# Data Science

## Session 2 - Collaborative development


hadrien.salem@centralelille.fr


introduction-to-data-science

# Introduction

# What did we do last time?


## The CRISP-DM method

**Cr**oss-**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: Collaborative development**

**Session 3: Preparing data - Managing missing data**

**Session 4: Preparing data - Dimensionality reduction**

**Session 5: Imbalanced data and deidentification**

**Session 6: Working with text**

## Machine learning course

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

# Why learn about 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**

- Research 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 more and more popular**

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

**It's a good way to collect your works in class :)**

- I do need to evaluate your practicals. I'm sorry.

# What are version control systems?

# 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**

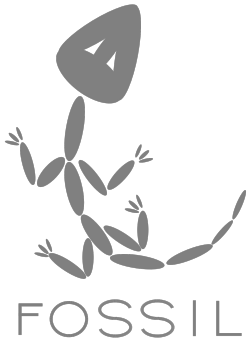
**Parallelize work**

# Vocabulary : Git, GitHub, GitLab and more


**What is the difference?**


# Vocabulary : Git, GitHub, GitLab and more




**Version control systems**

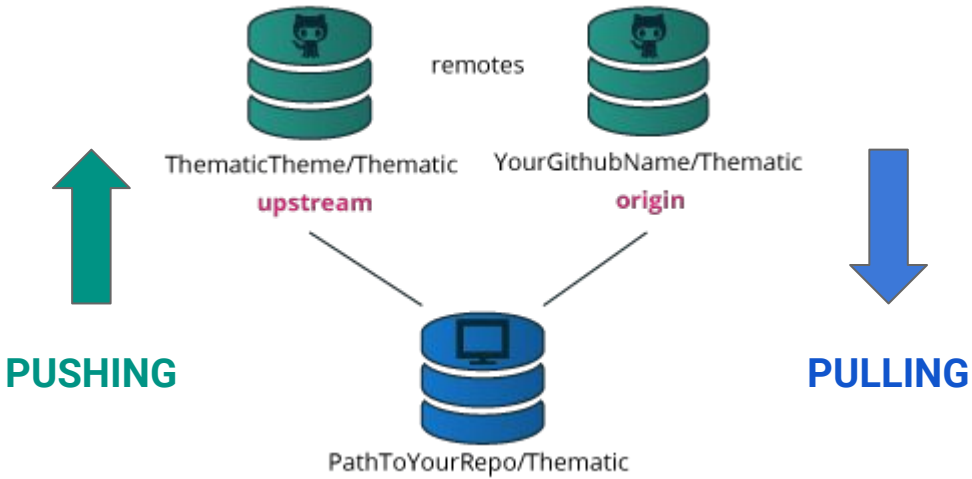




**Services to store git projects**

# Fundamental git concepts

# Repositories and remotes




A **git repository** contains a `.git` folder

A **remote** is a **repository** hosted on the cloud.

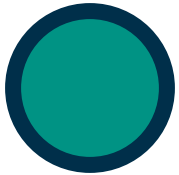
**NOTHING IS UPLOADED TO THE REMOTE UNTIL YOU PUSH IT YOURSELF!**

**You do not need a remote** to work with git.

Your local code can have **several remotes.**

A **fork** is an **independent copy** of a repository.

# Histories, commits and branches

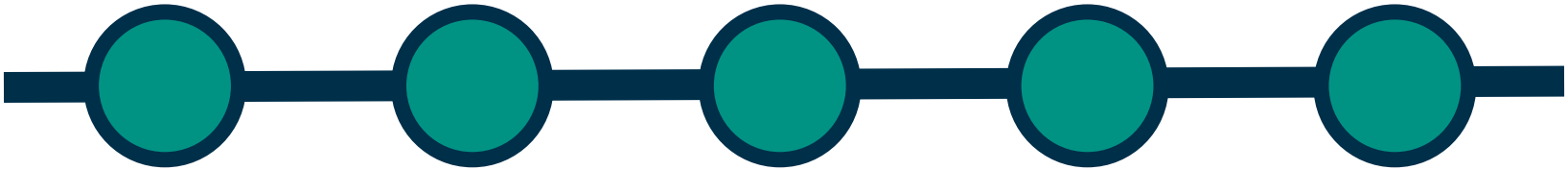


```
Commit 0000000000000000001
Author: Joe Mama <joe.mama@centralelille.fr>
Date: Fri Jul 27 15:29:25 2023
Message: Create slide 15 for GitHub session
+          slide15
-          slide25
renamed    slide11 > slide 12
```

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

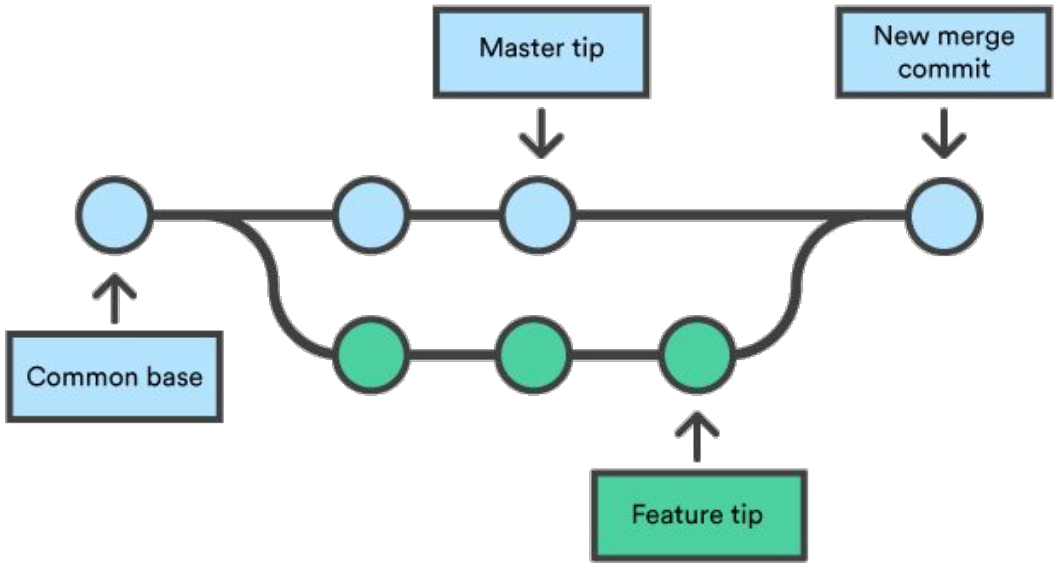
Committing your work is **saving a version of the repository**.

# Histories, commits and branches



A **branch** is a sequence of commits.
They are a convenient way to manage your commits and their history.

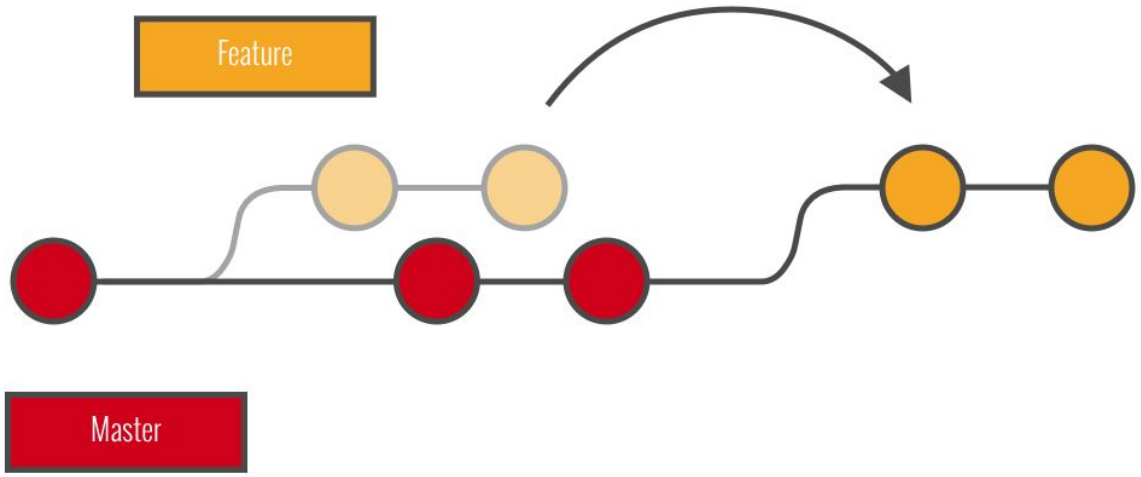
# Managing branches




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

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

# 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**.

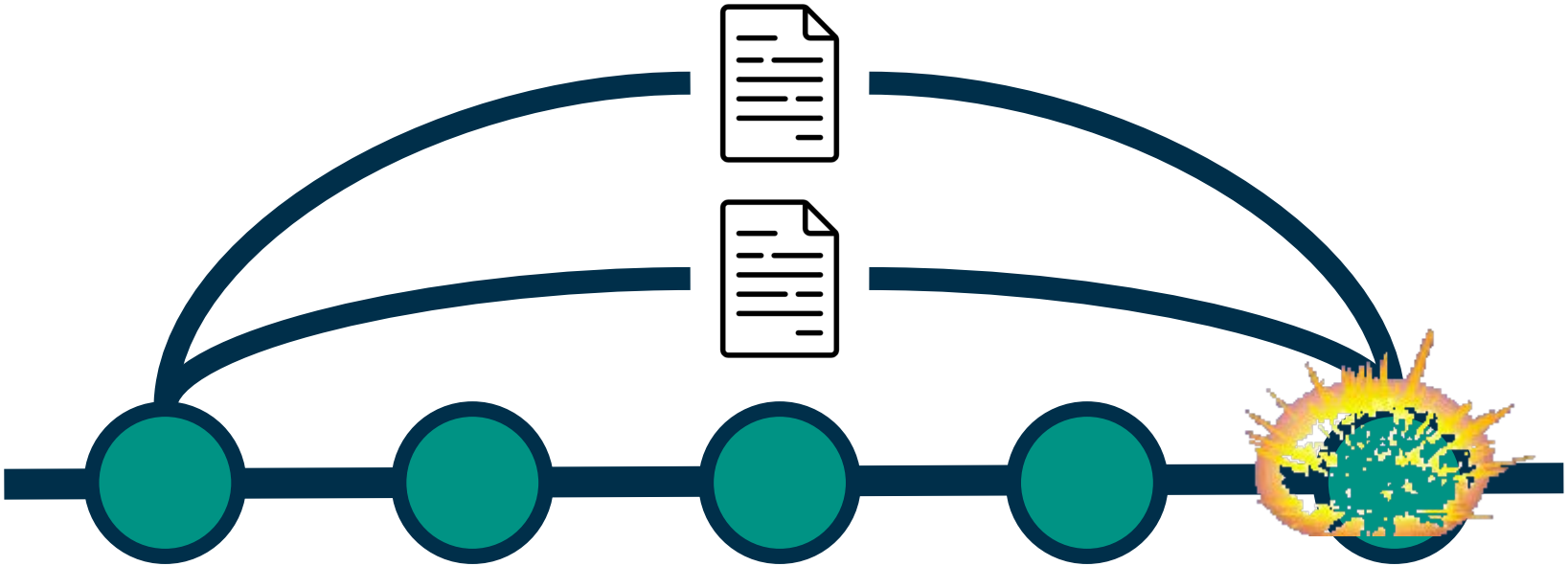


# NEVER CHANGE A SHARED BRANCH'S HISTORY

**You WILL regret it. Force push responsibly.**



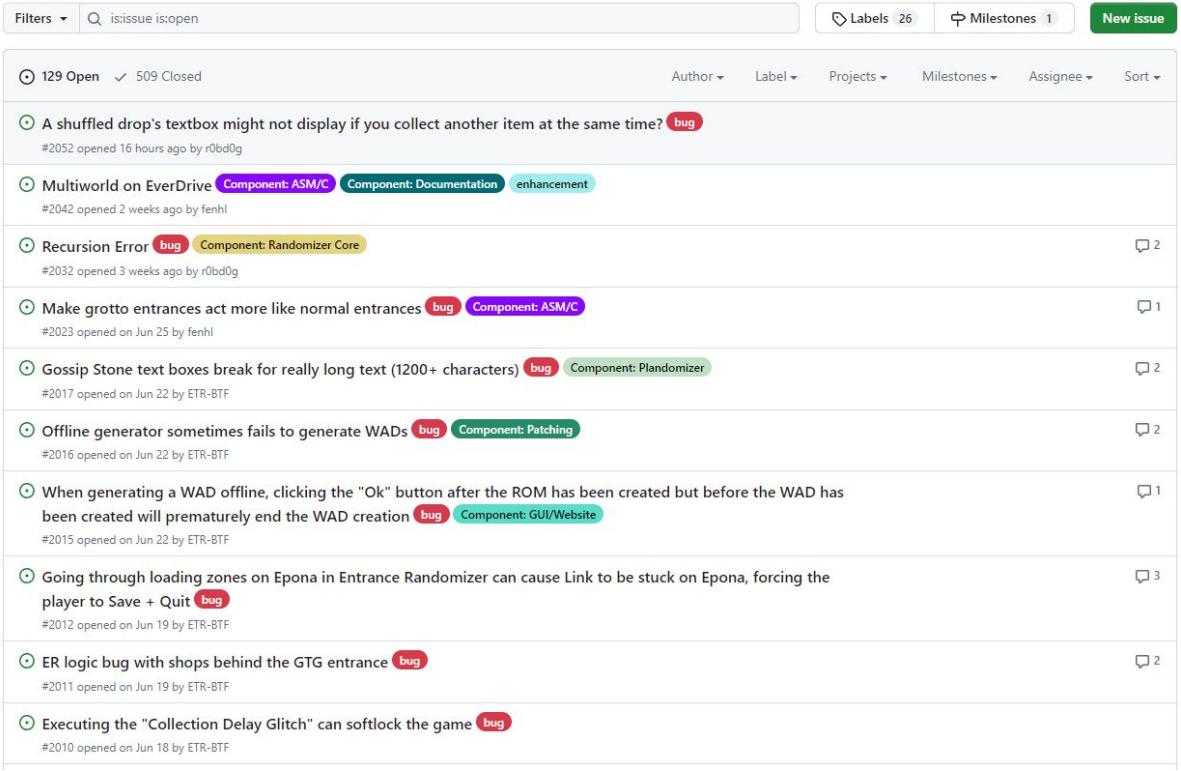
# Managing conflicts



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

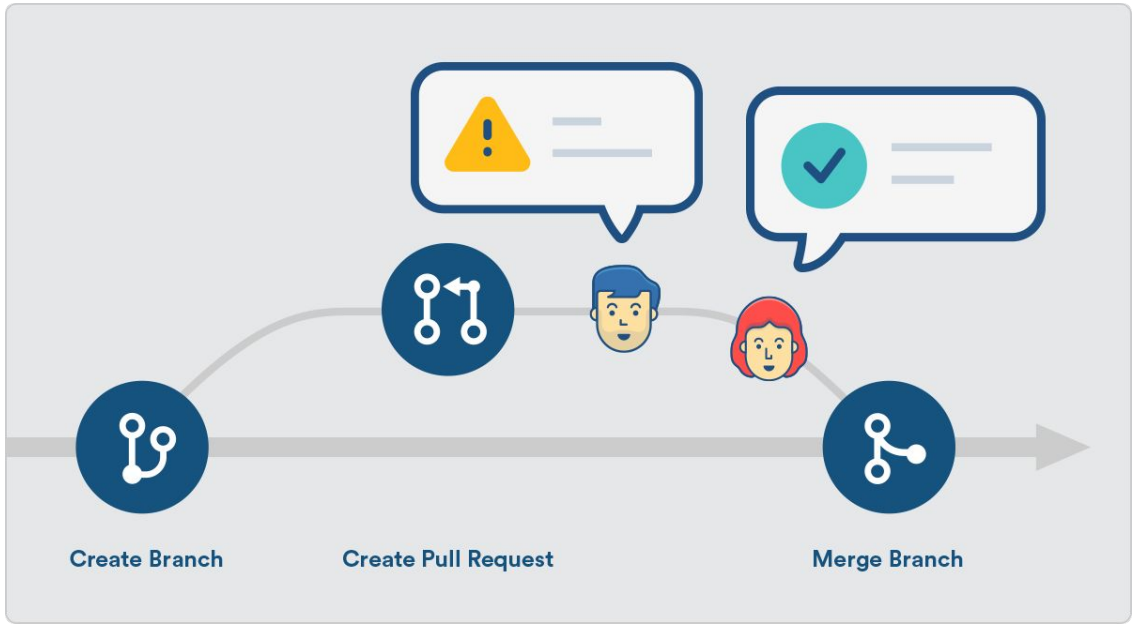
# Using git collaboratively

# Issues




**Issues** are akin to **tasks**.

# Pull requests

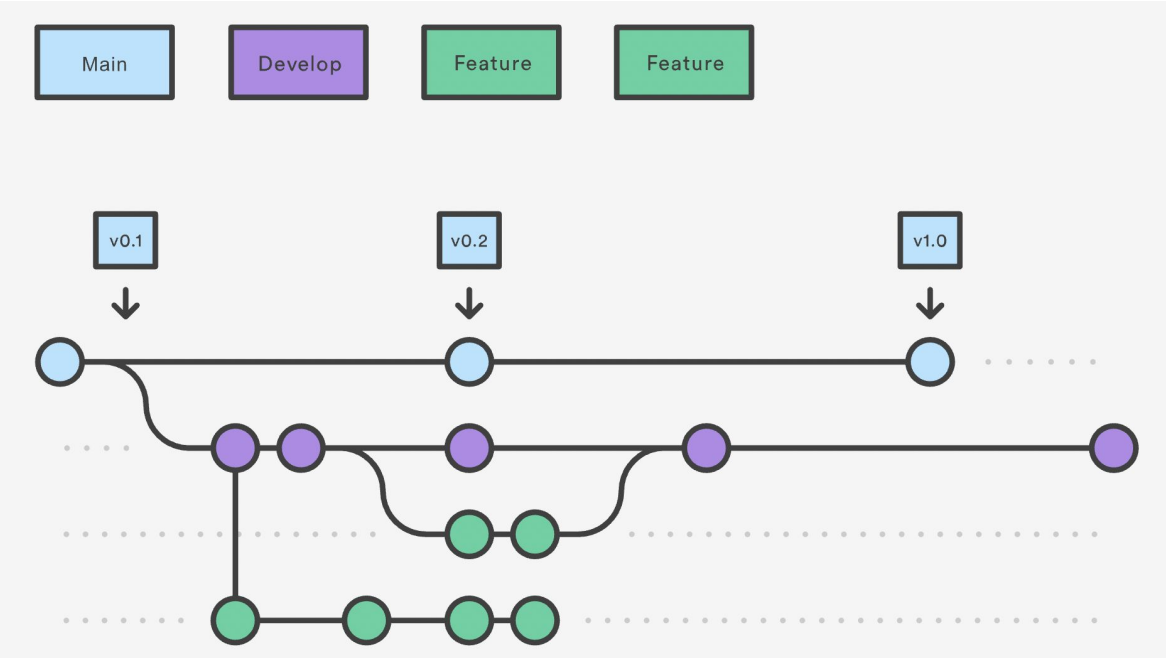



Pull requests are a **communication tool** between developers.

They allow for:

- Better code quality
- Knowledge sharing
- Communication

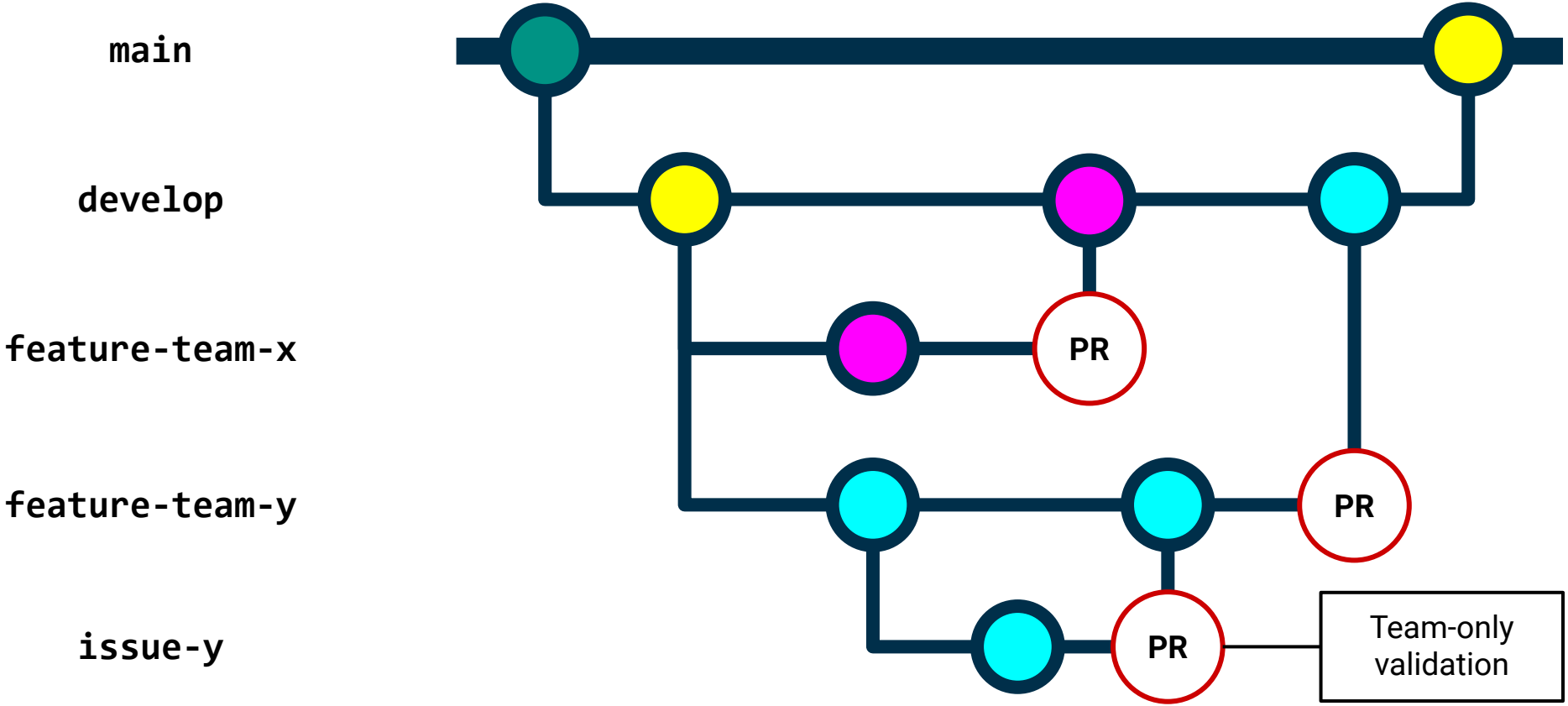
# Defining a git flow




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

# Practical work

Let’s build a mock app with Git!




The mock project’s git flow

# How we will use GitHub as a class

`practicals-2023-2024`

**`main`**

**`session_name-name1-name2`**


1. **Create a branch** from `main` with name `session-x-name1-name2` (e.g. `01_understanding_data-salem-salem`)
2. **Create a folder** with name `name1-name2` within the relevant folder (e.g. `01_understanding_data/salem-salem`)
3. Upload your work within the folder you created
4. Create a pull request for the practical

**Respecting these conventions is part of the evaluation!**

# Don’t forget to upload your work!

# Debrief

# Debrief

**What did we learn today?**

**What could we have done better?**

**What are we doing next time?**