

# Git for Version Control

*Instructor*

**Battista Biggio**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence

University of Cagliari, Italy

# About Git

- **Created by Linus Torvalds, creator of Linux, in 2005**
  - Came out of Linux development community
  - Designed to do version control on Linux kernel
- **Goals of Git**
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distribute
  - Able to handle large projects efficiently



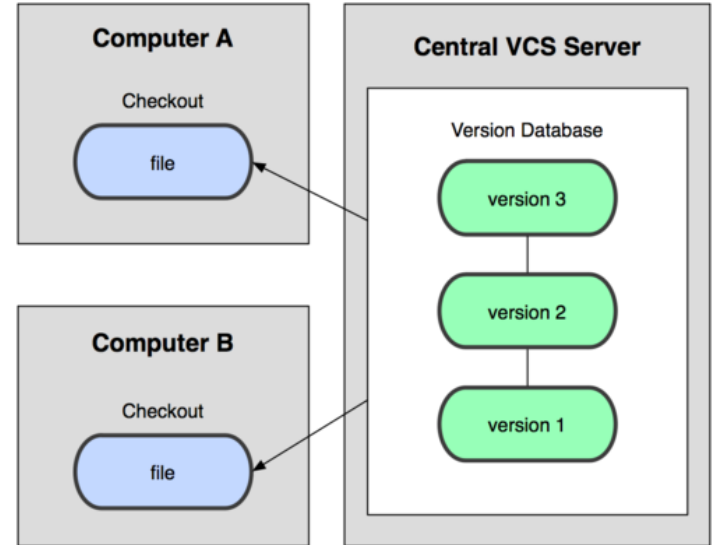
*(A "git" is a cranky old man. Linus meant himself.)*

# Installing / Learning Git

- Git website: <http://git-scm.com/>
  - Free on-line book: <http://git-scm.com/book>
  - Reference page for Git: <http://gitref.org/index.html>
  - Git tutorial: <http://schacon.github.com/git/gittutorial.html>
  - Git for Computer Scientists: <http://eagain.net/articles/git-for-computer-scientists/>
- At command line: (where verb = config, add, commit, etc.)
  - `git help verb`

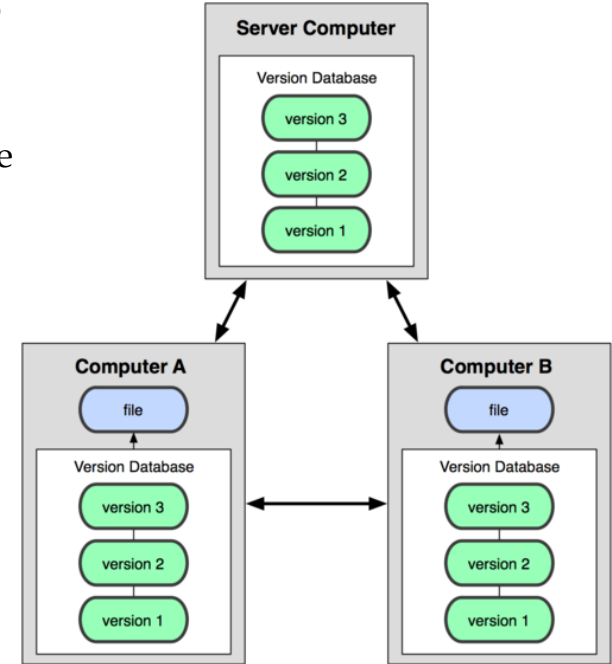
# Centralized VCS

- In Subversion (SVN), ... a central server repository holds the "official copy" of the code
  - the server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned
- When you're done, you "check in" back to the server
  - your "check in" increments the repo's version



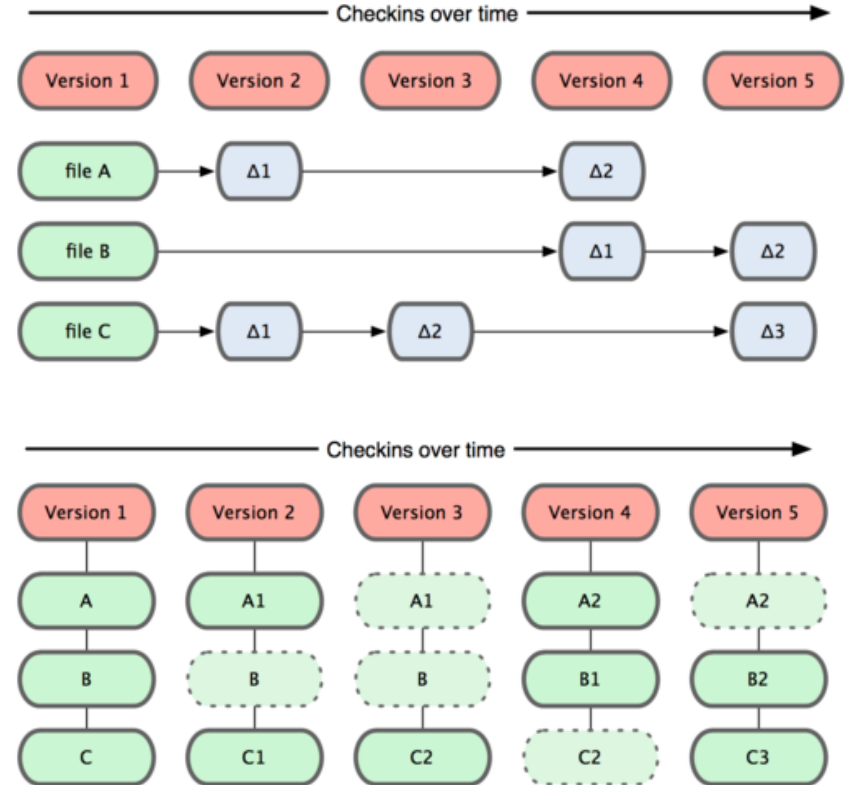
# Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout" from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote
  - yours is "just as good" as theirs
- Many operations are local
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history
- When you're ready, you can "push" changes back to server



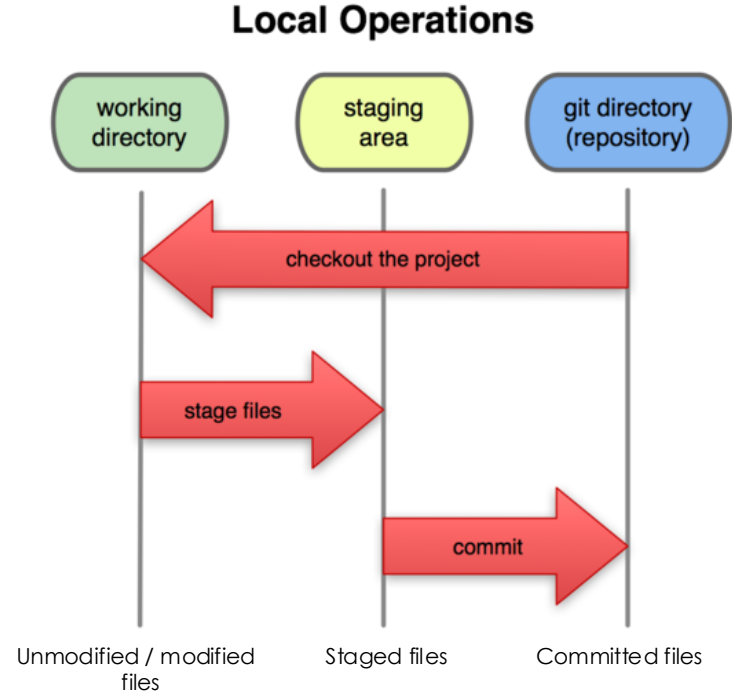
# Git Snapshots

- Centralized VCS like Subversion track version data on each individual file.
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.



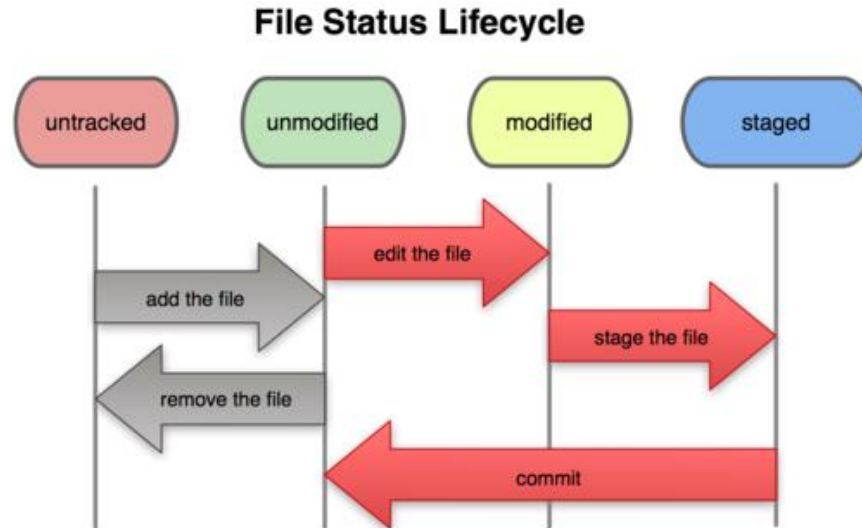
# Local Git Files

- In your local copy on git, files can be:
  - In your local repo (*committed*)
  - Checked out and modified, but not yet committed (*working copy*)
- Or, in-between, in a "**staging**" area
  - Staged files are ready to be committed
  - A commit saves a snapshot of all staged state



# Basic Git Workflow

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.





# Git Commit Checksums

- In Subversion each modification to the central repo increments the version # of the overall repo.
  - In Git, each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server.
  - So Git generates a unique **SHA-1 hash** (40 character string of hex digits) for every commit.
  - Refers to commits by this ID rather than a version number.
- Often we only see the first 7 characters:
  - 1677b2d Edited first line of readme
  - 258efa7 Added line to readme
  - 0e52da7 Initial commit

# Initial Git Configurations

- Set the name and email for Git to use when you commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - You can call `git config -list` to verify these are set.
- Set the editor that is used for writing commit messages:
  - `git config --global core.editor nano` (it is vim by default)

# Creating a Git Repository

*Two common scenarios: (only do one of these)*

- To create a new **local Git repo** in your current directory:
  - `git init`
    - This will create a `.git` directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - `git add filename`
  - `git commit -m "commit message"`
- To **clone a remote repo** to your current directory:
  - `git clone url localDirectoryName`
  - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual local repo)

# Git Commands

command	description
<code>git clone <i>url</i> [<i>dir</i>]</code>	copy a Git repository so you can add to it
<code>git add <i>file</i></code>	adds file contents to the staging area
<code>git commit</code>	records a snapshot of the staging area
<code>git status</code>	view the status of your files in the working directory and staging area
<code>git diff</code>	shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	get help info about a particular command
<code>git pull</code>	fetch from a remote repo and try to merge into the current branch
<code>git push</code>	push your new branches and data to a remote repository
others: <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>	

# Add and Commit a File

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
  - Takes a snapshot of these files, adds them to the staging area.
  - In older VCS, "add" means "start tracking this file."  
In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
  - All these commands are acting on your local version of repo.

# Viewing / Undoing Changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)  
1677b2d Edited first line of readme  
258efa7 Added line to readme  
0e52da7 Initial commit
  - `git log -5` (to show only the 5 most recent updates), etc.

# An Example Workflow

```
[rea@attu1 superstar]$ emacs rea.txt
[rea@attu1 superstar]$ git status
no changes added to commit (use "git add" and/or "git commit -a")
[rea@attu1 superstar]$ git status -s M rea.txt
[rea@attu1 superstar]$ git diff diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git add rea.txt
[rea@attu1 superstar]$ git status
# modified: rea.txt
[rea@attu1 superstar]$ git diff --cached
diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git commit -m "Created new text file"
```

# Branching and Merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - `git branch name`
- To list all local branches: (\* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout branchname`
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge branchname`



# Merge Conflicts

The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:

```
<<<<<< HEAD:index.html  
<div id="footer">todo: message here</div>  
=====
```

} branch 1's version

```
<div id="footer">  
thanks for visiting our site </div>  
>>>>>> SpecialBranch:index.html
```

} branch 2's version

Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

# Interaction with Remote Repository

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)
- To **fetch** the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`
- To put your changes from your local repo in the remote repo:
  - `git push origin master`

# GitHub/GitLab

- GitHub and GitLab are websites for online storage of Git repositories.
  - You can create a **remote repo** there and push code to it.
  - Many open source projects use it, such as the Linux kernel.
  - You can get free space for open source projects, or you can pay for private projects.
  - Free private repos for educational use
- Question: Do I always have to use GitHub/GitLab to use Git?
  - Answer: No! You can use Git locally for your own purposes.
  - Or you or someone else could set up a server to share files.
  - Or you could share a repo with users on the same file system, as long everyone has the needed file permissions).



# Git Practical Session / Workflow

*Instructor*

**Battista Biggio**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence

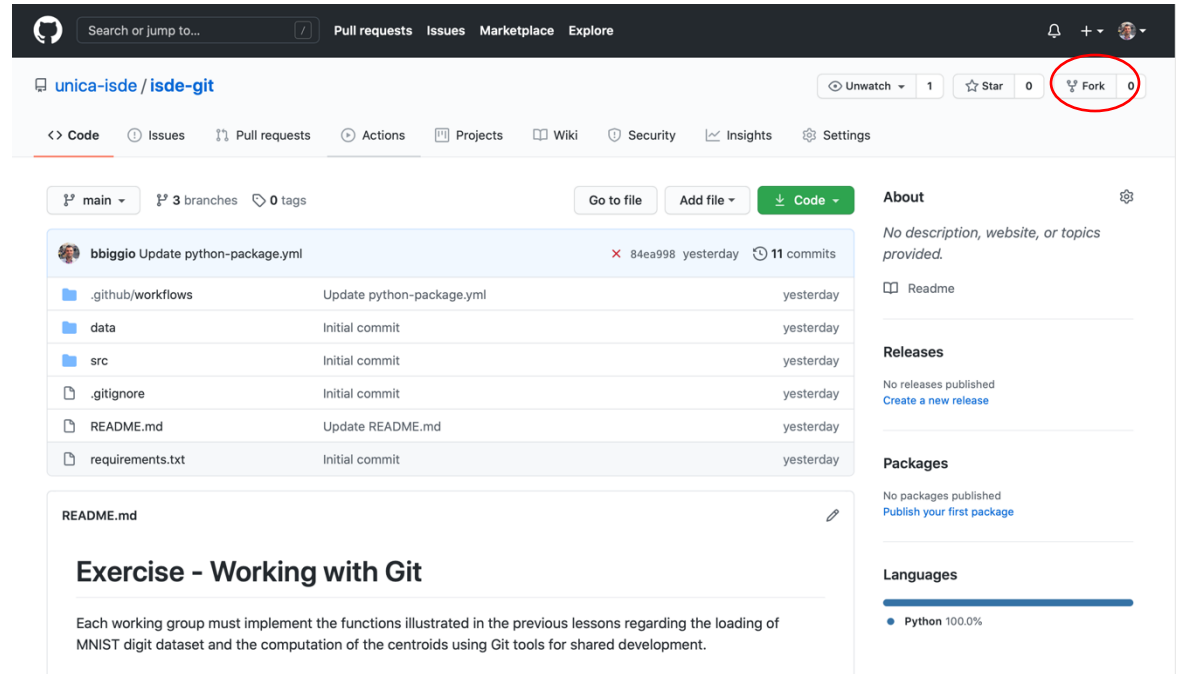
University of Cagliari, Italy

# Git Workflow

- We will solve the Git exercise proposed in <https://github.com/unica-isde/isde-git>
- We will use a simplified Git workflow and we will see its integration with PyCharm
- The basic workflow for **Git** is as follows
  1. *Maintainer:* Fork the project (create a new master project)
  2. *Maintainer:* Activate Issues and Actions
  3. *Maintainer:* Create an issue in Git
    - Each issue has a separate identifier (e.g., #1)
    - Assign the issue to the developer (assignee) – you need to add collaborators from the project settings
  4. *Assignee:* Create a branch associated to the issue to address the required changes
  5. *Assignee:* Initial Implementation / testing, commit and push
  6. *Assignee:* Create a merge/pull request associated to the aforementioned issue/branch
  7. *Assignee:* Refine implementation / testing until completed
  8. *Maintainer:* Merge branch to master and close the corresponding issue and merge/pull request
  9. *Maintainer:* Delete the working branch

# Fork the Project

- Go to <https://github.com/unica-isde/isde-git>
- Click the *Fork* button
- This will create a copy of the project into your user home in GitHub
- Go to your forked project and create a new issue (next slide)



The screenshot shows the GitHub repository page for `unica-isde/isde-git`. The repository has 1 branch and 0 tags. The `Fork` button is circled in red. The repository contains the following files and folders:

File/Folder	Commit Message	Commit Date
<code>.github/workflows</code>	Update python-package.yml	yesterday
<code>data</code>	Initial commit	yesterday
<code>src</code>	Initial commit	yesterday
<code>.gitignore</code>	Initial commit	yesterday
<code>README.md</code>	Update README.md	yesterday
<code>requirements.txt</code>	Initial commit	yesterday

The `README.md` file contains the following text:

## Exercise - Working with Git

Each working group must implement the functions illustrated in the previous lessons regarding the loading of MNIST digit dataset and the computation of the centroids using Git tools for shared development.

# GitLab WorkFlow

1. Fork Project
2. Open Issue
3. Create Branch and Merge Request
4. Work & Discuss
5. Merge

The screenshot displays the GitLab web interface for a project named 'SecML'. The main content area shows an issue titled 'Invalid format specifier when normalize=True in plot\_confusion\_matrix', reported by 'SecML-Bot'. The issue description states: 'When `normalize=True` in `plot_confusion_matrix`, the function returns `ValueError: Invalid format specifier`, as the format specifier is wrongly set to `%2f`. This is the old print format, and should be abandoned. It should be set to `.2f` instead.' The issue is marked as 'Open' and was opened 1 month ago by 'SecML-Bot'.

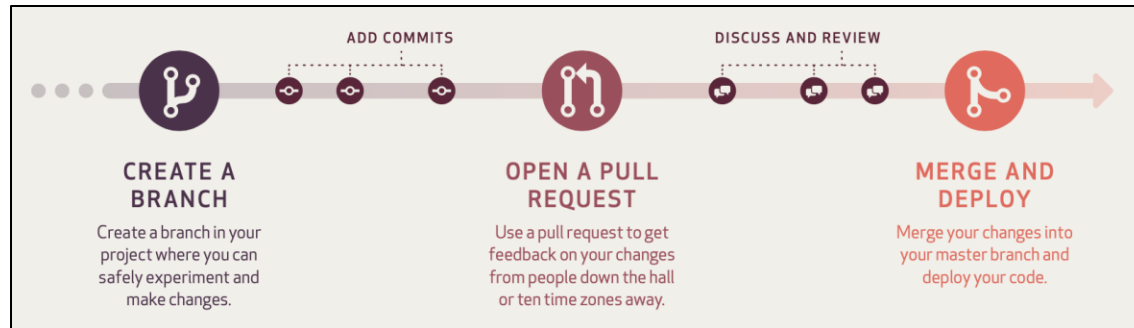
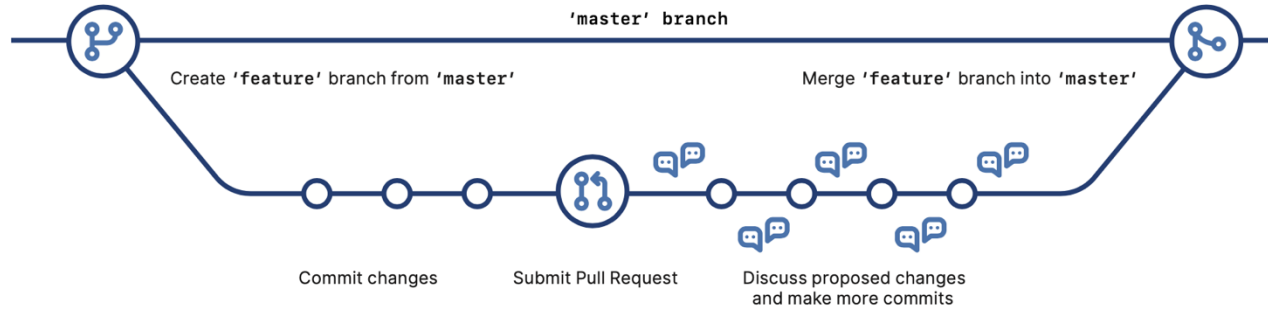
On the right side of the interface, there is a sidebar with various sections: 'To Do', 'Assignee' (Marco Melis), 'Epic' (This feature is locked), 'Milestone' (None), 'Time tracking' (No estimate or time spent), 'Due date' (None), 'Labels' (bug, confirmed, package: figure, v0.14), 'Weight' (None), 'Health status' (None), 'Confidentiality' (Not confidential), and 'Lock issue' (Unlocked).

A red box highlights the 'Create merge request' button and the dropdown menu that appears. The dropdown menu shows the following options:

- ✓ Create merge request and branch
- Create branch
- Branch name: 858-invalid-format-specifier-when
- Source (branch or tag): master
- Create merge request

# GitHub Workflow

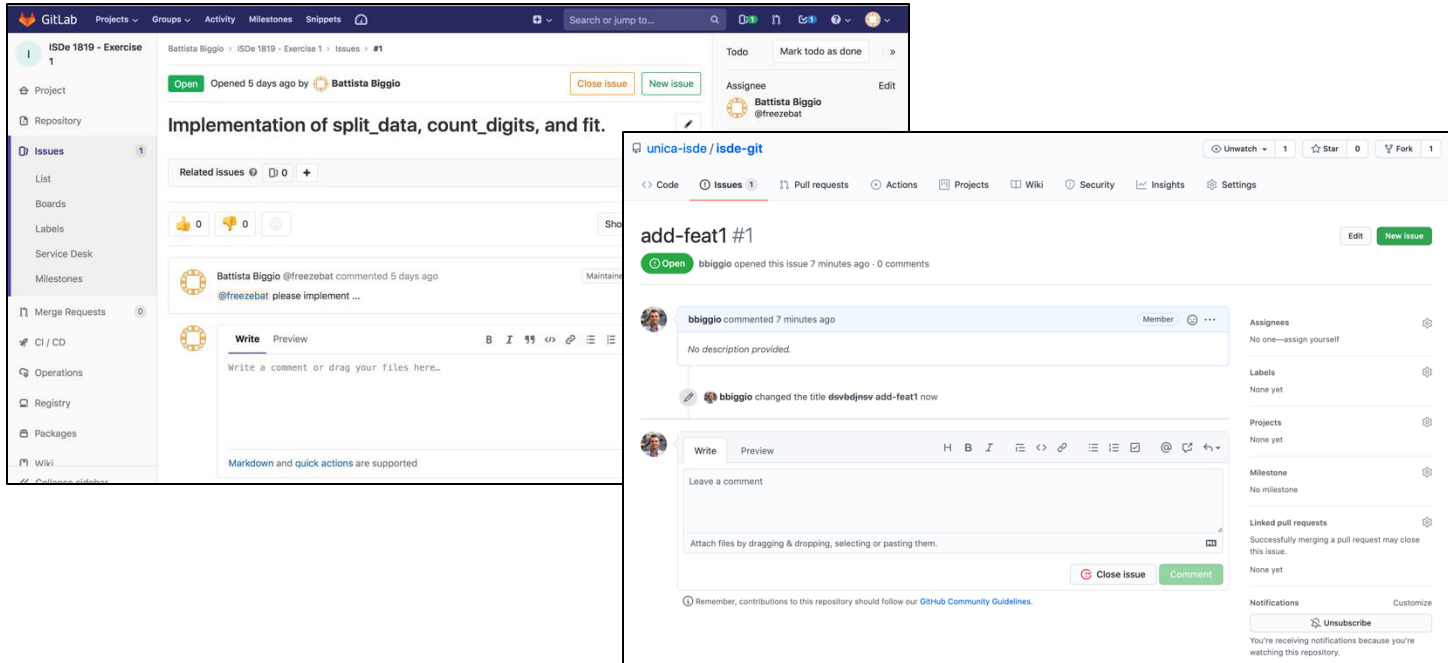
- **Main difference w/ GitLab:** no direct/one-to-one connection between issues and branches





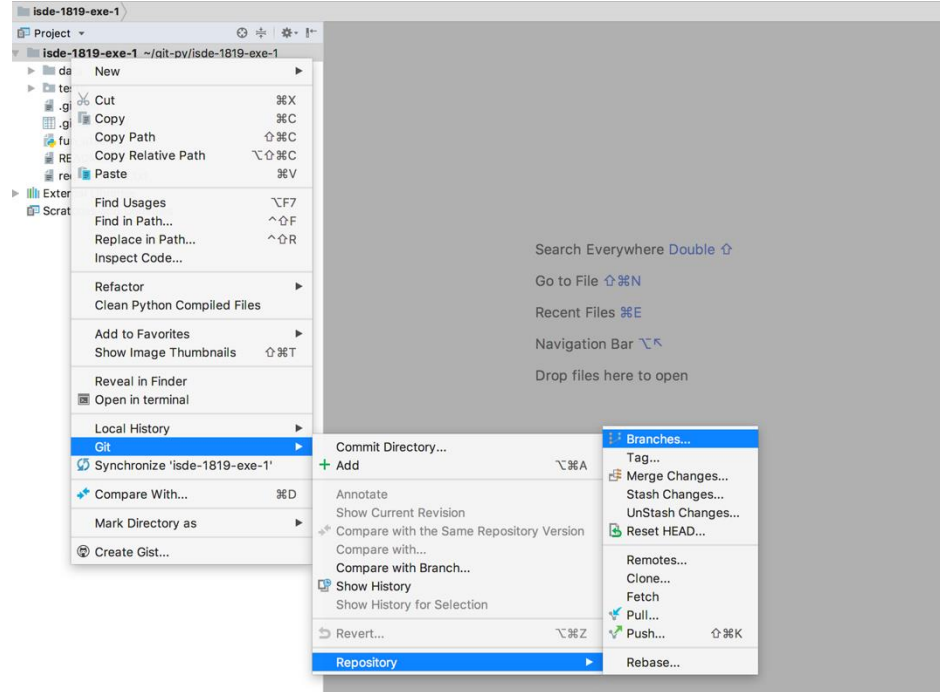
# Create a New Issue

- Create a new issue and a corresponding branch; recall however that the exercise required creating a different issue for each function to be implemented (3 in total)



# Create the Corresponding Branch

- Open Pycharm, and create a new project by checking out your forked project from Git
  - Use *checkout from version control*, select *Git*, and copy-paste the URL from the project page
  - The URL is something like:  
<https://github.com/your-username/isde-git.git>
- Create a new branch from the Git menu (or from the Git webpage)
  - Name it using the convention *issue\_id-branch\_name*
  - e.g., *1-split\_and\_fit*
- Push the branch to the remote repository



# Implement, Test, Commit & Push

- Implement the required functions
- Check if they pass the tests, otherwise keep working on them to fix the problems
- Commit (and keep coding if necessary)
  - Recall that *commit* only affects your local repository
- Push when finished
  - Pushes the changes to the remote branch

# Create the Merge/Pull Request

Active branches

1-split\_and\_fit  
4c432370 · New branch to address issue #1 · 41 seconds ago

01 Merge request Compare

master default protected  
7ae9a64f · Update README.md · 1 month ago

## New Merge Request

From 1-split\_and\_fit into master [Change branches](#)

Title

WIP: New branch to address issue #1

Remove the **WIP:** prefix from the title to allow this **Work In Progress** merge request to be merged when it's ready.  
Add [description templates](#) to help your contributors communicate effectively!

Description

Write Preview

B I

Closes #1

[Markdown](#) and [quick actions](#) are supported [Attach a file](#)

# Merging the Request and Closing the Issue

- Remove the «WIP» prefix from the merge request (GitLab) or comment that you finished on the pull request (GitHub)
  - This means: we are now done with this issue and ready to merge
- Merge the branch *1-split\_and\_fit* to the master/main
- Close the corresponding issue/merge request

**That's it! This was a simple exercise to demonstrate how to use Git and a very basic Git workflow. You should follow this workflow in your home assignments, also using the Git Board to prioritize and track issues.**

**Homework:** open another issue to document the code using docstrings, and then merge that to the master. Generate the docs locally on your machine (no need to upload them to the repository)

# Recap: GitHub and Gitlab Cheat Sheets

- GitHub Cheat Sheet (basic commands, workflow)
  - <https://education.github.com/git-cheat-sheet-education.pdf>
- GitLab Cheat Sheet (basic commands, workflow)
  - <https://about.gitlab.com/images/press/git-cheat-sheet.pdf>