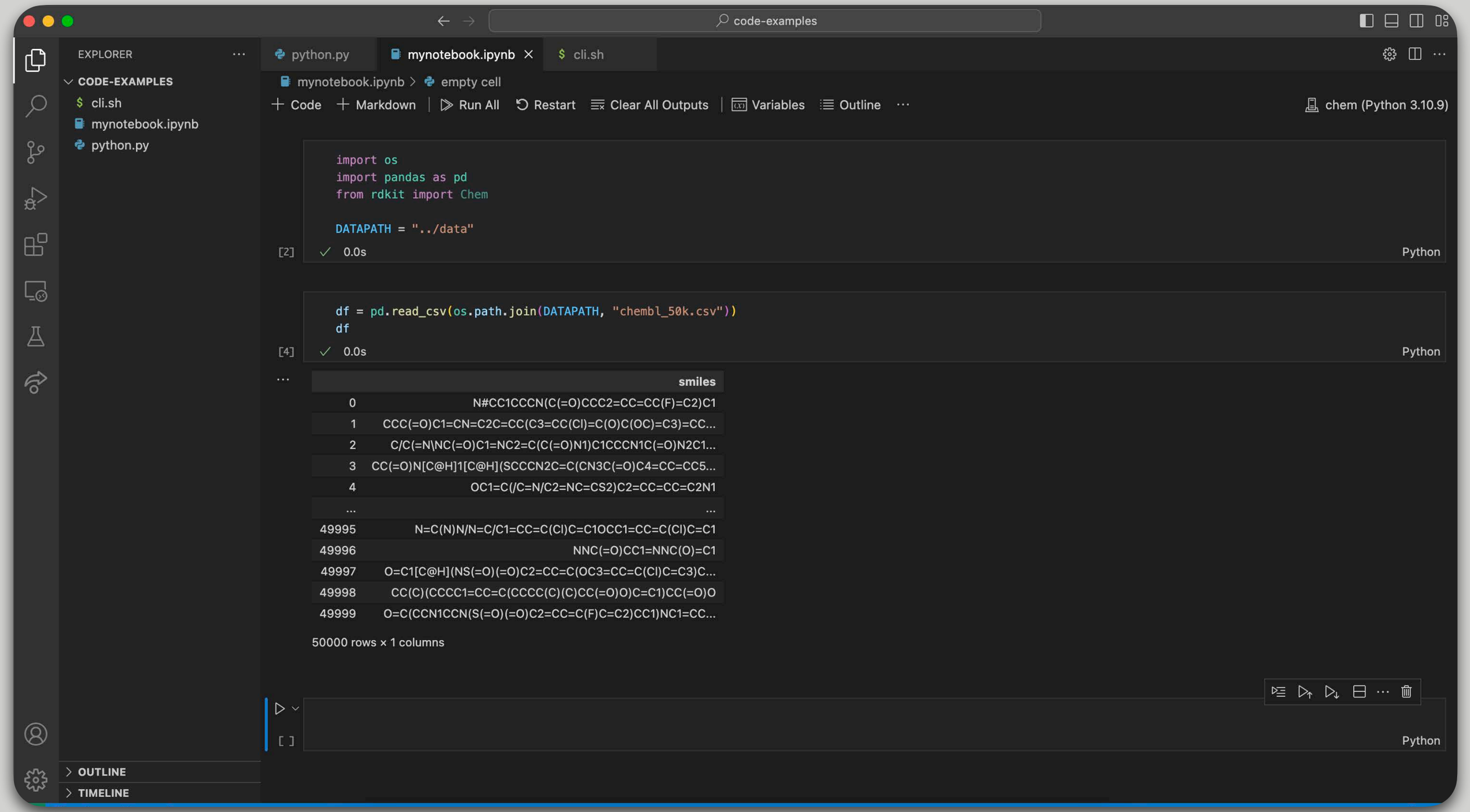
- A script is a .py file
- It has a series of commands that are run one after the other
- At the top, we need to import the required packages

```
# Run a python script

(mycondaenv) $ python myscript.py
DURPTKYDGMDSBL-UHFFFAOYSA-N
```

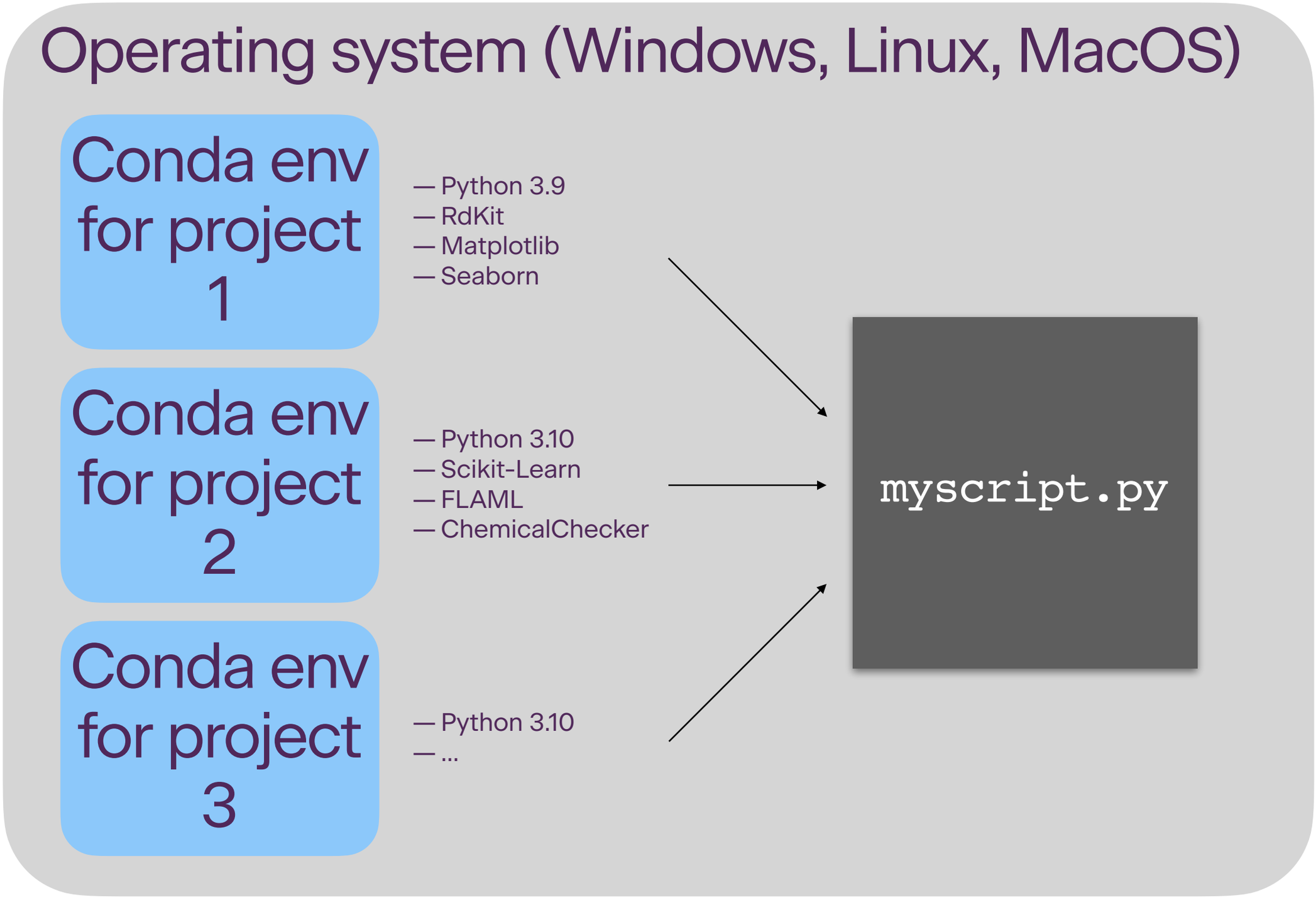
- We run it via the CLI

Types of python files: Jupyter notebooks

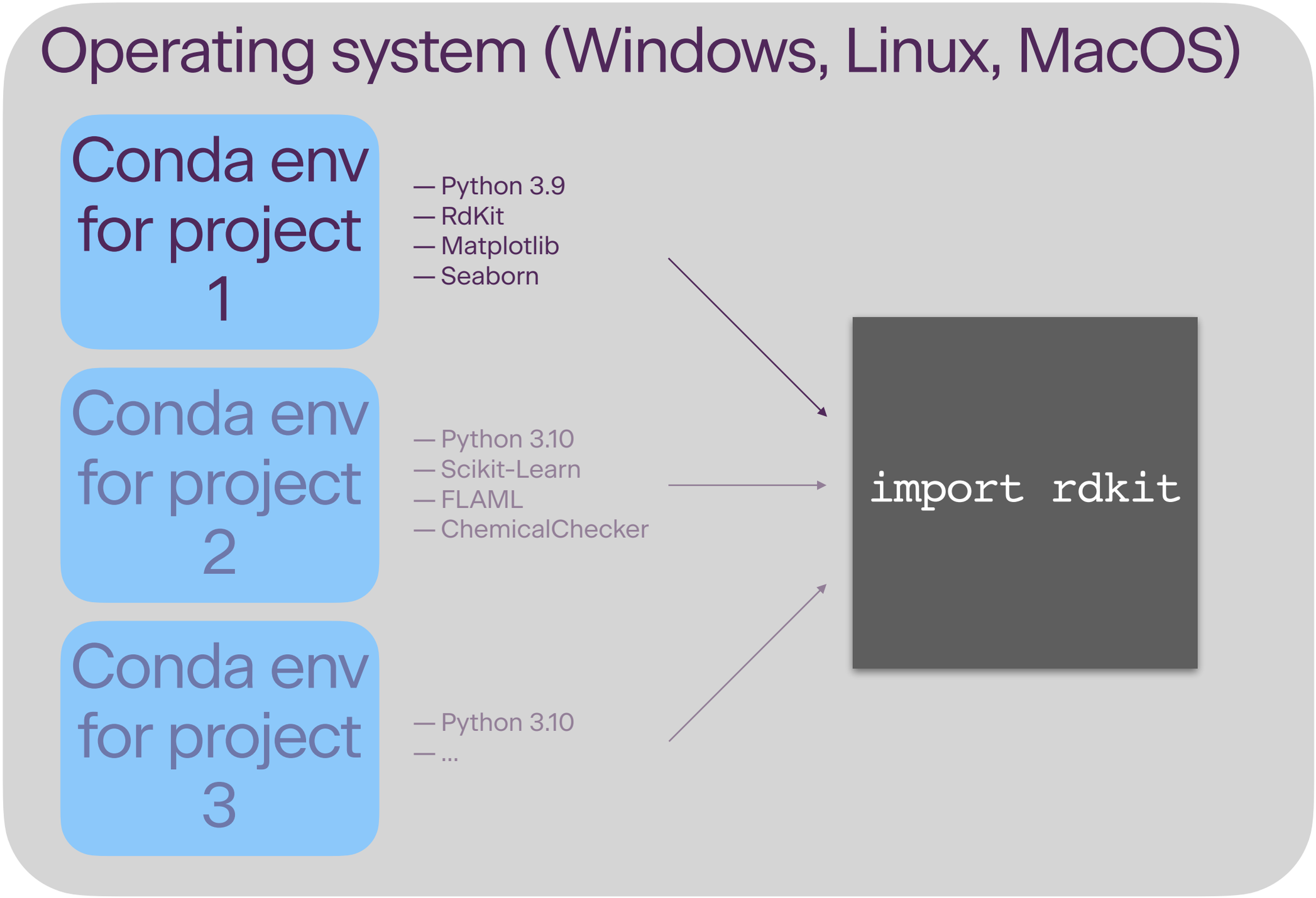
A notebook allows us to run small pieces of code at a time, making it very easy to perform data analysis.



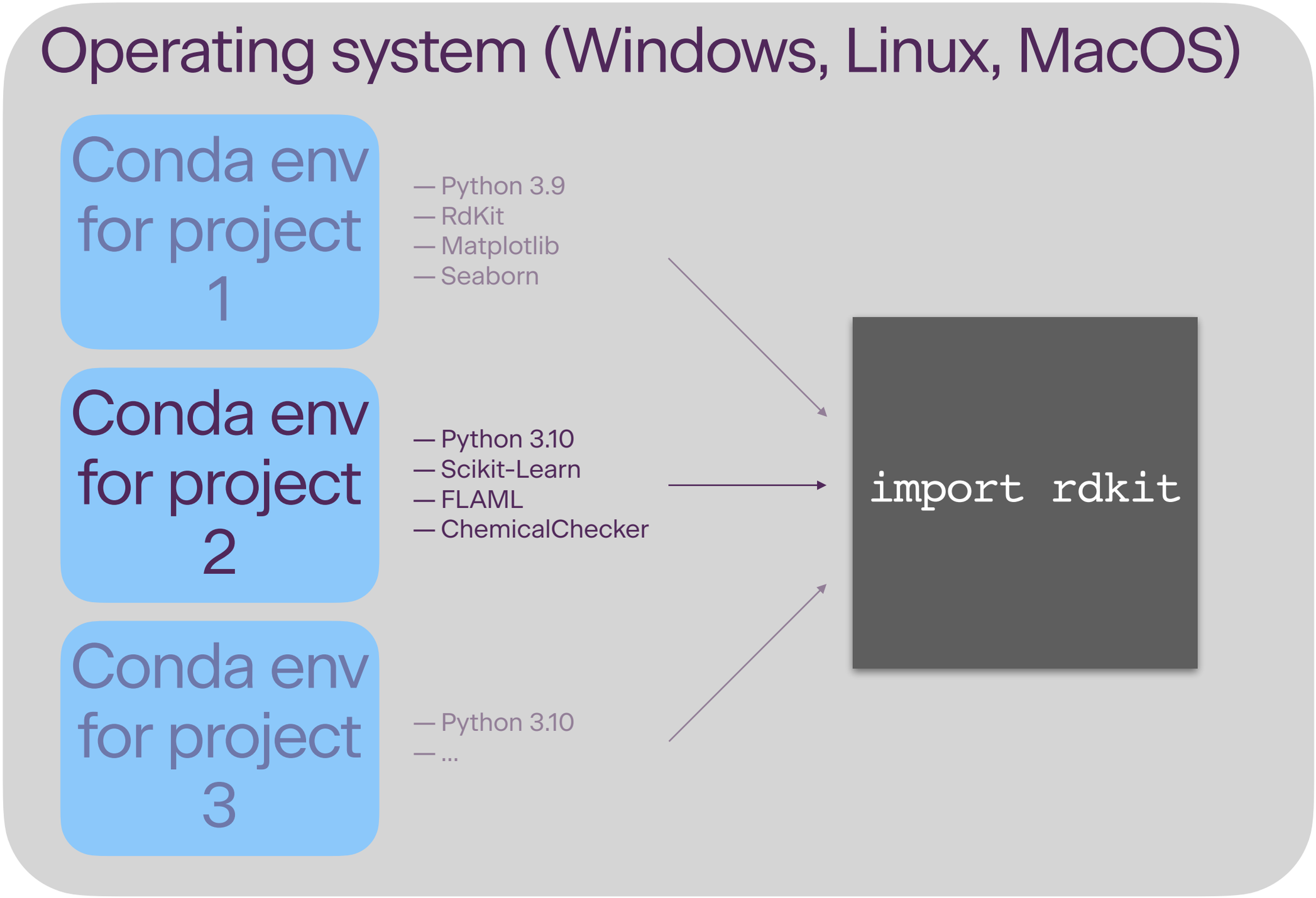
Where does Python run?



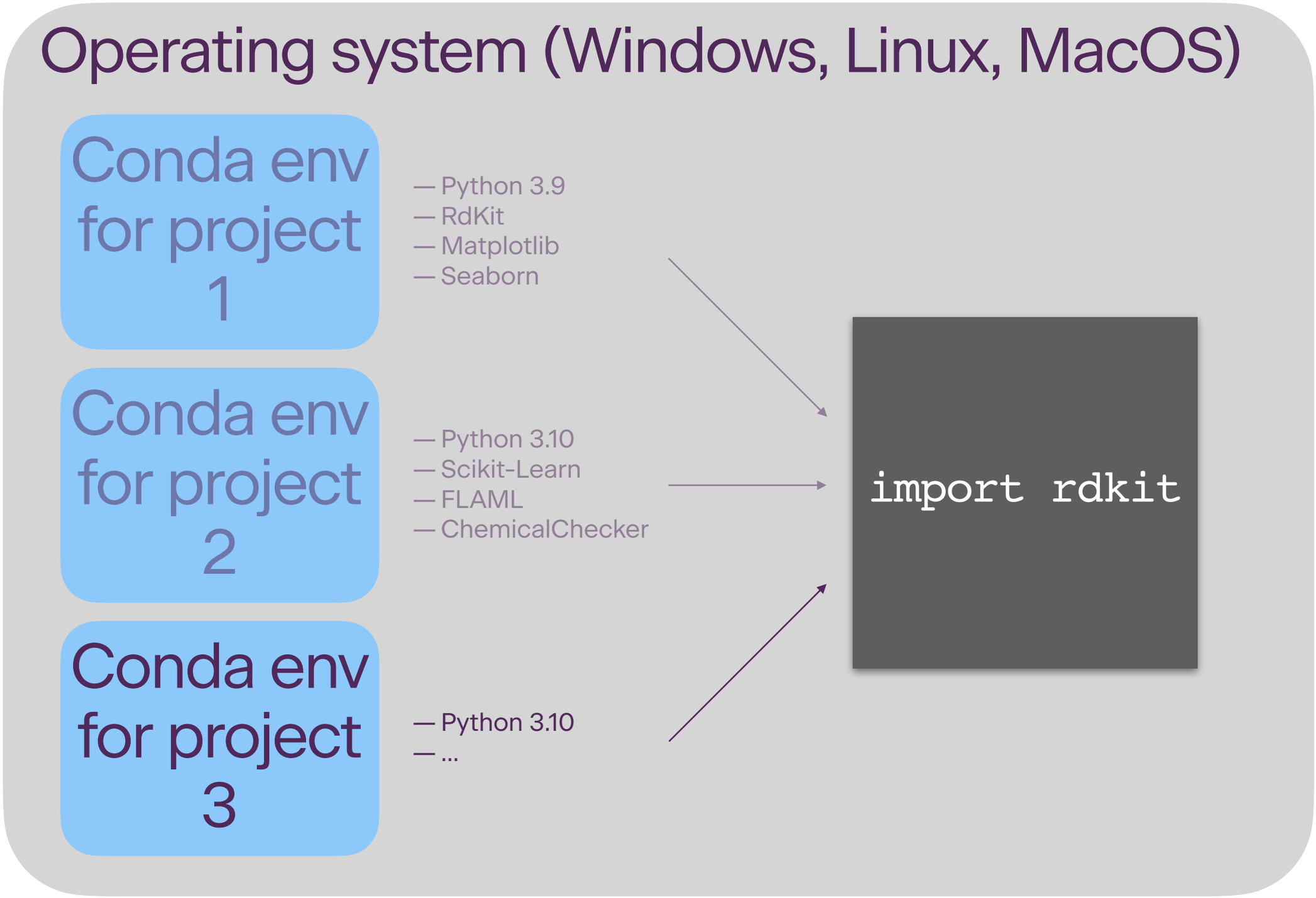
Where does Python run?



Where does Python run?

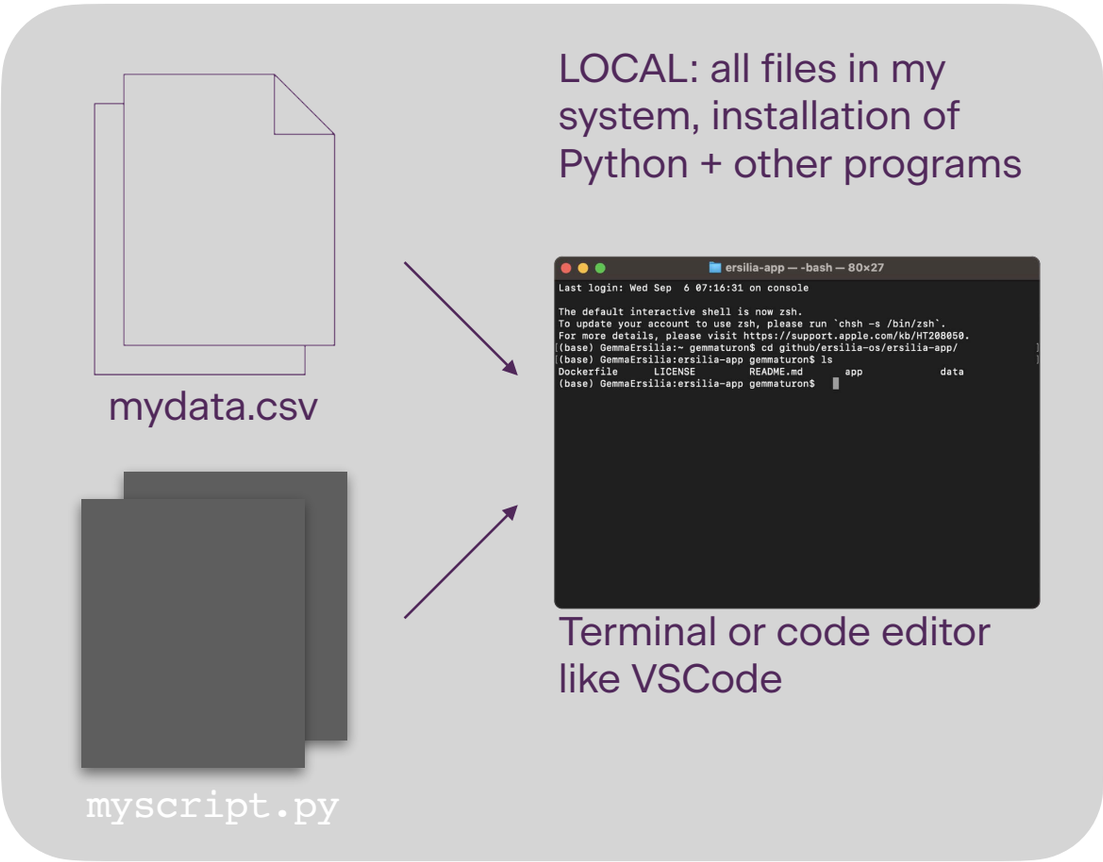


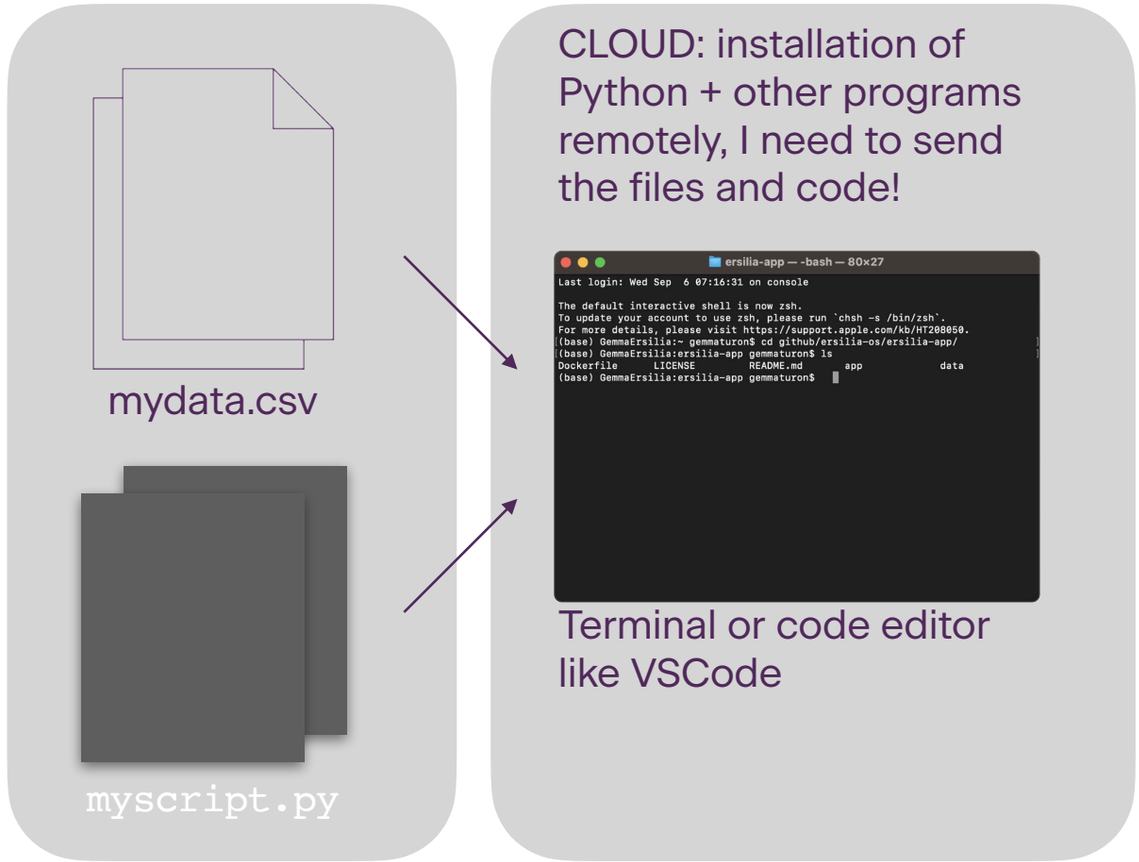
How do we run Python?



Where does Python run?

- Locally: we need python installed as well as a package manager like Anaconda
- Remote computer: all the packages are installed in the remote computer





Git and GitHub

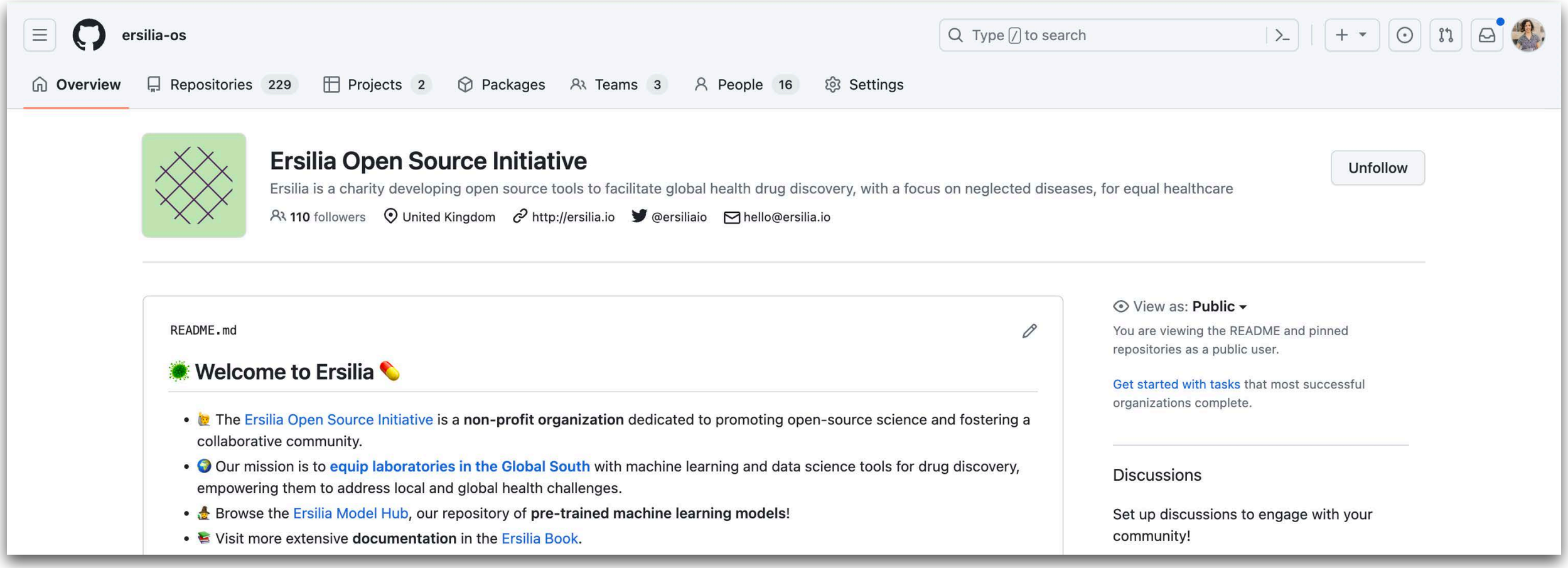
What is Git? What is GitHub?

Git is a software that tracks changes to files, including creating new files, deleting files and folders, and editing content

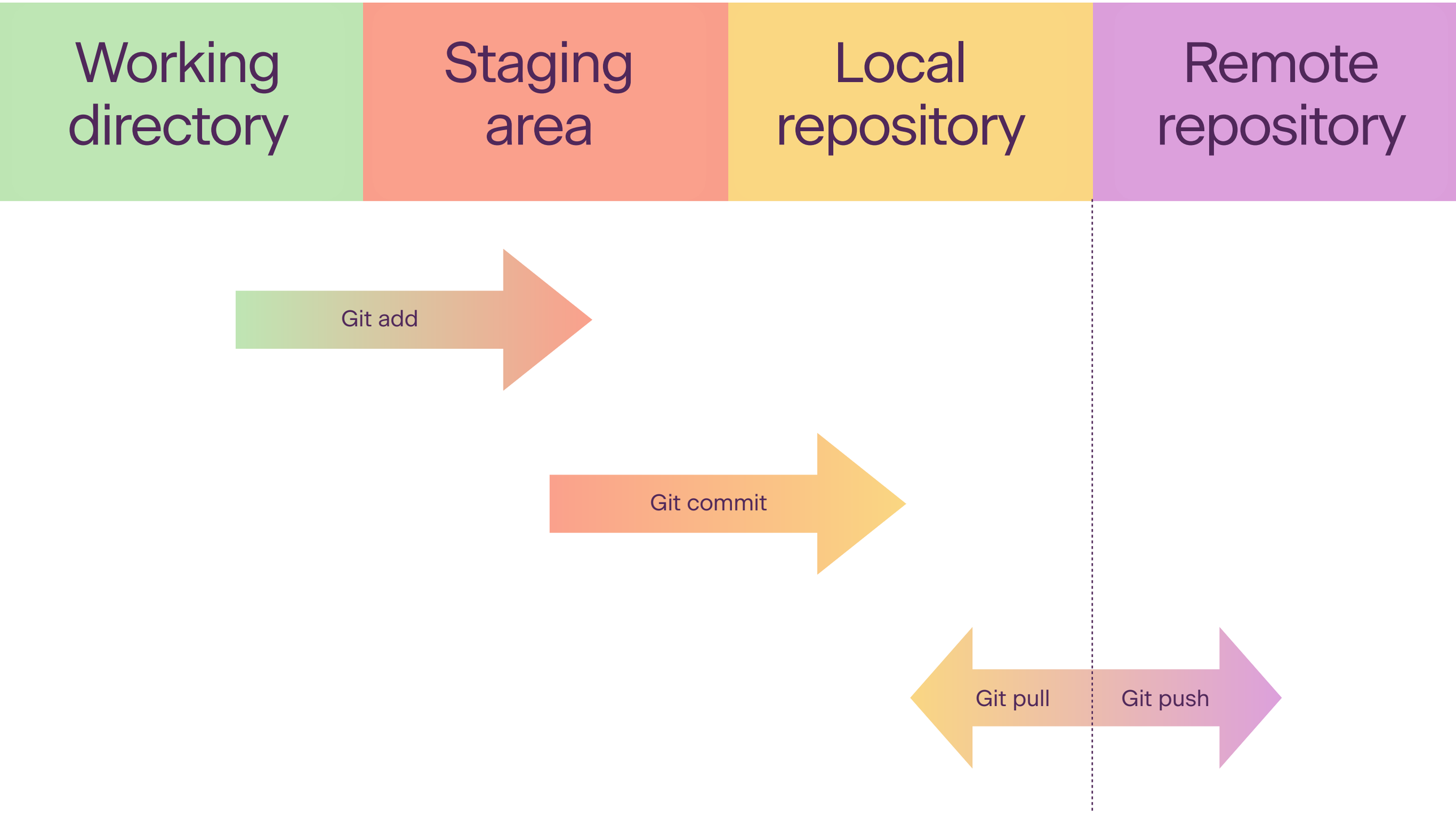
GitHub is an internet hosting service for software development and version control using Git

The importance of GitHub 🚀

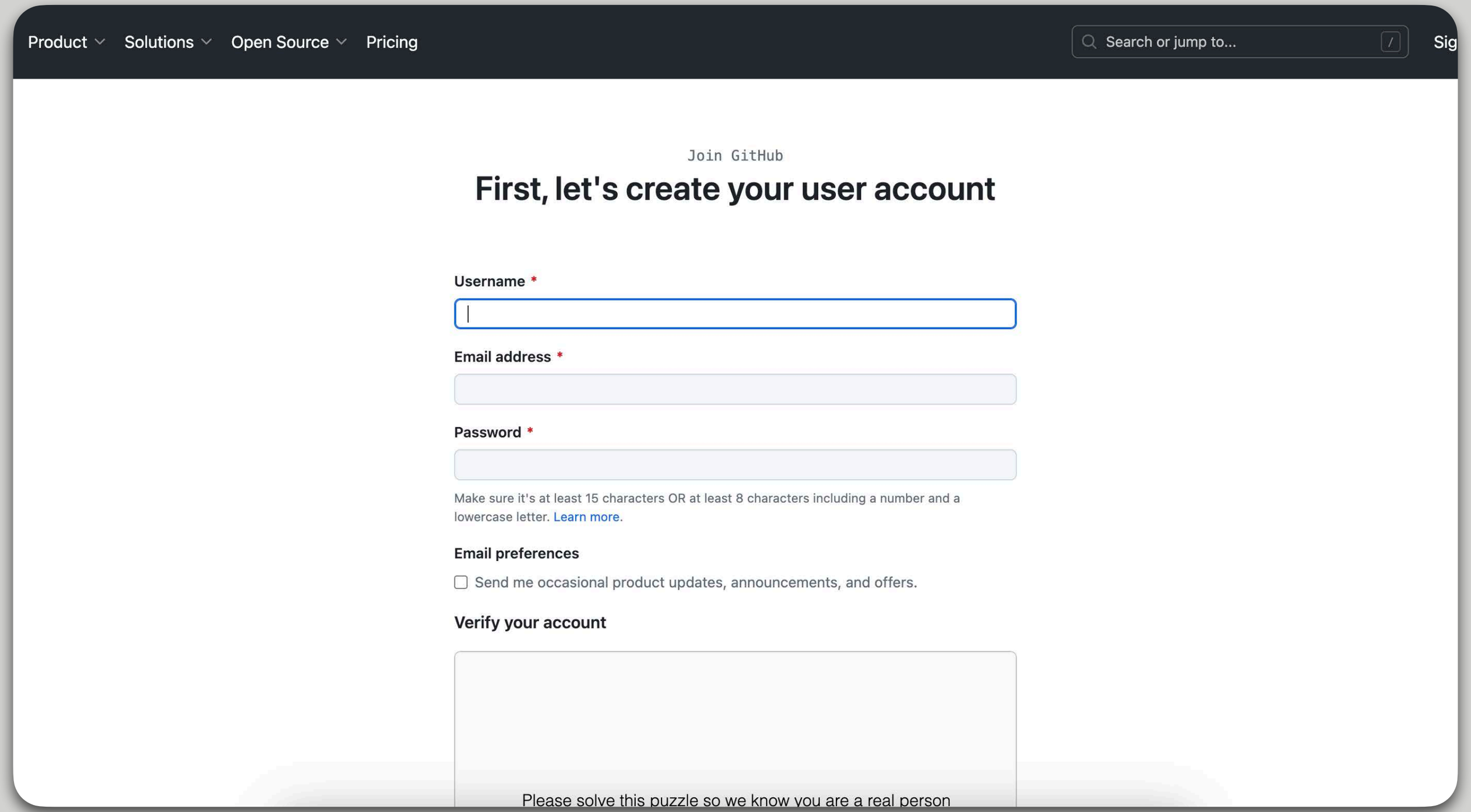
- 100 million developers
- 372 million repositories (28 million are public)
- More than 20,000 academic papers are referenced in GitHub repositories



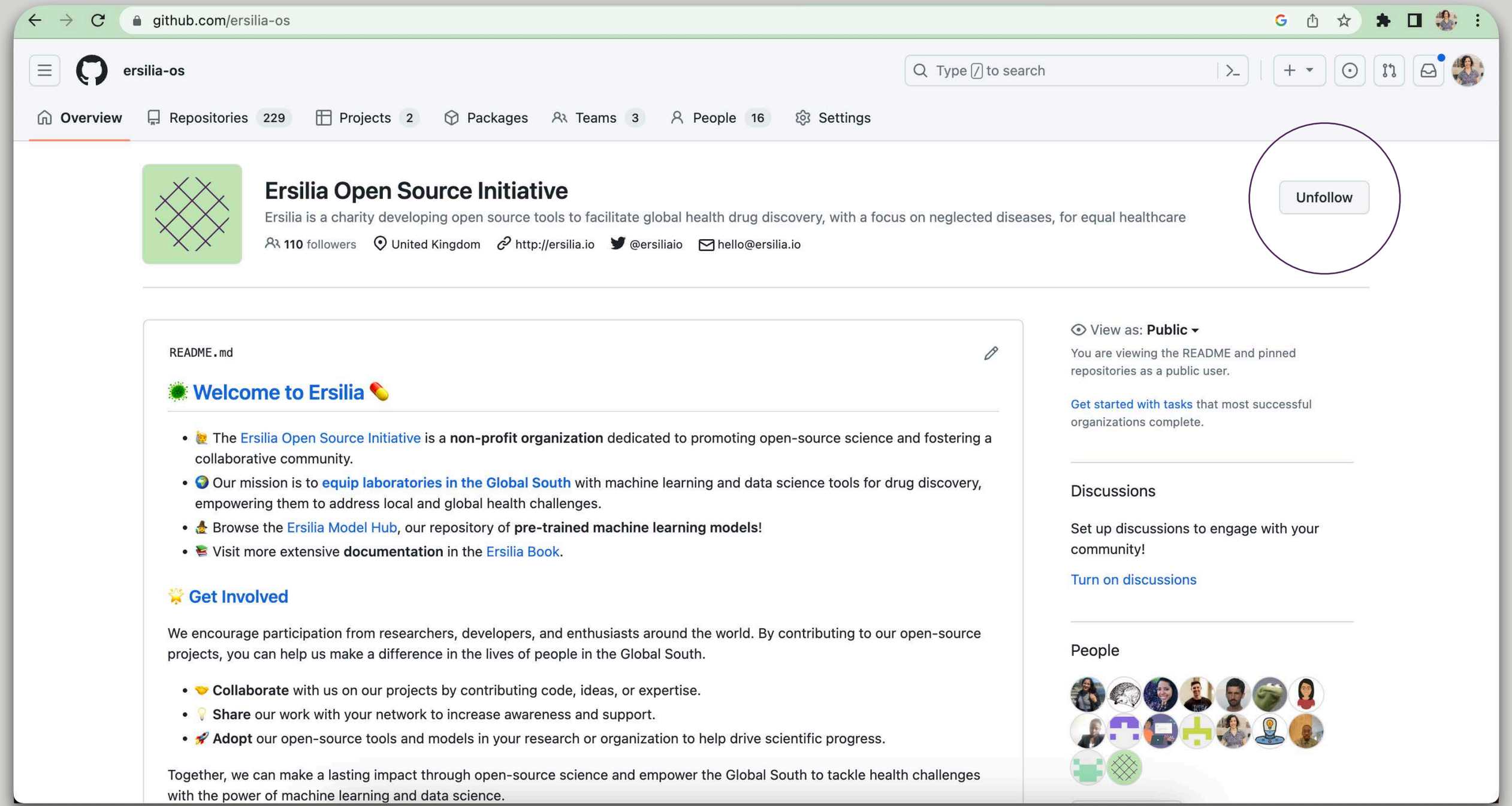
How does Git work?



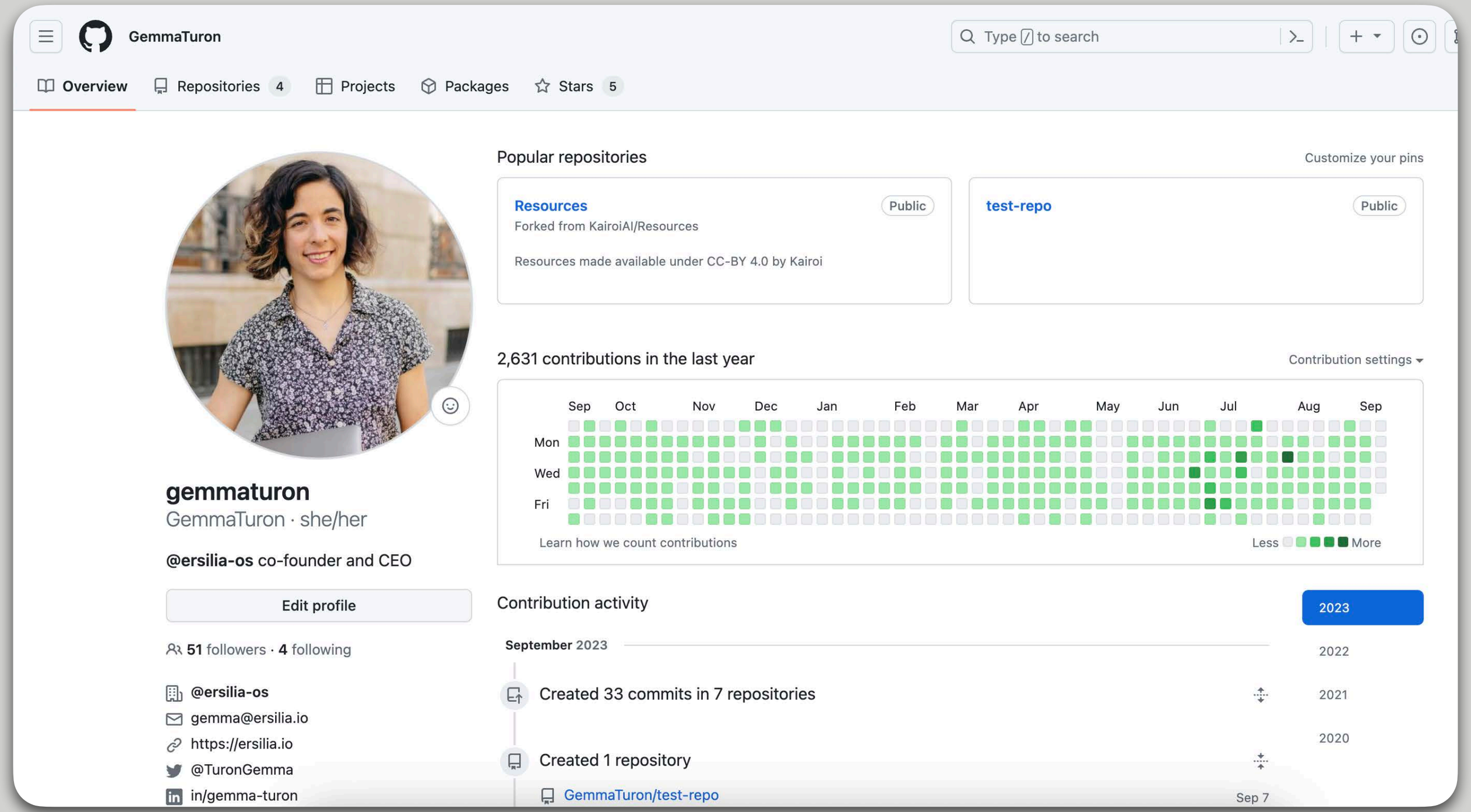
Join GitHub! 🤗
https://github.com/join



It's like social media - Follow Ersilia!
https://github.com/ersilia-os/



It’s like social media - Track your progress!



Let’s recap!

Go to: menti.com
Code: 1271 8086

Extra information

Basic commands with the CLI

```
#change directory (cd) command
$ cd Desktop
~/Desktop $

#bash is case sensitive
$ cd desktop
bash: cd: desktop No such file or
directory

#make directory (mkdir)
~/Desktop $ mdkir test
~/Desktop $ cd test
~/Desktop/test $

#go back one directory
~/Desktop/test $ cd ..
~/Desktop $
```

`$cd` move to a directory

`$mkdir` make a new directory

`$cd ..` move one directory up

Basic commands with the CLI

```
# list files and directories (ls)
~/Desktop $ ls
test
folder1
file1

~/Desktop $ cd test
~/Desktop/test $ ls

#create a file
~/Desktop/test $ touch example.txt
~/Desktop/test $ ls
example.txt

#add a line of text
~/Desktop/test $ echo "my first line" >
example.txt
~/Desktop/test $ vim example.txt
```

`$cd` move to a directory

`$mkdir` make a new directory

`$cd ..` move one directory up

`$ls` list files in directory

`$touch` make a new file

`$echo` add content to file

Basic commands with the CLI

```
# move the files to new directory
~/Desktop/test $ mkdir subfolder
~/Desktop/test $ ls
subfolder
example.txt
~/Desktop/test $ mv example.txt subfolder
~/Desktop/test $ cd subfolder
~/Desktop/test/subfolder $ ls
example.txt

#copy the file
~/Desktop/test/subfolder $ cp example.txt ../
~/Desktop/test/subfolder $ ls
~/Desktop/test/subfolder $

~/Desktop/test/subfolder $ cd ..
~/Desktop/test $ ls
subfolder
example.txt
```

`$cd` move to a directory

`$mkdir` make a new directory

`$cd ..` move one directory up

`$ls` list files in directory

`$touch` make a new file

`$echo` add content to file

`$mv` move a file

`$cp` copy a file

Basic commands with the CLI

```
# eliminate the files we have created
~/Desktop/test $ ls
subfolder
example.txt
~/Desktop/test $ rm example.txt
~/Desktop/test $ ls
subfolder

# use the -r flag to eliminate directories
~/Desktop/test $ rm -r
~/Desktop/test $ ls
~/Desktop/test $

~/Desktop/test $ cd ..
~/Desktop/ $ rm -r test
~/Desktop $ ls
folder1
file1
```

`$cd` move to a directory

`$mkdir` make a new directory

`$cd ..` move one directory up

`$ls` list files in directory

`$touch` make a new file

`$echo` add content to file

`$mv` move a file

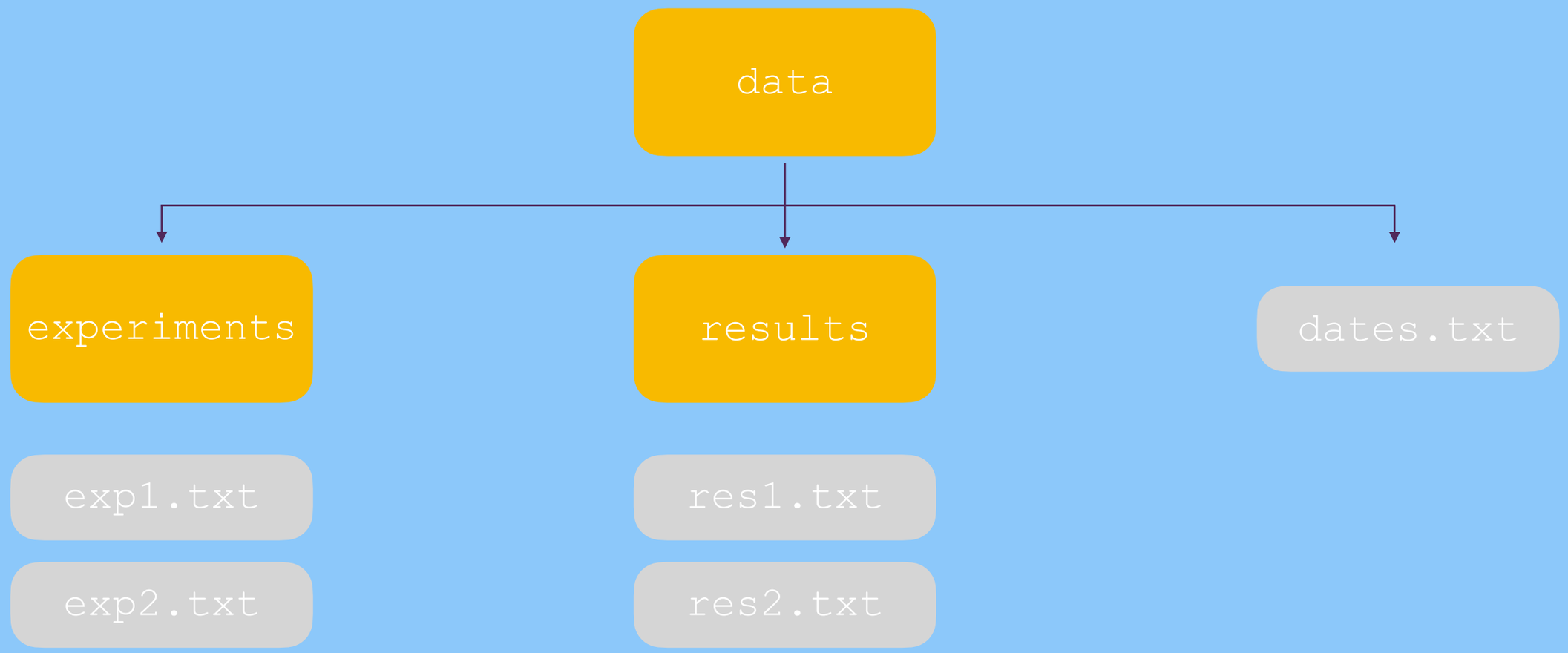
`$cp` copy a file

`$rm` eliminate file

`-r` recursive

👩‍💻 Practise CLI commands!

— Using the Command Line Interface create the following directory structure
— Add one line of text to each file



Git and version control

What is Git? What is GitHub?

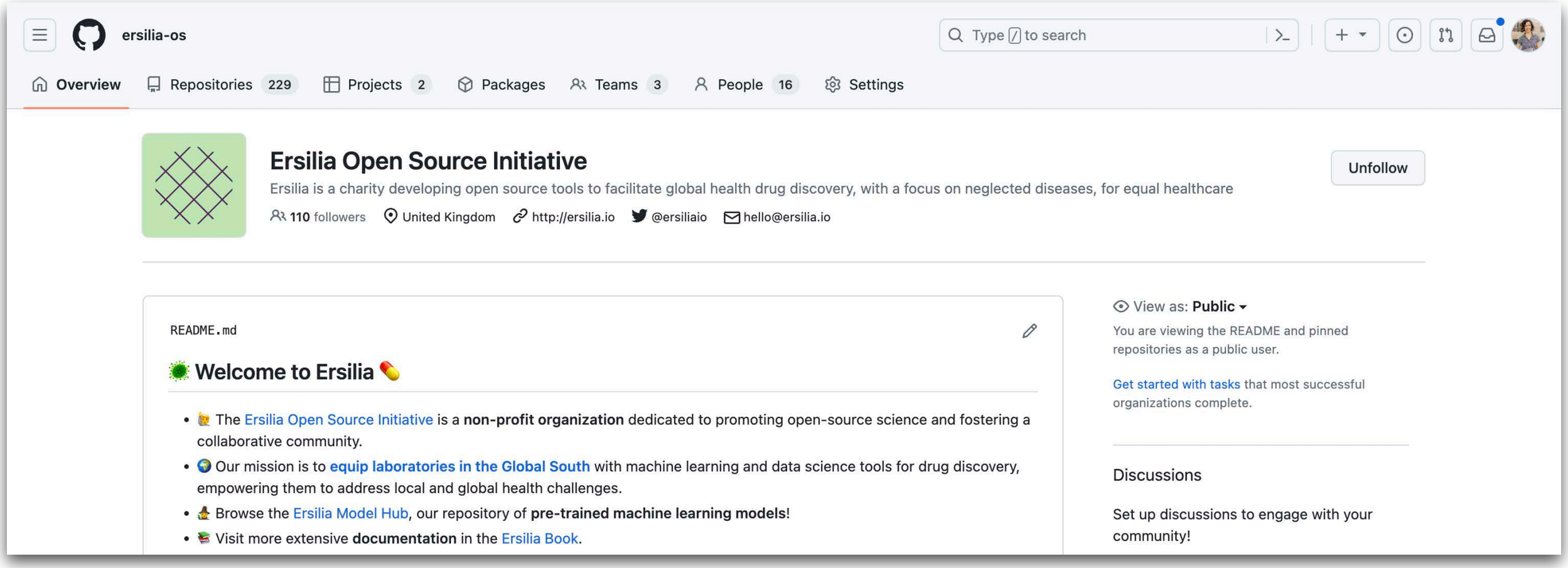
Git is a software that tracks changes to files, including creating new files, deleting files and folders, and editing content

GitHub is an internet hosting service for software development and version control using Git

👉 When can each be useful?

The importance of GitHub 🚀

- 100 million developers
- 372 million repositories (28 million are public)
- More than 20,000 academic papers are referenced in GitHub repositories



Installing Git on a Windows computer

— Get the latest Git: https://git-scm.com/download/win
— Double-click to install the downloaded file
— Adjust your PATH environment by selecting "Run Git from the Windows Command Prompt"
— Configure line ending by selecting "Checkout Windows-style, commit UNIX-style line endings"

Installing Git on a Linux computer

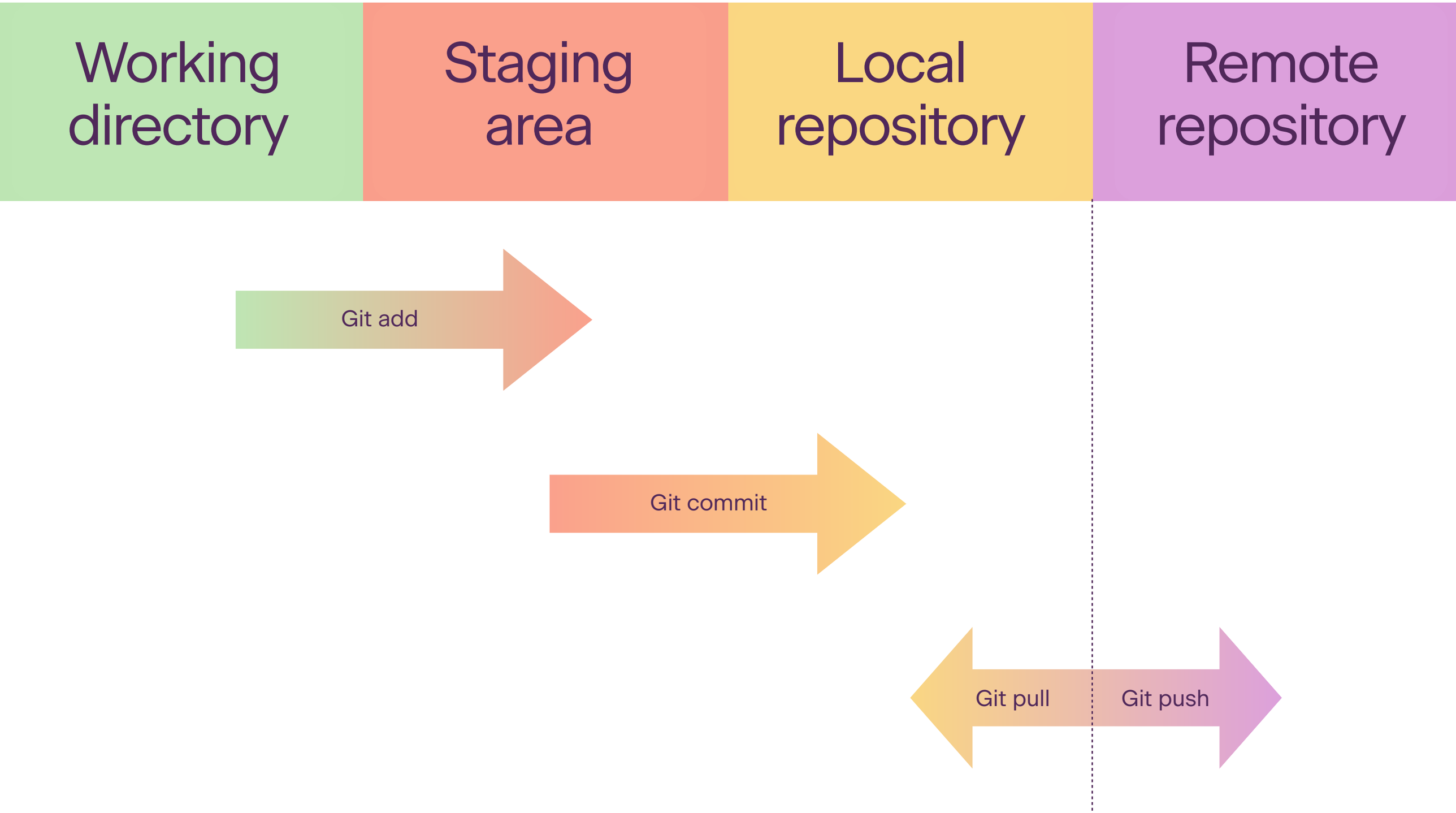
```
# Git is usually preinstalled in Linux
~/Desktop $ git
usage: git [--version] [--help] [-C <path>] [-c
<name>=<value>]
           [--exec-path[=<path>]] [--html-path]
[--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager]
[--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-
tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-
env=<name>=<envvar>]
           <command> [<args>]
These are common Git commands used in various
situations:
start a working area (see also: git help
tutorial)
clone Clone a repository into a new directory
init Create an empty Git repository or
reinitialise an existing one
work on the current change (see also: git help
everyday)
add Add file contents to the index
mv Move or rename a file, a directory, or a
symlink
restore Restore working tree files
…
```

— Run the Git command in the terminal to make sure it is installed

Installing Git on a MacOSX computer

- Check the install instructions: https://git-scm.com/download/mac
- Get Homebrew if you don't have it already
- In the Terminal, type: `brew install git`
- Check that Git works by typing the `git` command in the CLI

How does Git work?



Configure Git
Use the git command in the CLI

```
#first, let's check we have Git properly installed
~/Desktop $ git --version
git version 2.39.2 (Apple Git-143)
~/Desktop $

#Let's set our username (same as GitHub user)
~/Desktop $ git config --global user.name "Gemma Turon"
~/Desktop $ git config --global user.email "gemma@ersilia.io"
```

Start using Git
Explore the git subcommands (init, status...)

```
# Let's navigate to our folder test
~/Desktop $ cd test

# Convert the folder into a Git - TRACKED folder
~/Desktop/test $ git init
Initialized empty Git repository in /Users/username/Desktop/test/.git

# Check what is Git following in the folder
~/Desktop/test $ git status
On branch main

No commits yet

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

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

Start using Git
Explore the git subcommands (init, status...)

```
# Let's add a file to track
~/Desktop/test $ git add dates.txt
~/Desktop/test $ git status
On branch main

No commits yet

Changes to be committed:
(use "git rm --cached <file> ..." to unstage)
new file: dates.txt

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

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

Start using Git
Explore the git subcommands (init, status...)

```
# Let's add everything as tracked

# we use the dot (.) to indicate all

~/Desktop/test $ git add .
~/Desktop/test $ git status
On branch main

No commits yet

Changes to be committed:
(use "git rm --cached <file> ..." to unstage)
new file: dates.txt
new file: experiments/dates.txt
new file: experiments/exp1.txt
new file: experiments/exp2.txt
new file: results/res1.txt
new file: results/res2.txt
```

Start using Git
Explore the git subcommands (init, status...)

```
# Once everything is in the staging area, we can commit the changes

# We add a message to the commit so we which changes we are making

~/Desktop/test $ git commit -m "my first commit"
[main (root-commit) 184ba96] my first commit
7 files changed, 0 insertions (+), 0 deletions (-)
create mode 100644 dates.txt
create mode 100644 experiments/dates.txt
create mode 100644 experiments/exp1.txt
create mode 100644 experiments/exp2.txt
create mode 100644 results/res1.txt
create mode 100644 results/res2.txt
~/Desktop/test $ git status
On branch main
nothing to commit, working tree clean
```

Start using Git in the CLI

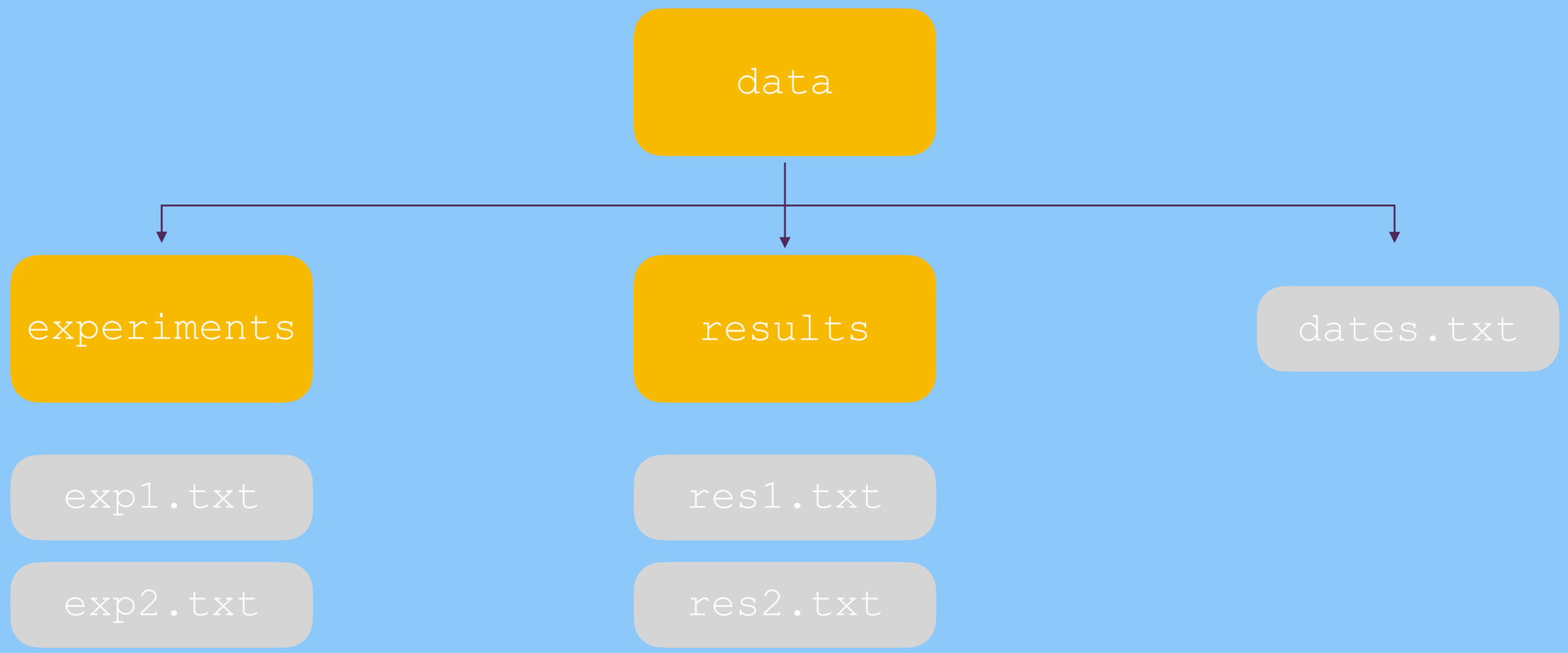
```
# Let's modify a file and see how we can track the changes
~/Desktop/test $ echo "my first line">dates.txt
~/Desktop/test $ git status
On branch main
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in workign directory)
modified: dates.txt

no changes added to commit (use "git add" and/or "git commit -a")
~/Desktop/test $ git add .
~/Desktop/test $ git commit -m "trying changes"
[main fe3a311] trying changes
1 file changed, 1 insertion(+)

~/Desktop/test $ git status
On branch main
nothing to commit, working tree clean
```

Start using Git
Explore the git subcommands (init, status...)

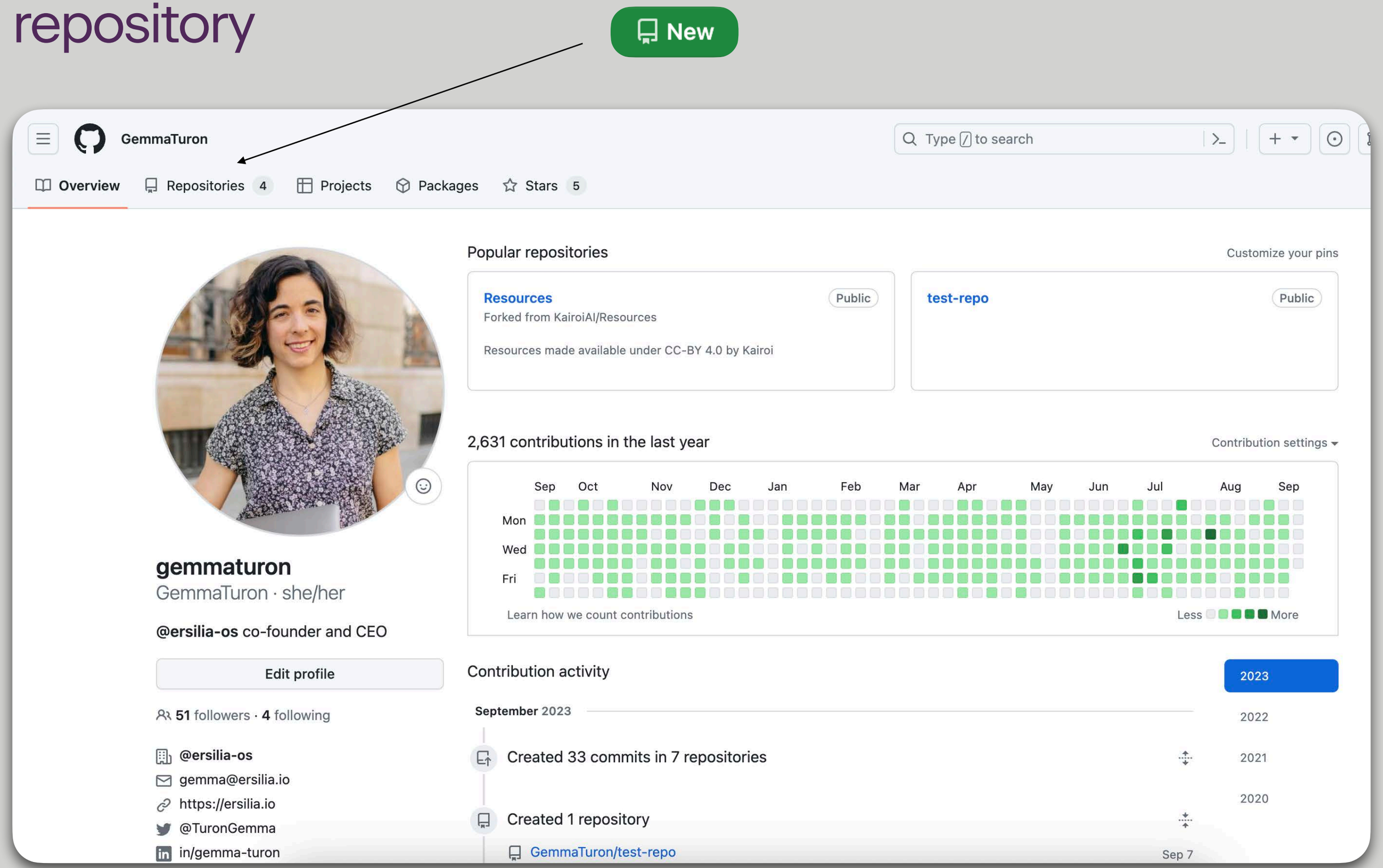
```
# Let's see how the history of changes is recorded
~/Desktop/test $ git log
commit fe3a3115944350b1045047aeb2cecd4a23d68e (HEAD -> main)
Author: Gemma Turon (gemma@ersilia.io)
Dte: Wed Sep 6 19:10:00 2023 +0200

trying changes

commit 184ba96df76f18097d2762c0d0596096455f621a
Author: Gemma Turon (gemma@ersilia.io)
Dte: Wed Sep 6 19:00:00 2023 +0200

first commit
```

👩‍💻 Practise Git commands!

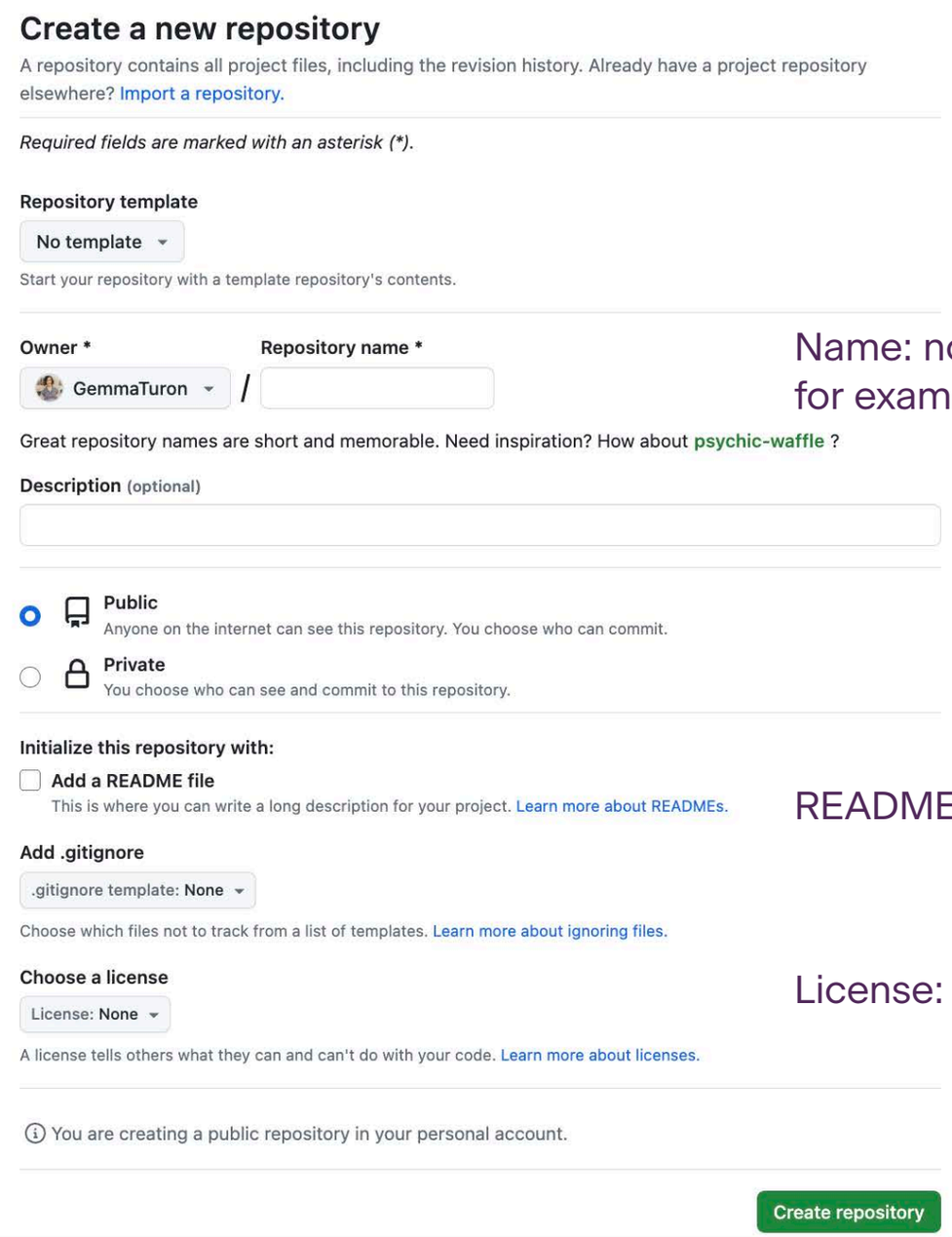
— Include some changes in your files manually or through the CLI and add them to the Git repository



Create a new GitHub repository



Create a new GitHub repository



Name: no spaces, use hypen to join words, for example `test-repo`

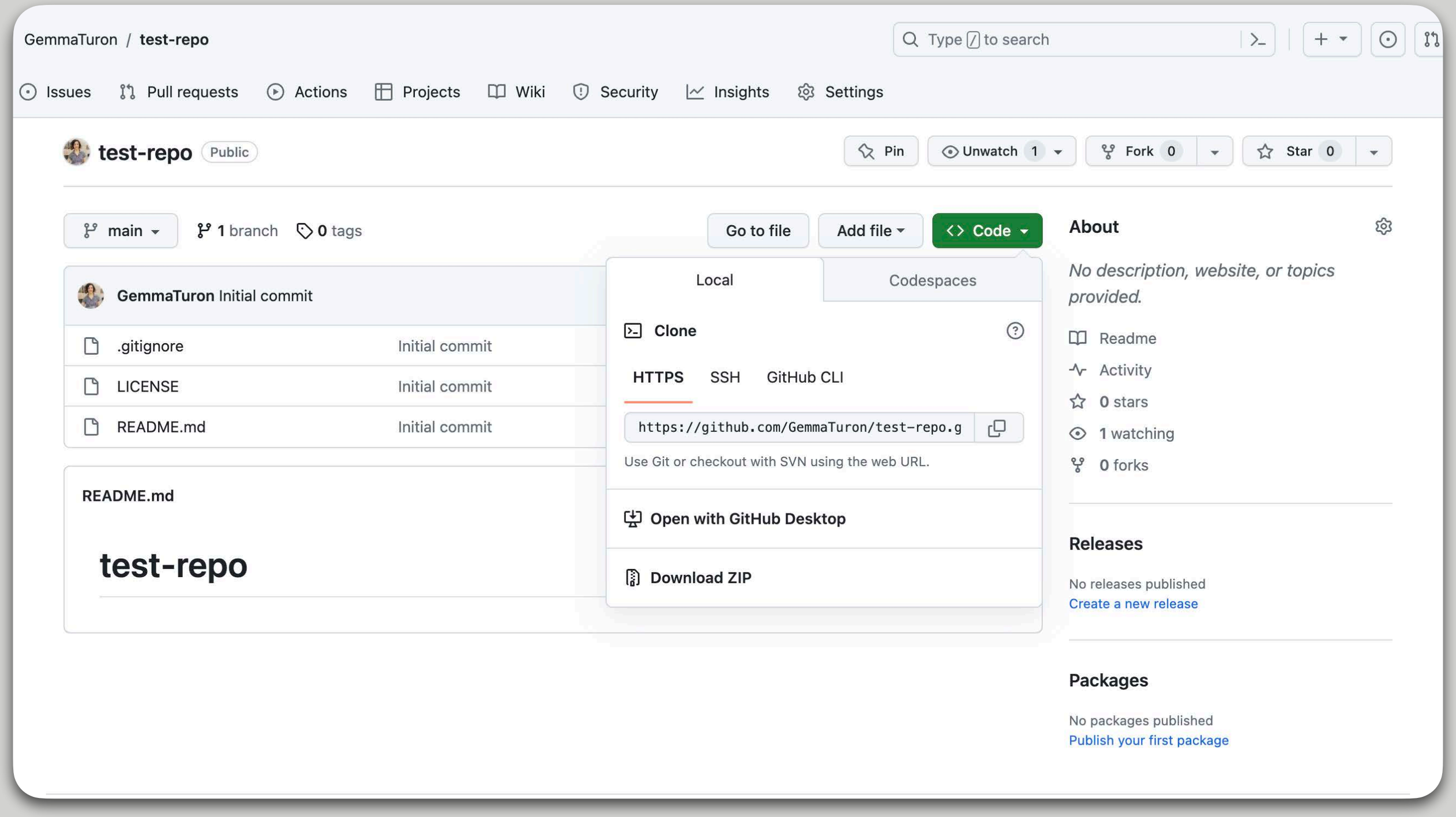
Private vs Public?

README: project descriptions

.gitignore: git ignores certain types of files created by the computer (i.e python-related files)

License: who can use this code and how?

Clone the repository in your local (directly as a Git folder)



Clone the repository in your local (directly as a Git folder)

```
# Let's go to our Desktop and clone the repository there
$ cd Desktop

# Use Https
~/Desktop $ git clone https://github.com/GemmaTuron/test-repo.git
# Use SSH
~/Desktop $ git clone git@github.com:GemmaTuron/test-repo.git

Cloning into "test-repo" ...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

~/Desktop $ ls
test-repo
test
~/Desktop $ cd test-repo
LICENSE      README.md
```

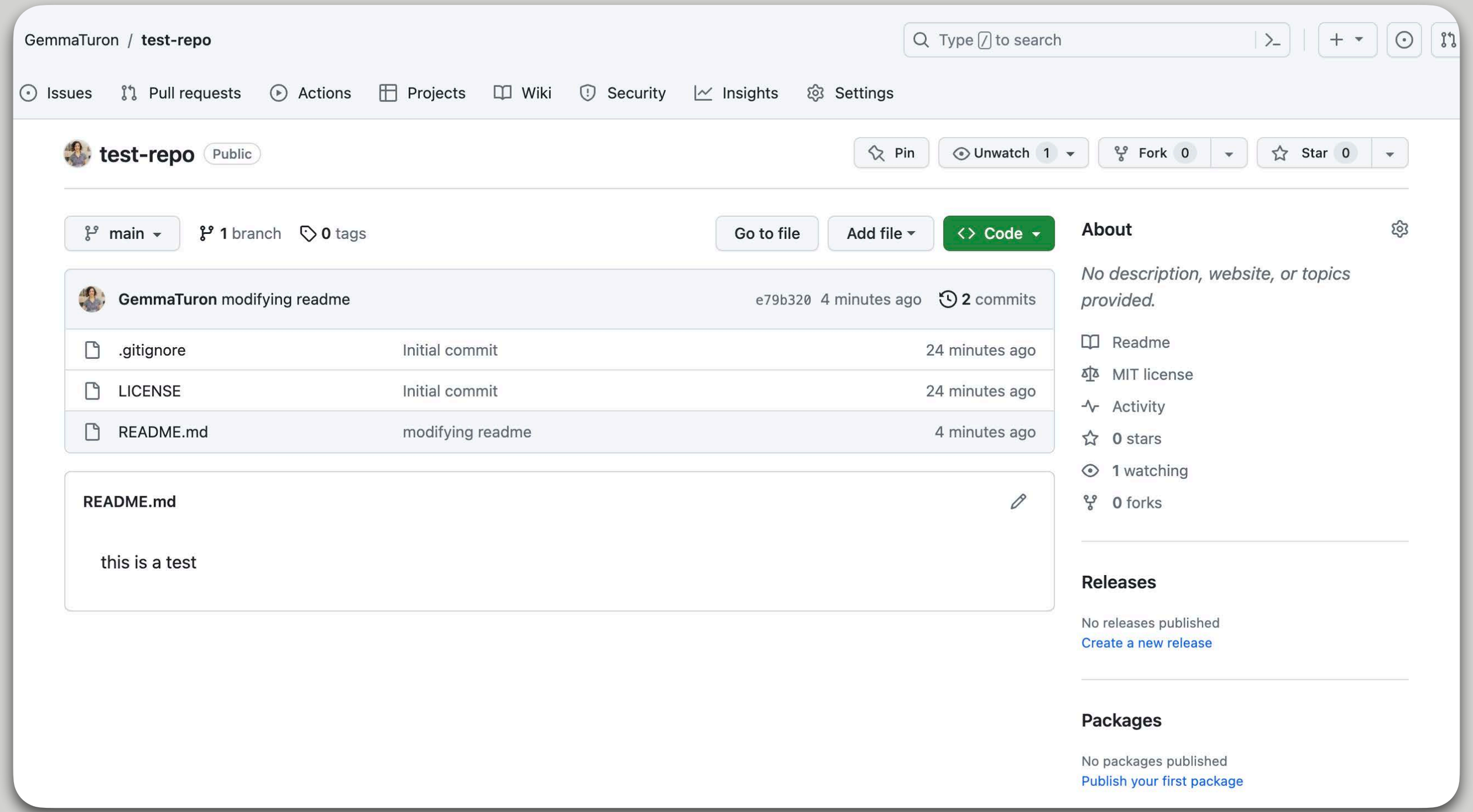
Make local changes

```
~/Desktop/test-repo $ echo "this is a test" > README.md
~/Desktop/test-repo $ 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

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

# Let's add the modified file to the staging area
~/Desktop/test-repo $ git add README.md
~/Desktop/test-repo $ git commit -m "modifying readme"
[main e79b320] modifying readme
1 file changed, 1 insertion(+), 1 deletion(-)
~/Desktop/test-repo $ git status
On branch main
Your branch is ahead of "origin/main" by 1 commit.
(use "git push" to publish your local commits)
```

Push changes to remote end & go online to see what happened!

```
# Let's push the changes to the remote end
~/Desktop/test-repo $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 267 bytes | 267.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:GemmaTuron/test-repo.git
9f83f39..e79b320 main -> main
```

Updated GitHub repository 🎉



👩‍💻 Practise Git commands!

— Go to your online repository and add a file directly by using the “Add File” button
— Explore the `git pull` command to get a copy of this new file in your local repository as well!

The Python programming language

Multiple programming languages

Programming languages are sets of rules that convert human-readable text to machine code
Their instructions are precise and only have one single meaning

— Human readable syntax
— Flexibility for several applications
— Lots of pre-built packages

Basic Python Concepts

- Variable: to store information, for example
  `x = 5`
- Data Types:
  - Numeric: `int` (integer) or `float` (decimal)
  - Text: `"string"`
- Lists: ordered list of items (strings, integers...) for example
  `["one", "two", "three"]`
  `[1,2,3]`
- Functions: reusable blocks of code to perform a specific task:

```
def sum_numbers(x, y):
  return (x + y)
```

Python packages

```
# Install package via manager (pip or
conda)

!pip install rdkit

# Import modules or functions from the
package

import rdkit
from rdkit import Chem


# Call a specific function to use it
molecule =
Chem.MolFromSmiles("CCCCOCCCC")
```

Python packages are collections of reusable Python modules that provide pre-written code and functionality. They simplify code reuse and maintenance.

Jupyter Notebooks

A notebook allows us to run small pieces of code at a time, making it very easy to perform data analysis.

Where can we run Jupyter Notebooks?

- Locally: we need python installed as well as a Jupyter system. The best is to do so via a Package Manager called Anaconda
- Remote computer: all the packages are installed in the remote computer and we access it via a CLI and/or GUI

Data science with Jupyter Notebooks

ersilia-os

cyp450.ipynb

open-data-cleaning > notebooks > cyp450.ipynb > import pandas as pd

+ Code + Markdown | Run All Clear All Outputs | Outline ··· Select Kernel

PUBCHEM AID1851

```python
df = pd.read_csv(os.path.join(DATAPATH, "PUBCHEM1851.csv"), low_memory=False)
```

Python

```python
# remove substances without associated smiles
print(len(df))
df = df[~df["smiles"].isna()]
print(len(df))
```

Python

```python
#separate by cyp
cyps_1851 = {
            "cyp2c9" : df[df["Panel Name"]=="p450-cyp2c9"],
            "cyp2c19" : df[df["Panel Name"]=="p450-cyp2c19"],
            "cyp2d6" : df[df["Panel Name"]=="p450-cyp2d6"],
            "cyp3a4" : df[df["Panel Name"]=="p450-cyp3a4"],
}
```

Python

```python
for k,v in cyps_1851.items():
    delete_duplicates(v)
    keep_actinact(v)
    pubchem_outcomes(v)
    cyps_1851[k] = v
```

Python

```python
for k,v in cyps_1851.items():
    v['cid'] = 'CID' + v['cid'].astype("str")
```

main 0 1 0 Live Share Cell 1 of 25

GitHub Codespaces

Codespaces is a cloud service provided by GitHub which gives users 60h of free cloud computing time per month.

We will use it throughout the course, why?

— Integrated within GitHub, where all our code resides
— Allows for preinstallation of packages

👉 Sign into your GitHub account and go to https://github.com/ersilia-os/ersilia-intro-workshop

Let’s recap!

In this module we have heard about...

— The command line interface (CLI)
— Git and GitHub
— Python and Jupyter Notebooks

🙇‍♀️ Go to the course repository!
🚀 Try GitHub Codespaces!