Data Science

Session 2 - Clean code & Git


hadrien.salem@centralelille.fr


introduction-to-data-science

Introduction

What did we do last time?

The CRISP-DM method

Cross-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining

- ➔ Published in 1999
- ➔ Common in the industry
- ➔ Still relevant today

The CRISP-DM method to carry out data-driven projects

(Image source: Wikipedia)

Course outline

Data science course

Session 1: Understanding data

Session 2: Clean code & Git

Session 3: Preparing data - Cleaning & Missingness

Session 4: Preparing data - Dimensionality reduction

Session 5: Preparing data - Data imbalance

Machine learning course

What the f*** does this have to do with Data Science?

Why learn about clean code and collaborative development in a Data Science course?

Data scientists produce code

- Research is pointless if you are not able to share your code with the people who will deploy it

Data scientists do not work alone

- Your code is pointless if you are the only one who can understand it

Version control systems are everywhere in the industry

- Even as a manager, it is good to understand how your tech team manages their code

Open source projects are a goldmine for tools

- Platforms like GitHub and GitLab are what allow open source projects to grow

It's a good way to manage your project deliverables

- The sooner you learn, the more efficient you will be throughout your Master's

You are never coding alone

You are already working with the people who will come after you

That could include yourself!

⇒ Good code is code that can be used
for a long time by all members of the project

The foundations of “good” code

These three pillars constitute the main elements to writing successful code.

Correct
The code does what it is supposed to do in all situations

Maintainable
The code is understandable and robust to changes

Extensible
The code is flexible and can accommodate additions

Outline

Clean code and collaborative development

Part 1 - Writing clean code

Part 2 - Version control and code sharing

Part 3 - Project management and VCS

Writing clean code

Five principles to follow to build robust projects

Principle 1

Clarity

Writing code that speaks for itself

An example of unclear code

```
1 df = pd.read_csv("C:/super/long/path/to/my/file/please/help/me.py")
2 df.drop(df.columns[0], axis=1)
3 df["price_eur"] = df.price_dollar * 0.9
```

What does line 2 even do?

I have a different path on my PC!!

What was axis=1 again? I need to check the docs...

Why 0.9??

What even is inside my dataframe?!

Same code, after corrections

```
1 PRICES_DATA_PATH = "C:/super/long/path/to/my/file/please/help/me.py"
2 dollar_to_euro_conversion_rate = 0.9
3
4 item_prices = pd.read_csv(DATA_PATH)
5 item_prices.drop("item_id", axis="columns")
6 item_prices["price_euro"] = df["price_dollar"] * dollar_to_euro_conversion_rate
```

- Variables have been **renamed** to be more **explicit**
- Arguments are **clearer** and **consistent**
- **Magic numbers** are gone

Code clarity

Ideally, code should be self-explanatory: you should understand what it does just by reading it linearly

Names should be...

- Descriptive
- Unambiguous
- Consistent

* If writing long names is a problem to you, please use a better IDE
> VSCode is free to use, PyCharm has a free student license

Do not use magic numbers

Numbers serve a purpose, **call them by name** instead.
The same principle applies to strings and other objects.

Use functions to your advantage

Replace blocs of code with **reusable** functions
Avoid long, difficult to navigate scripts at all costs

Principle 2

Documentation

The keys to understanding your code

Code without docs is like a map without directions

The information is there, but you don't understand the intent behind it

Paris metro map, 1973 (left) vs. today (right)

Documentation inside your code

```python
def complex_operation(a, b):
    # Add 1 to a
    new_a = a + 1

    return new_a + b
```

```python
def complex_operation(a, b):
    # Offset by one day
    new_a = a + 1

    return new_a + b
```

If your code is clear,
avoid adding noise to it

Use comments to…
➔ explain **intent**
➔ **clarify** difficult code
➔ **warn** developers

Do not comment useless code,
remove it. Prefer VCS for backups.

Documentation outside your code

External documentation can have several purposes

➔ **For engineers**

Explain **solution architecture** and **theory**

➔ **For developers**

Explain **how to contribute** to the project

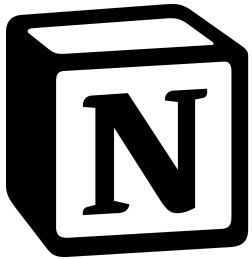
➔ **For users**

Explain **how to use** the project

Documentation outside your code

Writerside

JETBRAINS IDE



Prefer online tools that are easily
accessible, **maintainable** and **shareable**

Make sure everyone in your team knows where to find documentation when they need it

Do NOT wait until the end of a project to document it

Documentation will help you and is
an essential **communication tool** in your project.

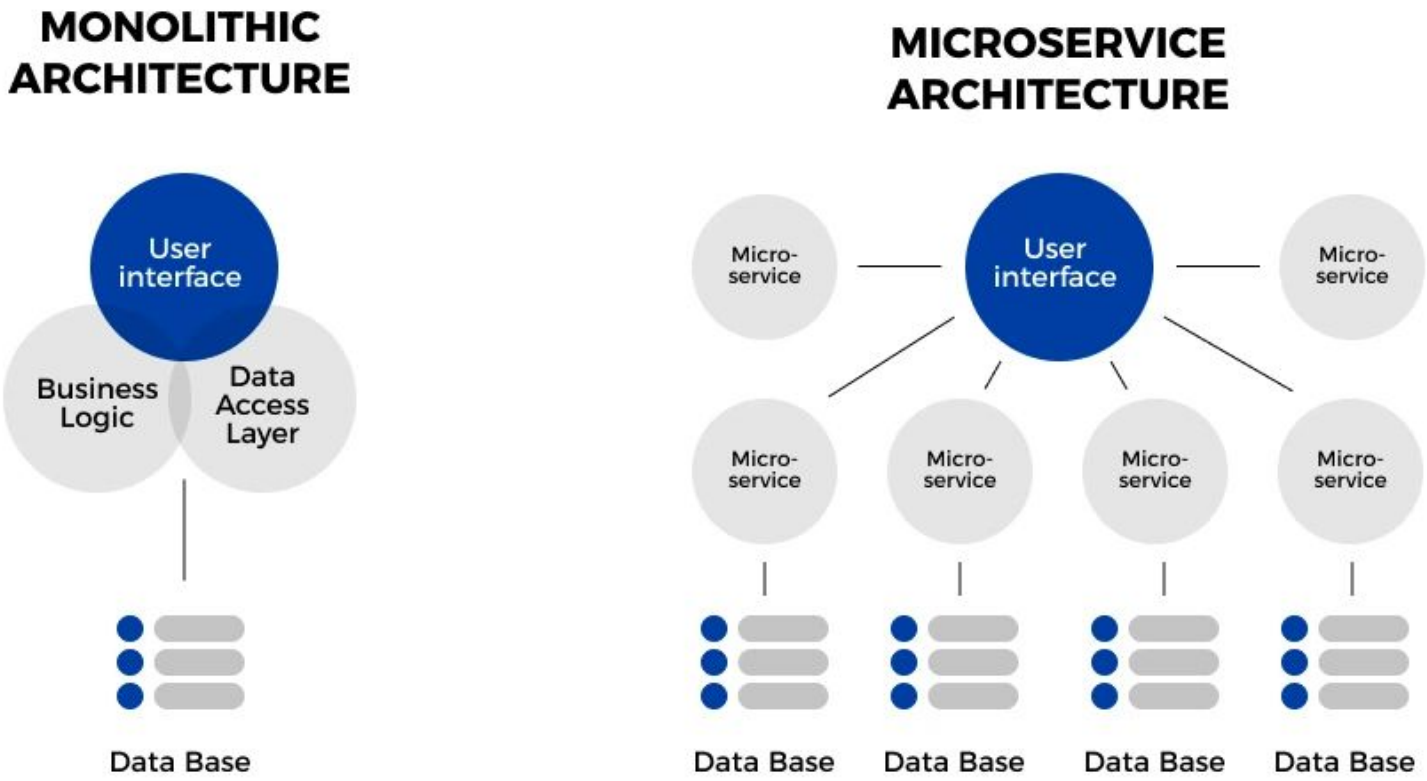
Write down your goals, your conventions,
and make sure anyone can reference them when they need it

Principle 3

Modularity

Thinking code in blocks

“Combining mechas” are a typical example of modularity



An example in web development: Microservice Architectures

Modularity

Modularity goes hand in hand with clarity and is key to **maintainable and extensible code**

Modular code is easier to modify and test, and more robust to scaling

DRY - Don’t Repeat Yourself

Isolate reusable parts of your code
Duplicated code is extremely difficult to maintain

KISS - Keep It Simple, Stupid

Break down your code into **smaller**, **independent** parts

Use package structure to your advantage

Use files, folders and classes to organize your code and make it easier to navigate

Principle 4

Reproducibility

If your code only works on your computer, it doesn’t work

The exact method used to construct the pyramid remains a subject of debate
In other words, it would be difficult to rebuild them (using the same tools)

Reproducibility

If you are the only one who can run your own code, it is effectively unusable

Always make sure your code can run anywhere

Several factors can cause unexpected behavior
(e.g. OS, library version, hardware, etc.)

- ➔ In Python, share your virtual environment using **pip freeze**
- ➔ For advanced developpers, look into **Docker**

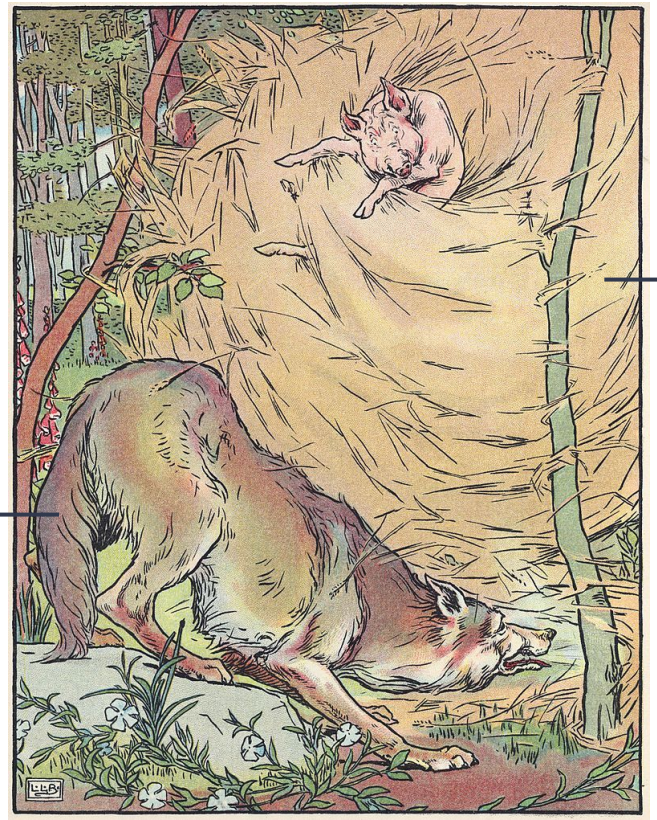
Save the parameters of your experiments

Use configuration files and document your methods

Principle 5

Testing

Nothing ever works on the first attempt



Your code

Any user that isn’t you

The wolf blows down the straw house in a 1904 adaptation of *The Three Little Pigs* - Illustration by Leonard Leslie Brooke

Don’t wait until the big bad wolf blows on your code to find out whether it will actually resist!

NOTHING ever works the way you intended it

There will be **bugs** and unforeseen **edge cases** in almost *every* piece of code you produce

This is perfectly **normal**, and you should prepare yourself to deal with it

Testing

Nothing ever works the way you intended it… So make sure you test everything you develop

Find bugs before they can bite you

Some bugs remain undiscovered until later in the lifecycle of your code… These are extremely difficult to solve. Test as much and as regularly as you can!

How to test code

Don’t be nice to your program -- try your hardest to break it. Test the “happy path”, of course, but don’t forget about the edge cases. If you don’t find them, your users will.

Unit testing is your best friend

They automatically test small parts of your code
Unit tests are the best way to protect yourself from side effects

They take time to write*, but the **return on investment is colossal**

**Generative AI performs well with these repetitive, time-consuming tasks*

Summary

Five principles to help you write correct, maintainable and extensible code

Clarity

Documentation

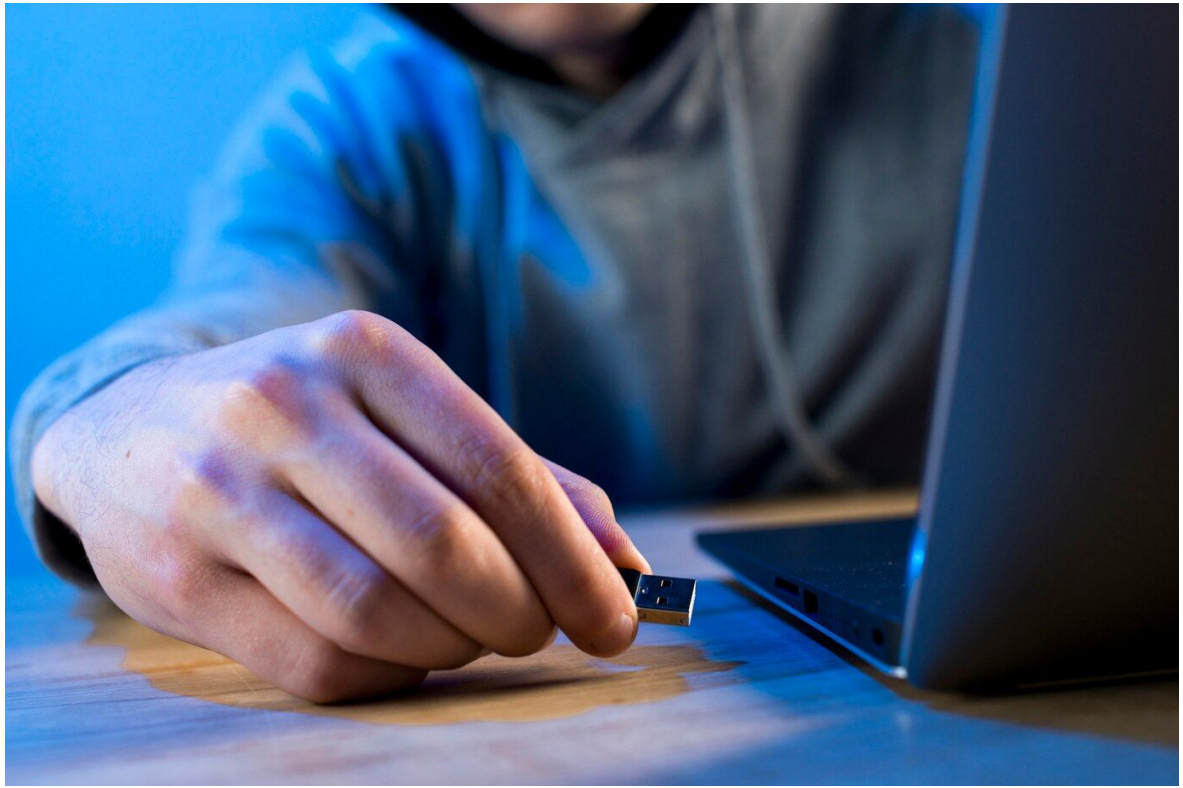
Modularity

Reproducibility

Testing

Version control and code sharing

Working on code as a team



Some of you may have done this to share code in the past...

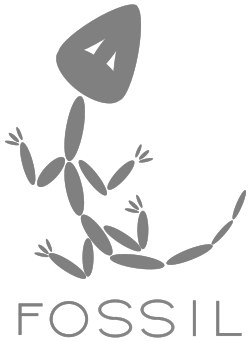
Using the wrong tools to share your code can lead to disastrous consequences
(Note: Google drive is no better, and Messenger conversations are an insult to proper version control)

Git is the solution to all these problems

Vocabulary : Git, GitHub, GitLab and more

What is the difference?

Vocabulary : Git, GitHub, GitLab and more



Version control systems

Hosting services for git projects

What are version control systems?

They are systems that allow the management of different versions for one or several files

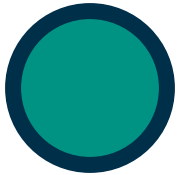
Simplify code storage

Simplify code versioning

Keep a history of changes

Parallelize work

Histories, commits and branches



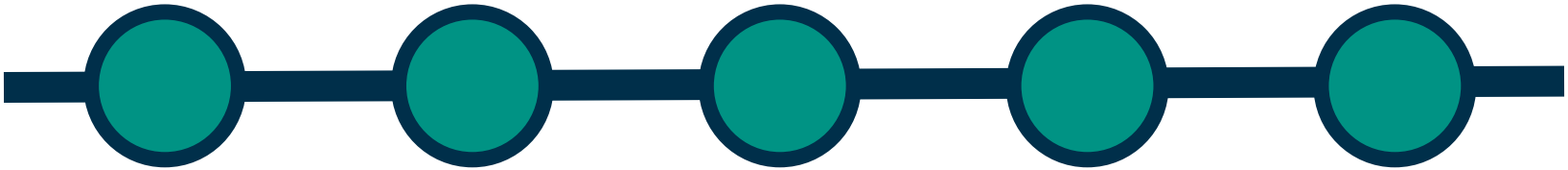
```
Commit 0000000000000000001
Author: Joe Mama <joe.mama@centralelille.fr>
Date: Fri Jul 27 15:29:25 2023
Message: Create slide 15 for GitHub session
+           slide15
-           slide25
renamed     slide11 > slide 12
```

A **commit** is a list of changes.

Committing your work is **saving a version of the repository**.

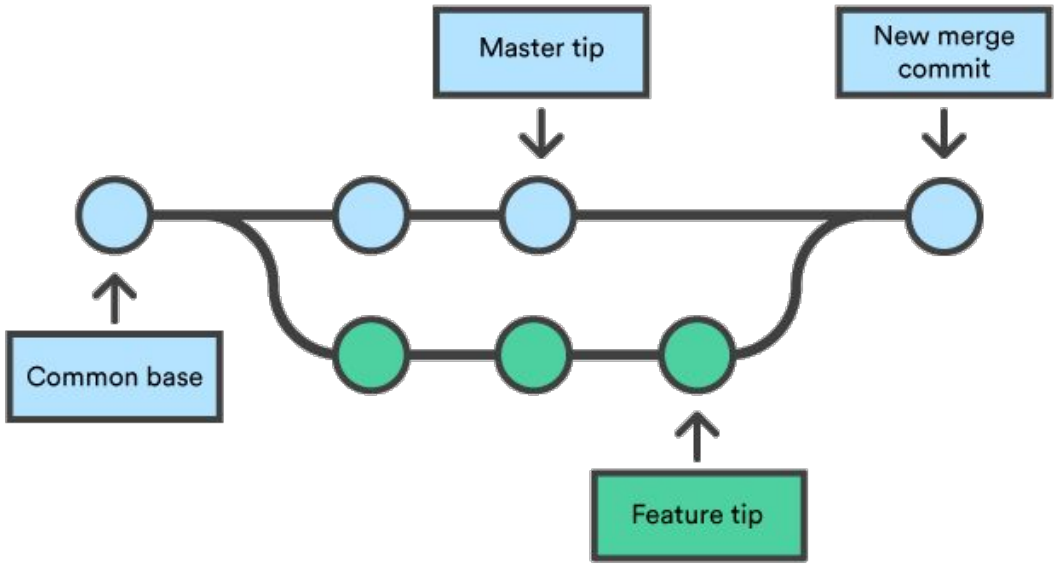
You can revert your code back to any commit at any time.

Histories, commits and branches



A **branch** is a sequence of commits.
They are a convenient way to manage your commits and their history.
Commits and branches can be stored in a shared **repository**.

Managing branches

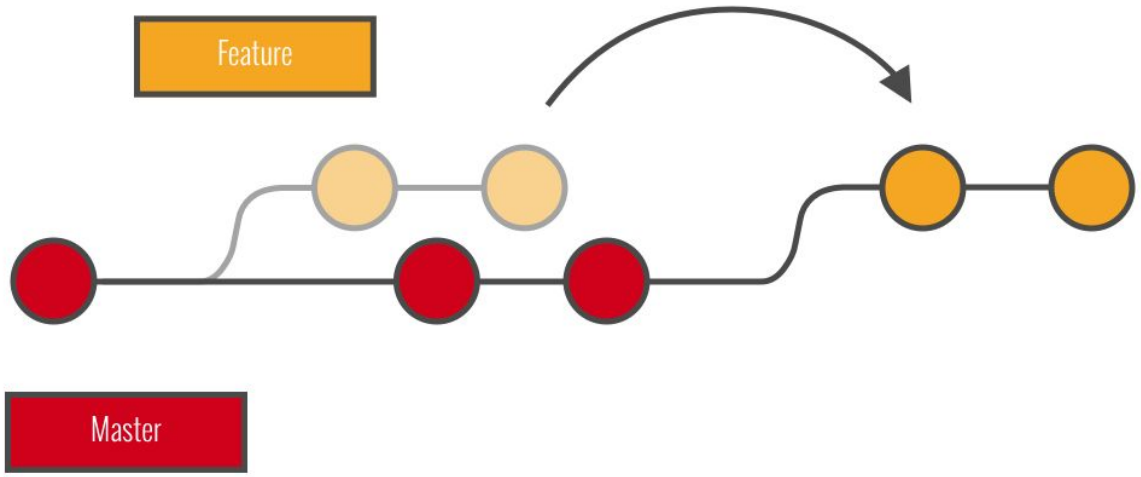


Branches can be created from other branches to work in parallel.

Bringing back commits from a branch to the other is called **merging**.

Conflicts are immediately identified by git.

Managing branch histories



Rebasing a branch is **redefining its origin** by rewriting the commit history.

It can help having a **clean history**.

It can however generate **conflicts**.

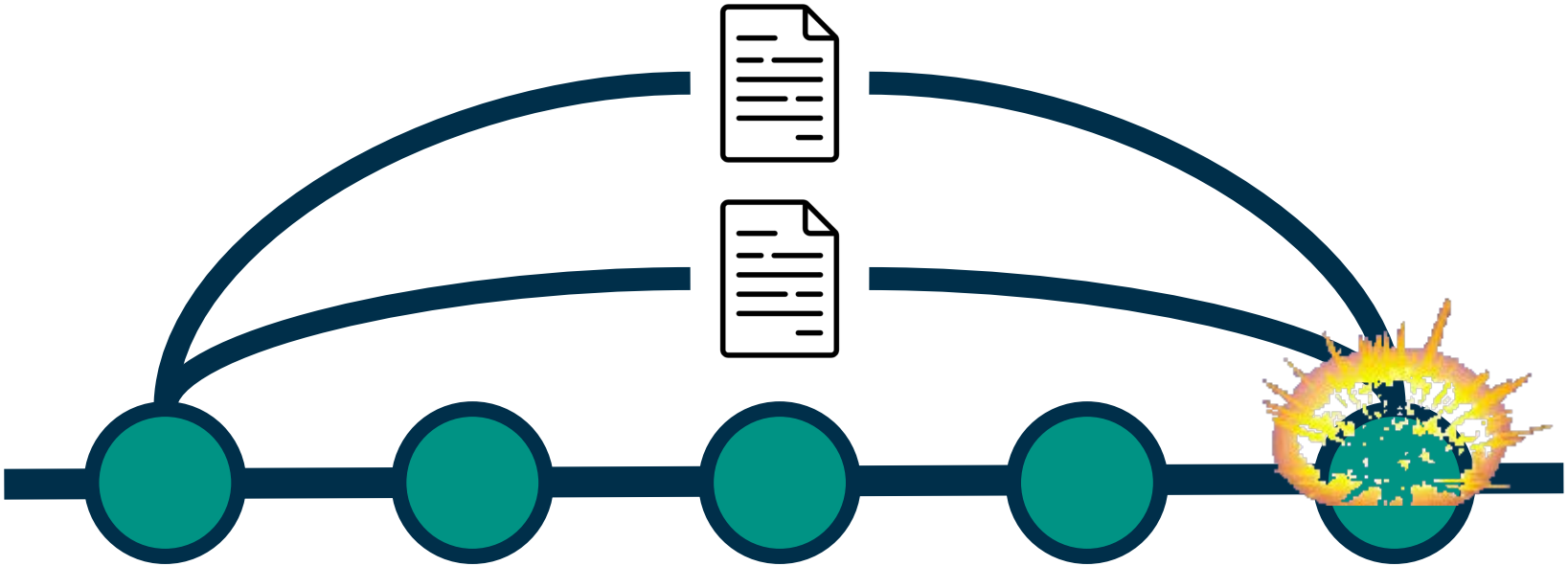


NEVER CHANGE A SHARED BRANCH'S HISTORY

You WILL regret it. Force push responsibly.



Managing conflicts



Conflicts happen when **histories do not match**.
This happens when **several people change the same file** and try to merge their work.

Practical work

Let’s build a mock app with Git!

Goals

We will set up git for everyone, and learn how to use it in a day to day context.

1. Get started with git and GitHub

- Install git
- Create a GitHub account
- Initialize your first (?) repository

2. Learn how to use git

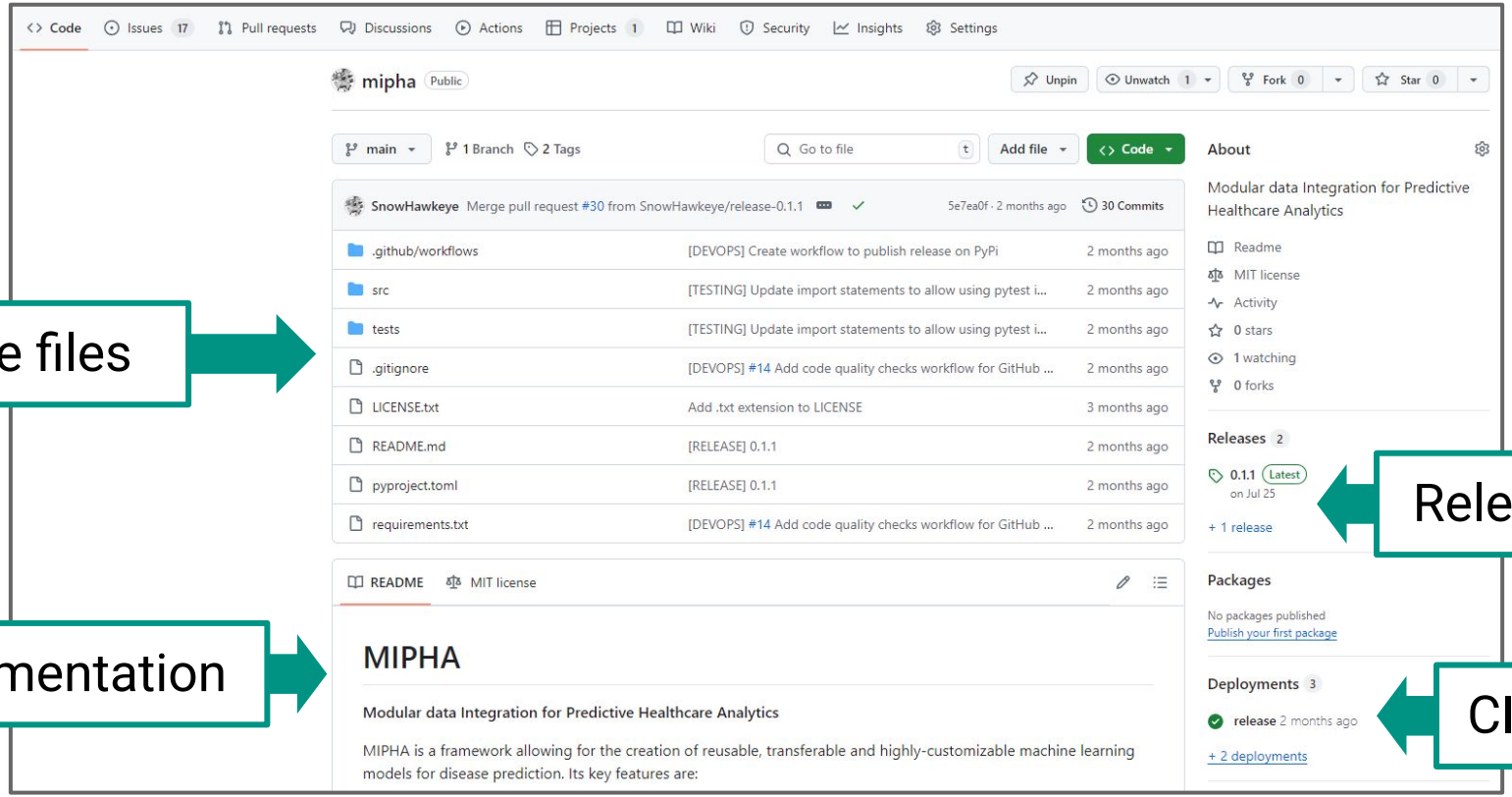
- Navigating branches
- Committing your changes
- History manipulation and conflict management

3. Setting up git for the rest of the course

- Create a dedicated organization and repo
- Connect with collab (or your IDE)

Integrating VCS with project management

Tools and methods to collaborate efficiently



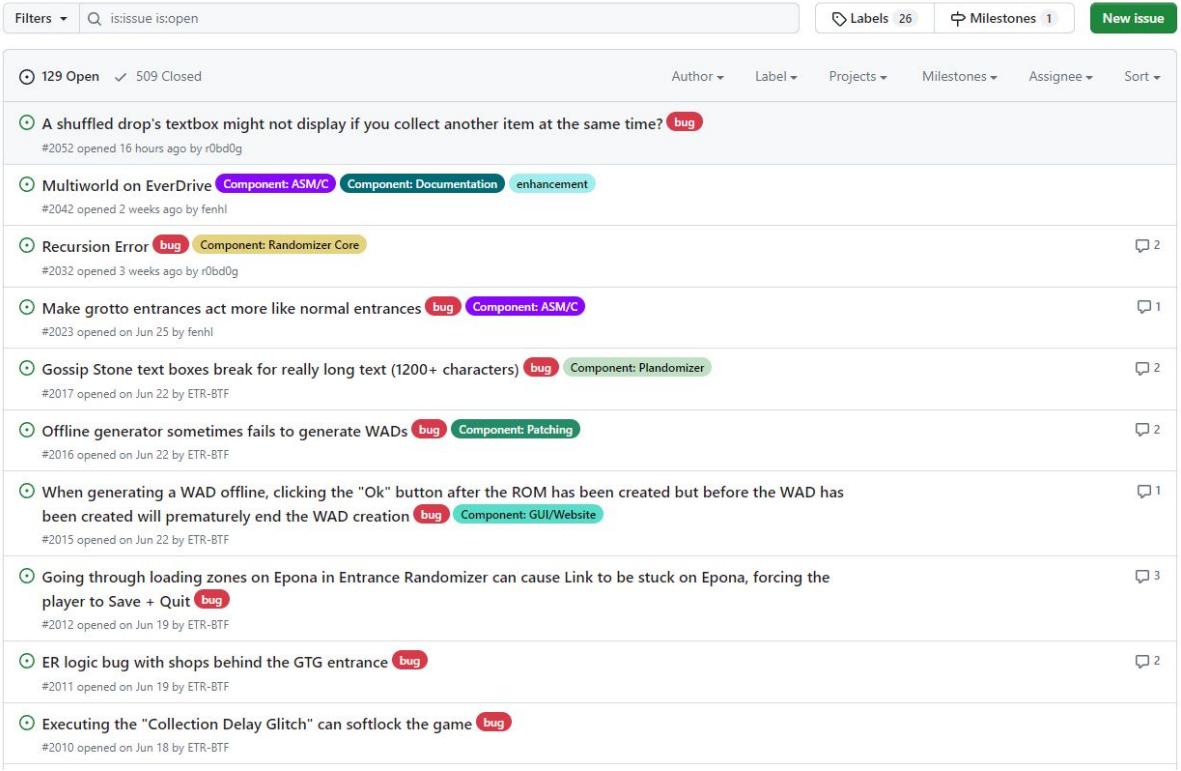
GitHub is an all-in-one solution to share and organize your code

Other services include…

- Tasks / tickets management
- Pull requests and code reviews
- Communication tools
- Continuous integration monitoring
- Documentation via a wiki

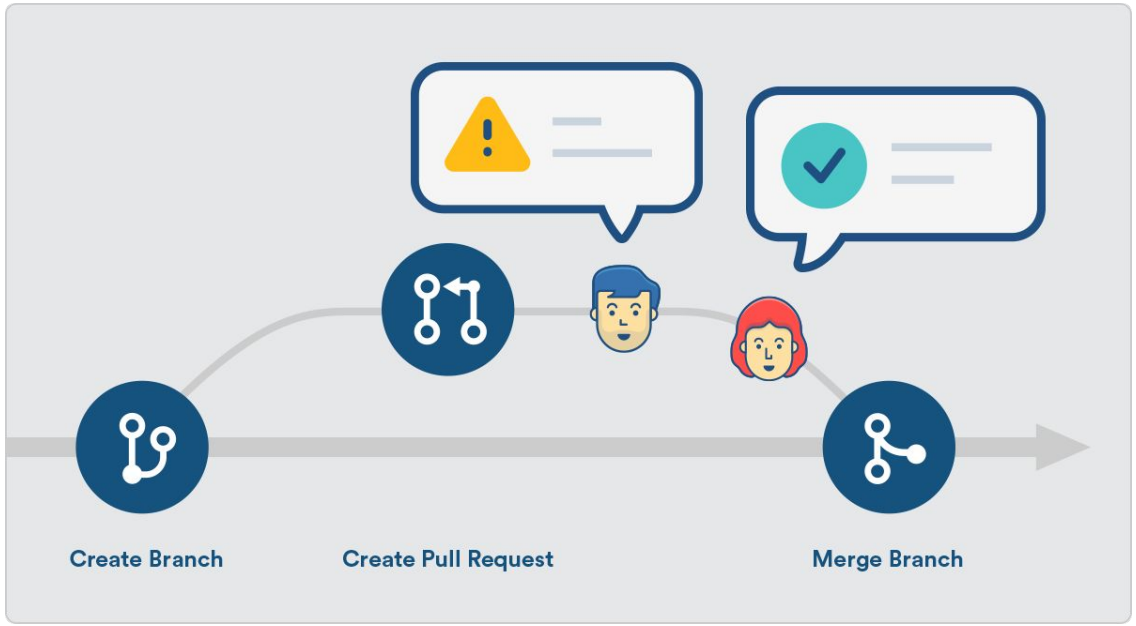
⇒ **You can centralize everything related to your code project**

Issues



Issues are akin to **tasks**.

Pull requests

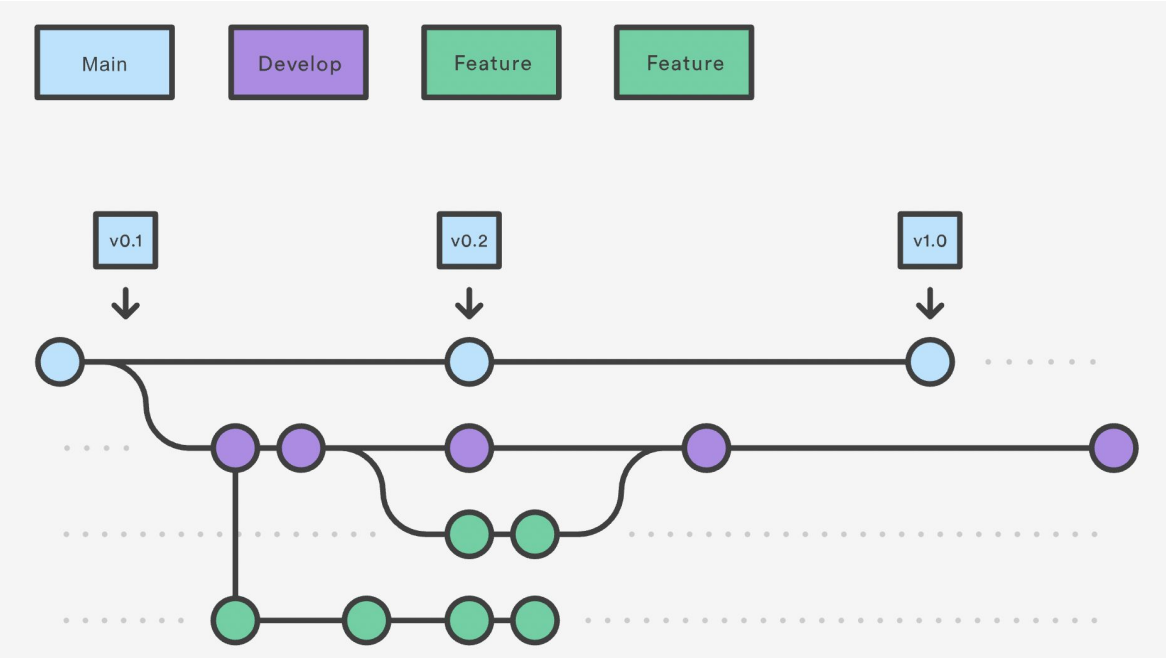


Pull requests are a **communication tool** between developers.

They allow for:

- Better code quality
- Knowledge sharing
- Communication

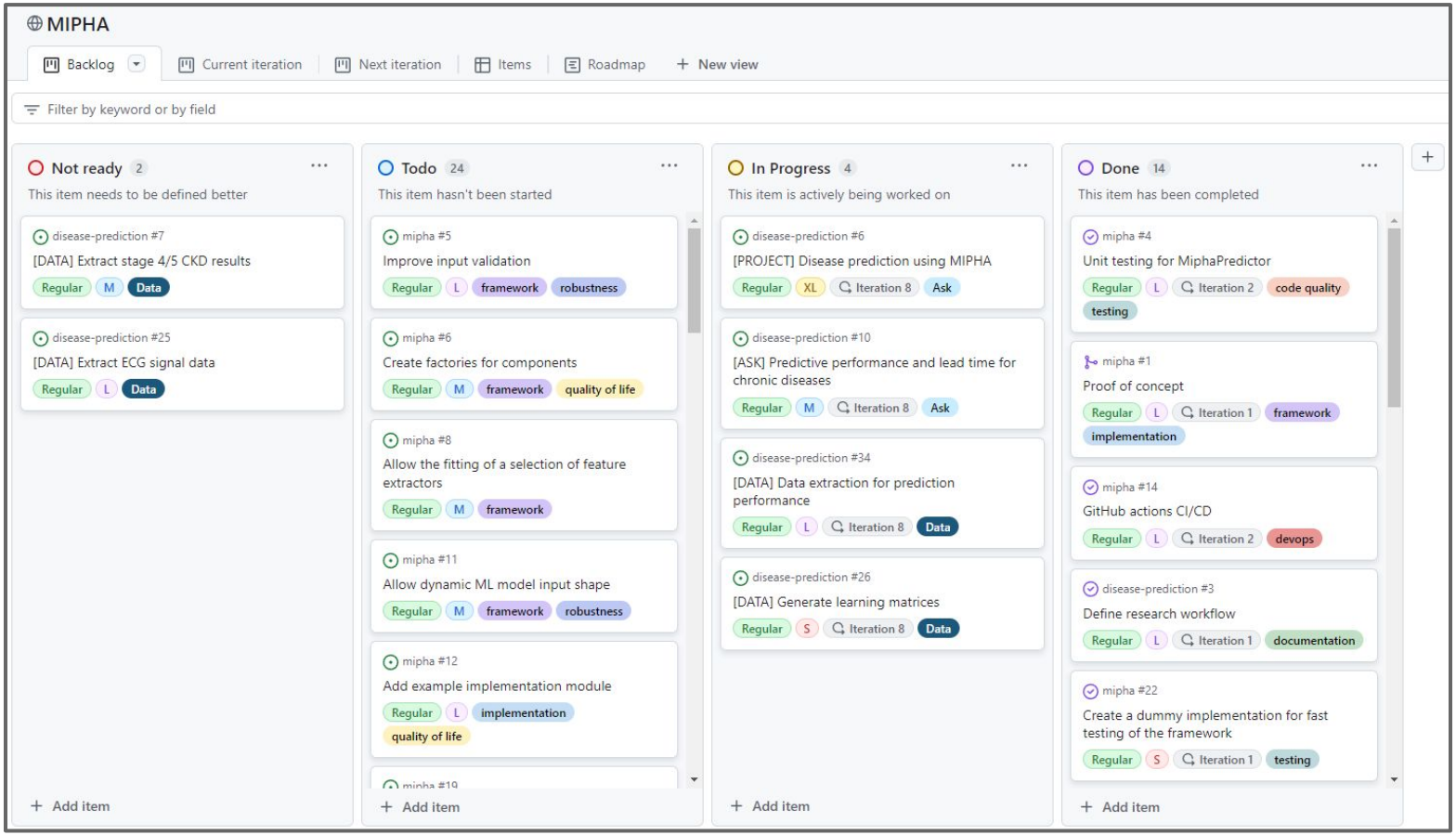
Defining a git flow



Git flows are **conventions** within development teams to mange their repository.

You should **define** and **write down** your conventions as part of your project's documentation.

(Your auditors will like that a lot, too!)



GitHub Projects is a great project management tool (see it in action!)

Closing words
What you should remember

Summary

What we saw today was an introduction to the basics of clean code and collaborative development

You will ***need*** some time to learn to apply these principles -- the resources in this slide are short reads that may help

But most importantly, **practice** makes perfect! Pick the right tools for your project, and use them regularly

How to produce clean code

Clarity, **documentation**, **modularity**, **reproducibility**, **testing** are all essential principles to produce **correct**, **maintainable** and **extensible** code.

Read: Summary of “Clean code” by Robert C. Martin

What Git is, and how GitHub helps you share code

Version control and code sharing are both extremely important. Using them regularly as part of your workflow will help you greatly.

Read: This tutorial I made (in French) on the basics of git
(includes a cheatsheet with the most important commands)
and this website to practice

How to integrate VCS with project management tools

Version control is a powerful tool, but it needs to be orchestrated by a robust workflow. Your auditors will also like the use of professional project management tools!

Debrief

Debrief

What did we learn today?

What could we have done better?

What are we doing next time?

Data Science

Session 2 - Clean code & Git


hadrien.salem@centralelille.fr


introduction-to-data-science