

COMP310/ECSE427 Lab1

Git

Jason Zixu Zhou

What is Version Control?

- **Definition:**

- A system that records changes to a file or files over time so you can recall specific versions later.

- **Use Cases:**

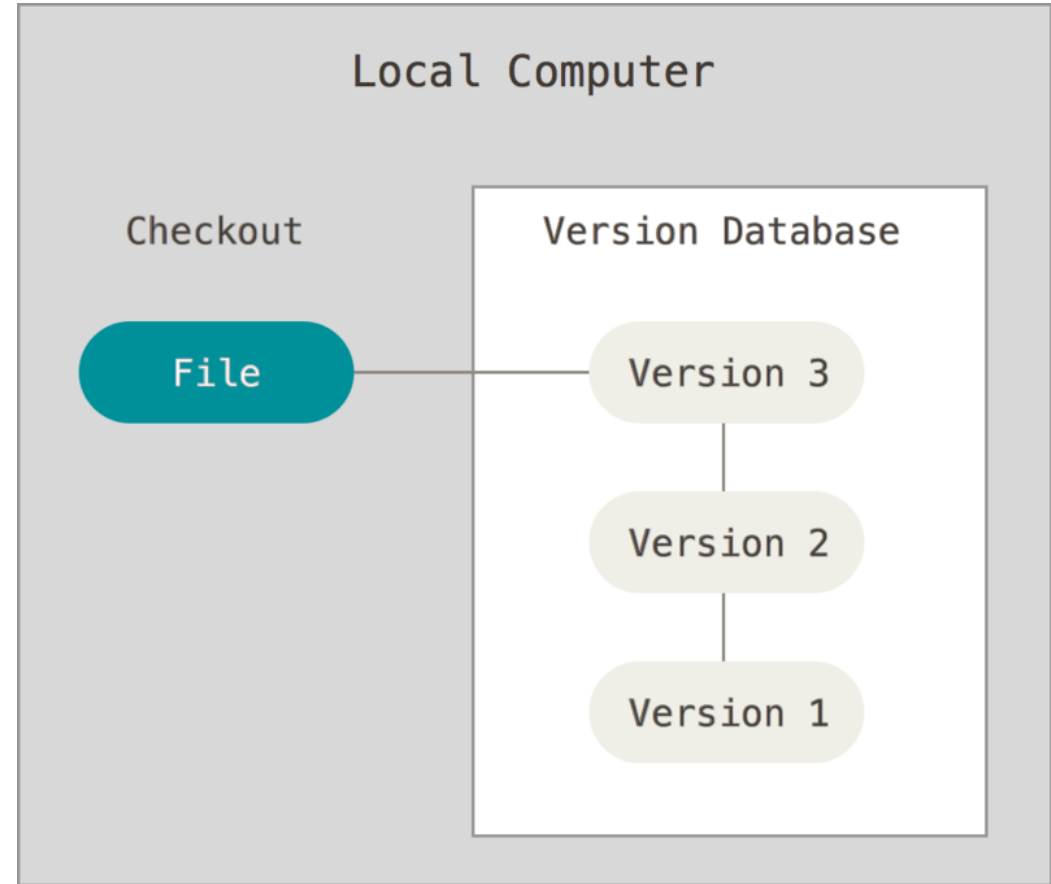
- Ideal for software source code and nearly any file type.

- **Benefits:**

- Revert files or entire project to a previous state.
- Track changes and identify who made specific changes.
- Recover lost files with minimal overhead.

Local Version Control

- **Early Methods:**
 - Copying files into time-stamped directories.
- **Problems:**
 - Error-prone and easy to lose track.
- **Local VCS:**
 - Uses a simple database to manage file revisions.
- **Example:**
 - RCS (Revision Control System).



Centralized Version Control

- **Need:**

- Collaboration among developers on different systems.

- **How It Works:**

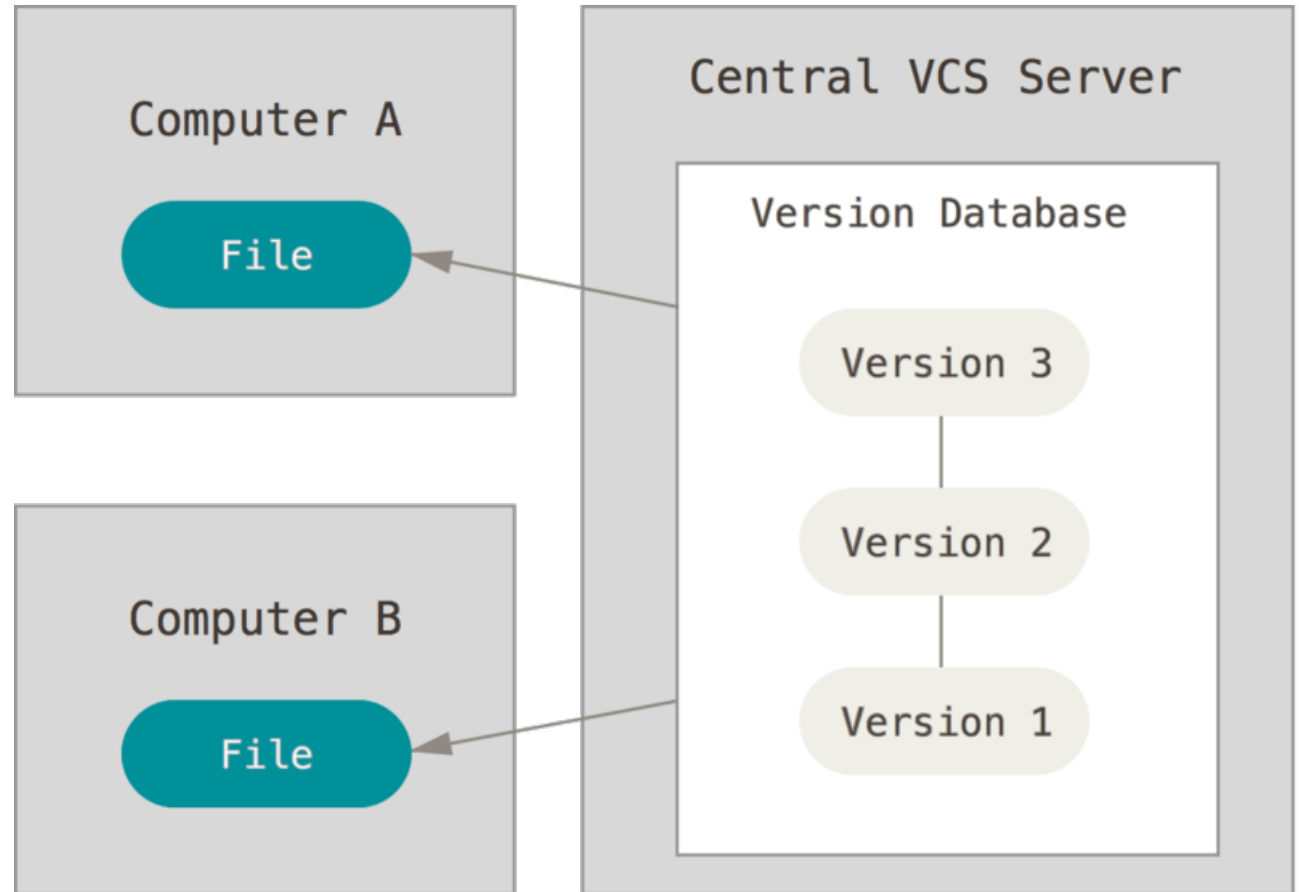
- A central server stores all versioned files, clients check out files from this server.

- **Advantages:**

- Easier administration.
- Enhanced visibility into project activities.

- **Risks:**

- Single point of failure; total loss of history if the server fails without backups.



Distributed Version Control Systems

- **Introduction:**

- Overcomes limitations of centralized systems.

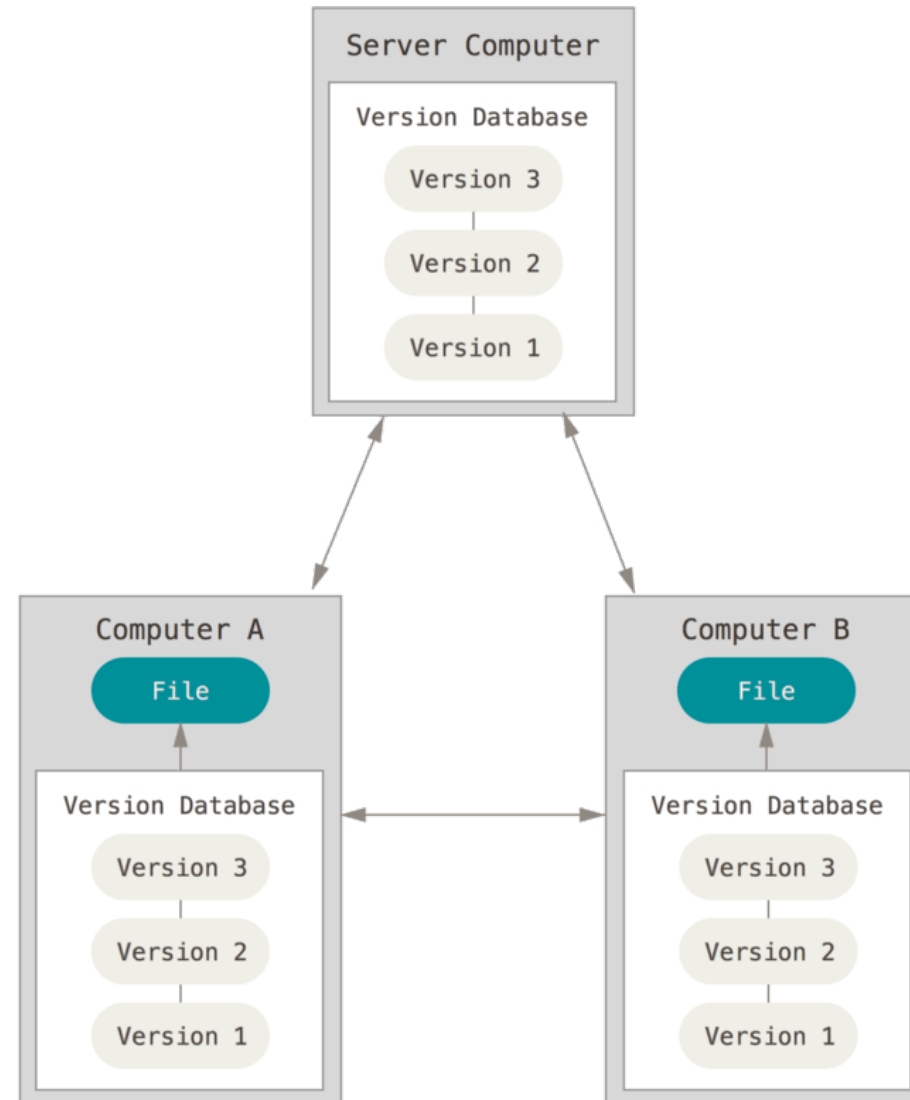
- **Operation:**

- Clients mirror the repository, including its full history.

- **Benefits:**

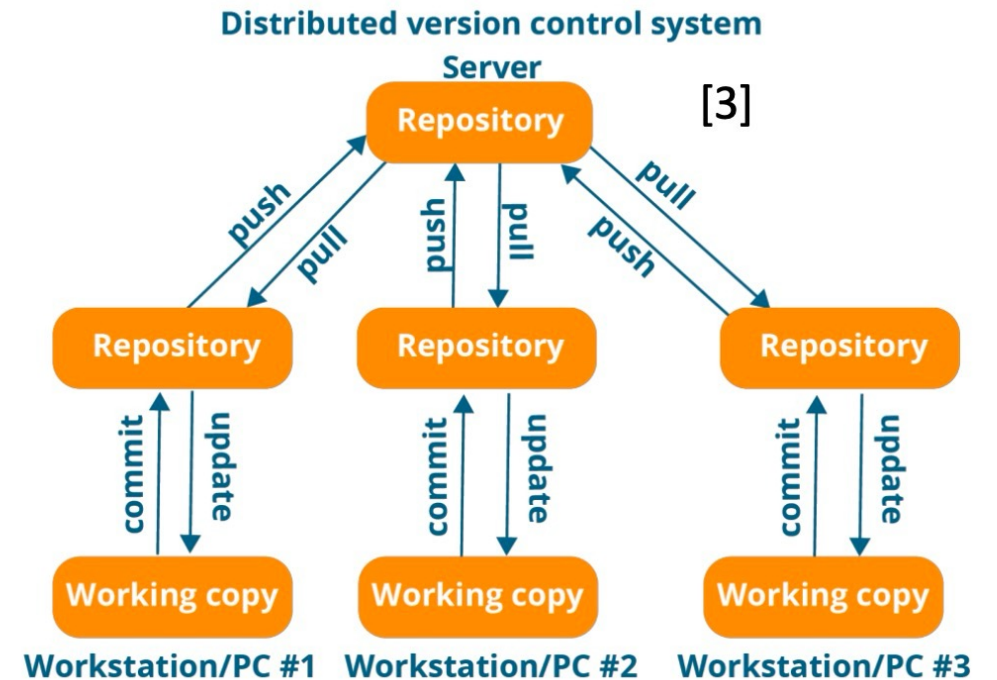
- Each clone is a full backup.
- Robust against server failures.
- Supports multiple remote repositories and diverse collaborative workflows.

<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



Git

- Distributed Version Control system
- Created in a month in 2005 by Linus Torvalds of Linux fame
- Each developer has an independent copy of the whole history
- If server dies, there are many backups of the history



Copy from the slides of Sebastian Rolon

Git Terminology

- **Repository:** A "folder" containing all the code files and the entire history of the code
- **Remote repository:** Your repository, but stored in a server where it is always accessible, safe, and your teammates can access it too
- **Staging area:** After you modify the code, the area in your computer where Git keeps track of which changes you want to save
- **Commit:** "Save" or register your changes into the history of the repository
- **Push:** Upload the saves to the remote repository
- **Pull:** Get the latest saves that people have uploaded to the remote repo
- **Clone:** Download the repository for the first time

Add Commit Push



Branch

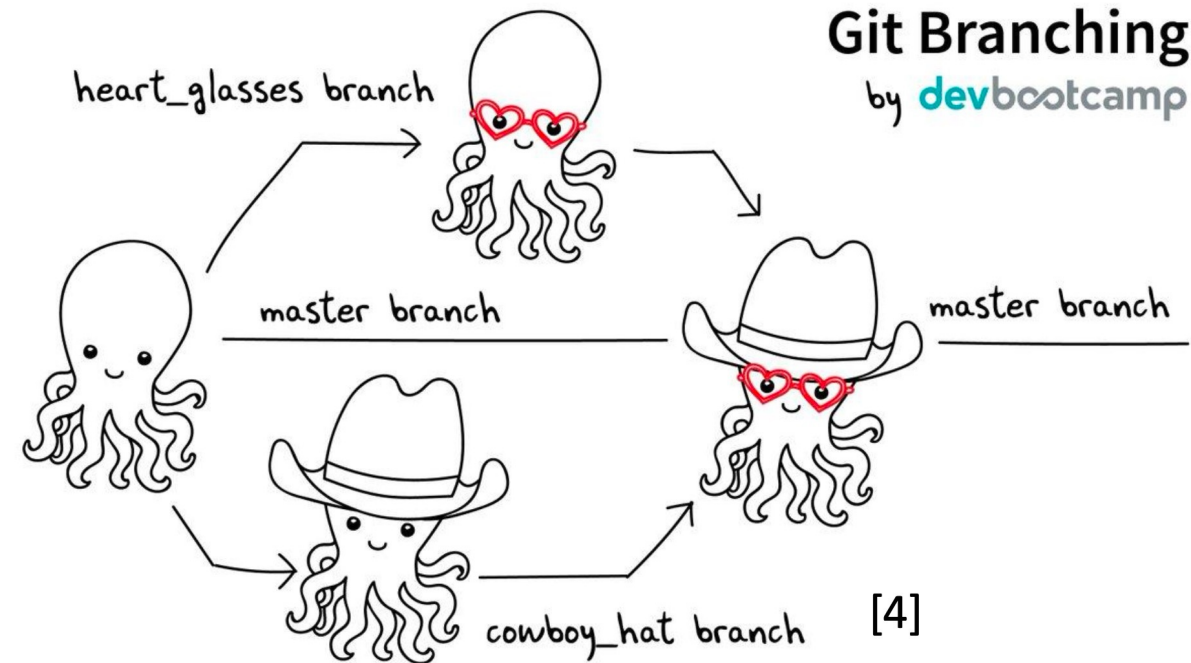
Git allows you to have “parallel universe” versions of your code

They are called Branches

Branches are part of your repository

Why have this?

- Work on multiple ideas in parallel without risking the history of the code
- Work on a feature until it's ready to be used without disturbing the main code

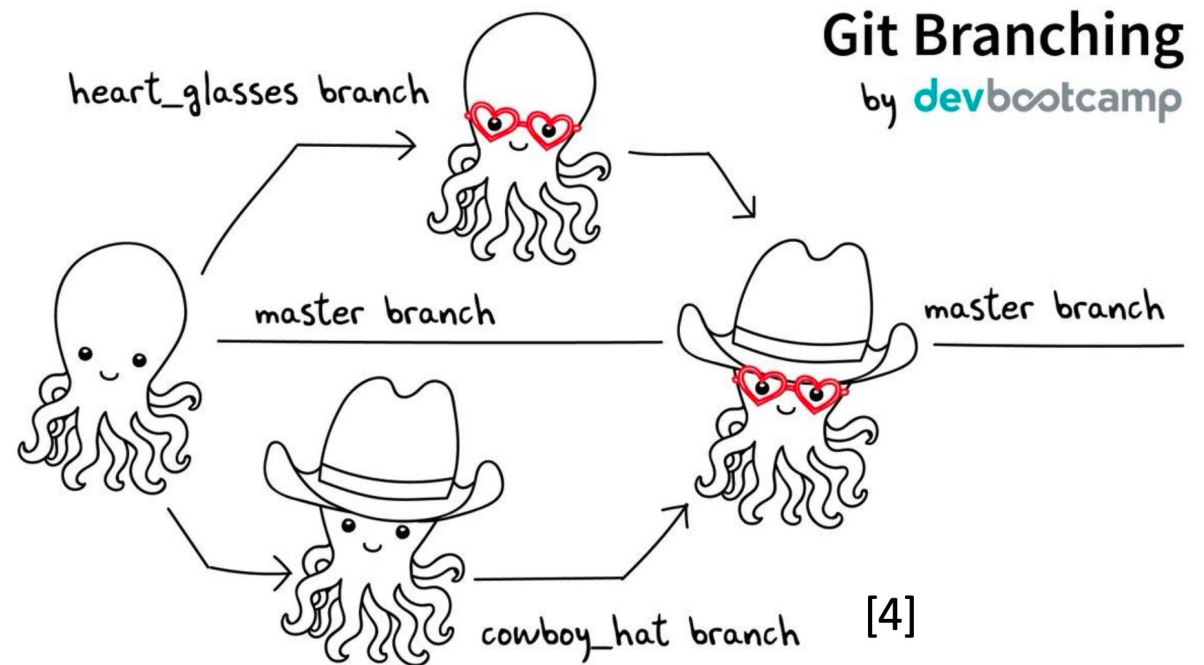


Merge

- Merging two branches is combining their changes
- One of the branches ends up with all the changes

What if two branches have different changes in the same place?

- This is called a merge conflict
- You will have to manually go through the conflicts and decide what to keep
- Merging is automatic if there are no conflicts



Fork

- **Definition:**

A fork is a personal copy of another user's repository that's stored in your account. It allows you to experiment, make changes, and propose those changes back to the original repository.

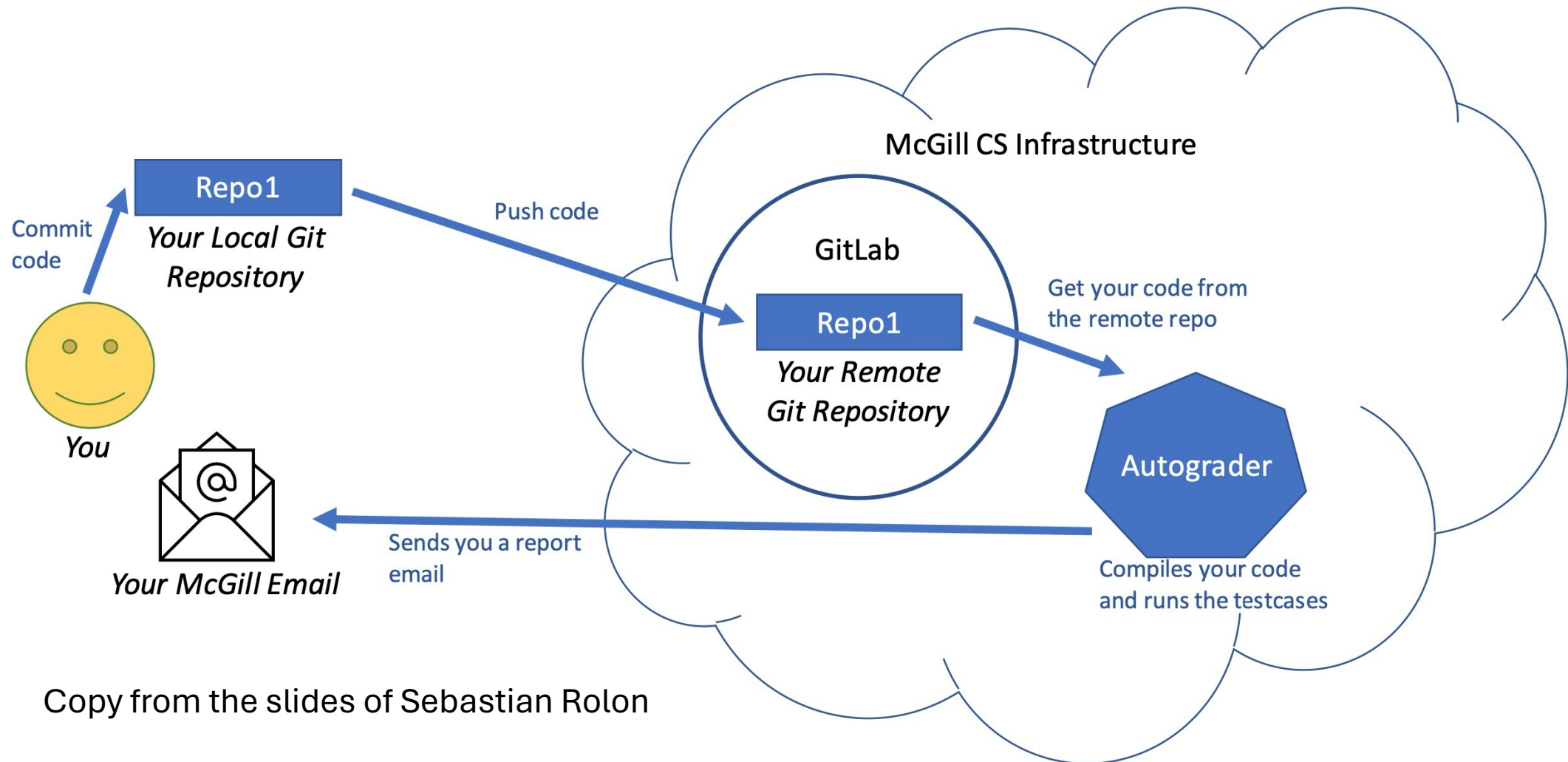
- **Purpose:**

- **Experimentation:** Test changes without affecting the original project.
- **Contribution:** Submit patches or enhancements to the project.
- **Independence:** Develop new features or take the project in a different direction without needing permissions from the original repository owners.

Workflow of Fork

- 1.Fork the Repository:** Make your own copy of a repository.
- 2.Clone the Fork:** Work locally on your machine.
- 3.Make Changes:** Update, add, delete files.
- 4.Commit Changes:** Save your work to your fork.
- 5.Push Changes:** Upload the changes to your GitHub fork.
- 6.Pull Request:** Send a request to the original owner to pull your changes(Unnecessary).

Autograder and git



Copy from the slides of Sebastian Rolon

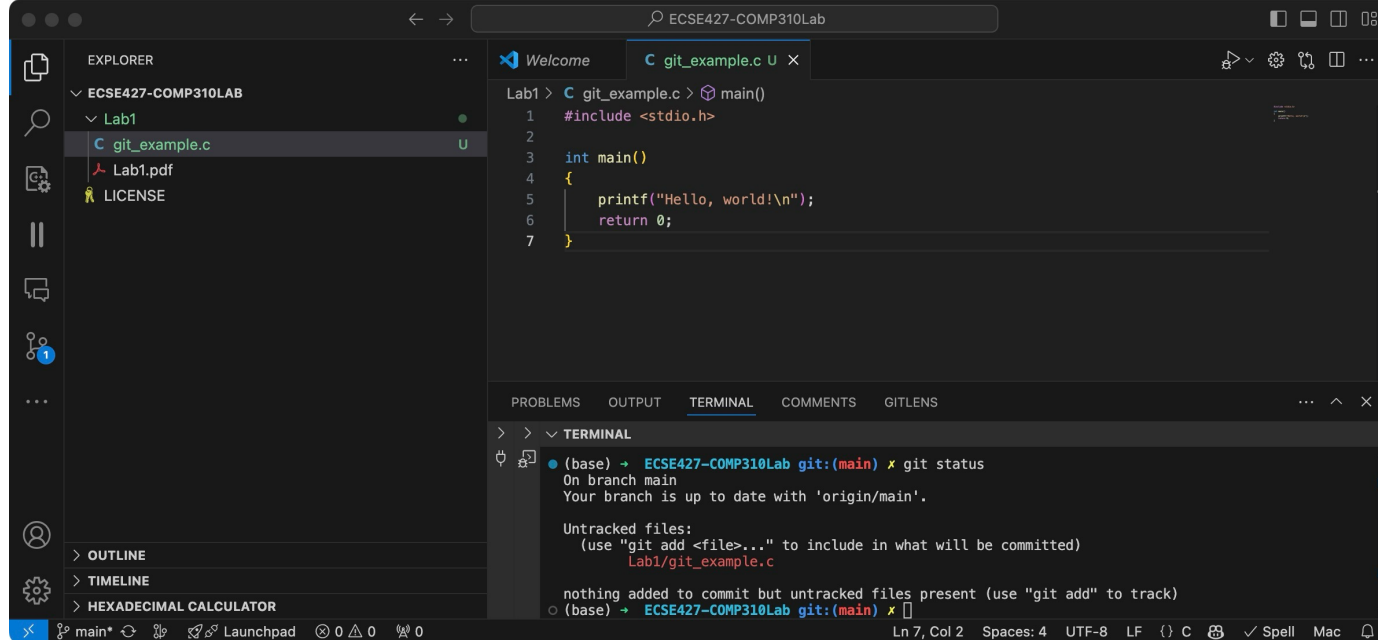
Hello World

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Example-Work in Team

- Run `git add git_example.c` to stage the file for committing.
- Run `git commit -m "Add git_example.c"` to commit the file to the repository.



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows a project named 'ECSE427-COMP310LAB' with a subdirectory 'Lab1' containing 'git_example.c' and 'Lab1.pdf'. The main editor shows the content of 'git_example.c', which includes a C program with a `main` function that prints 'Hello, world!'. The TERMINAL panel at the bottom shows the following output:

```
(base) → ECSE427-COMP310Lab git:(main) ✗ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Lab1/git_example.c

nothing added to commit but untracked files present (use "git add" to track)
(base) → ECSE427-COMP310Lab git:(main) ✗
```

```
• (base) → Lab1 git:(main) ✗ git commit -m "Add git_example.c"

[main ec1e6c0] Add git_example.c
1 file changed, 7 insertions(+)
create mode 100644 Lab1/git_example.c
• (base) → Lab1 git:(main) git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
○ (base) → Lab1 git:(main) □
```

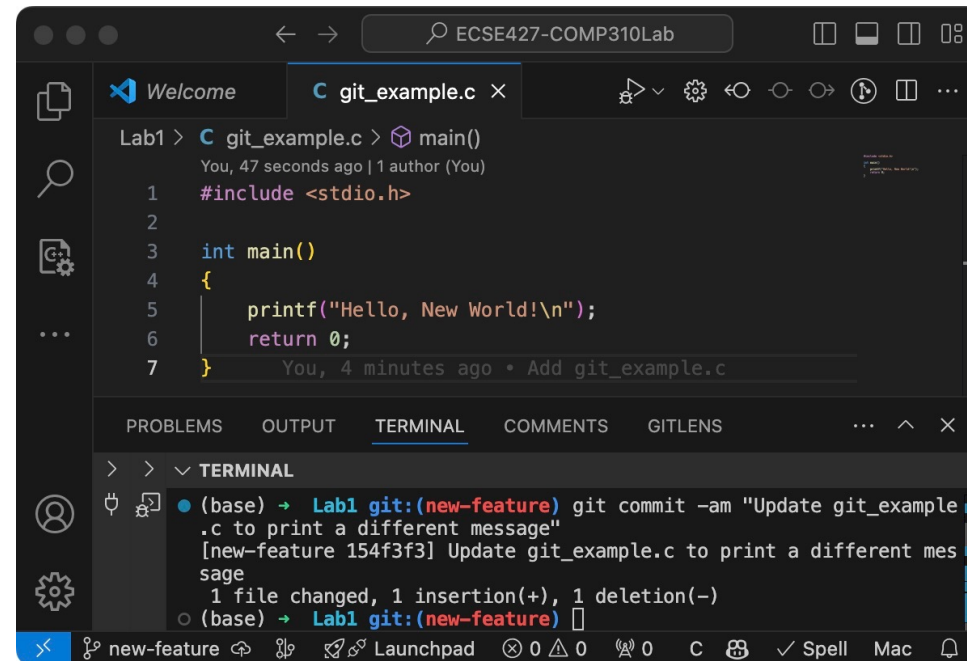
Example

- Run `git branch new-feature` to create a new branch.
- Run `git checkout new-feature` to switch to the new branch.
- Modify `git_example.c`, perhaps to print a different message.
- Commit the changes with `git commit -am "Update git_example.c to print a different message"`.
- `git push origin new-feature`

```
● (base) → Lab1 git:(main) git branch new-feature
⊗ (base) → Lab1 git:(main) git branch -all
error: did you mean '--all' (with two dashes)?
● (base) → Lab1 git:(main) git branch --all

* main
  new-feature
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
(END)

● (base) → Lab1 git:(main) git checkout new-feature
Switched to branch 'new-feature'
● (base) → Lab1 git:(new-feature) git status
On branch new-feature
nothing to commit, working tree clean
```



The screenshot shows a code editor with a dark theme. The top panel displays the `git_example.c` file with the following content:

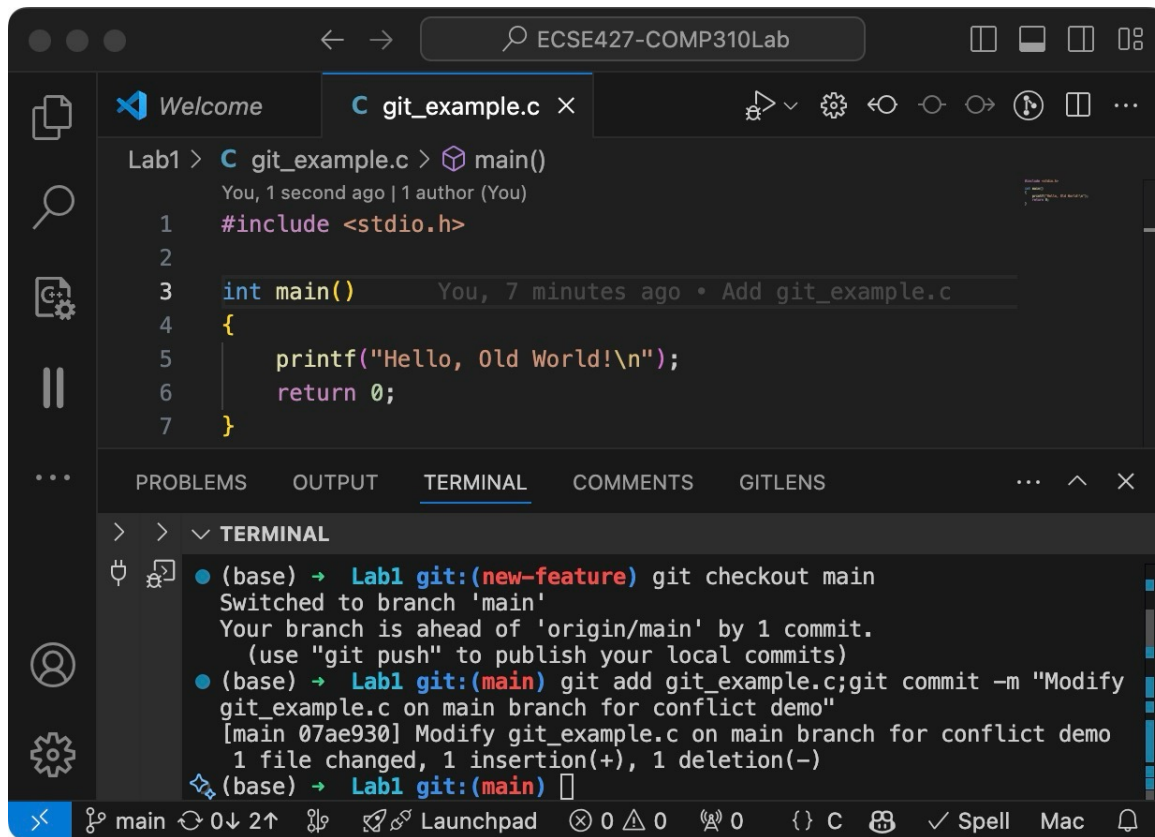
```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello, New World!\n");
6     return 0;
7 }
```

The bottom panel shows the terminal output:

```
● (base) → Lab1 git:(new-feature) git commit -am "Update git_example.c to print a different message"
[new-feature 154f3f3] Update git_example.c to print a different message
1 file changed, 1 insertion(+), 1 deletion(-)
```


Example

- Run `git checkout main` to switch back to the main branch. Introduce a change in `git_example.c` on the main branch that conflicts with your branch change. Commit this change.



The screenshot shows a code editor window with a dark theme. The top bar indicates the file path `ECSE427-COMP310Lab`. The editor has two tabs: `Welcome` and `git_example.c`. The `git_example.c` tab is active, showing the following code:

```
Lab1 > C git_example.c > main()
You, 1 second ago | 1 author (You)
1 #include <stdio.h>
2
3 int main() You, 7 minutes ago • Add git_example.c
4 {
5     printf("Hello, Old World!\n");
6     return 0;
7 }
```

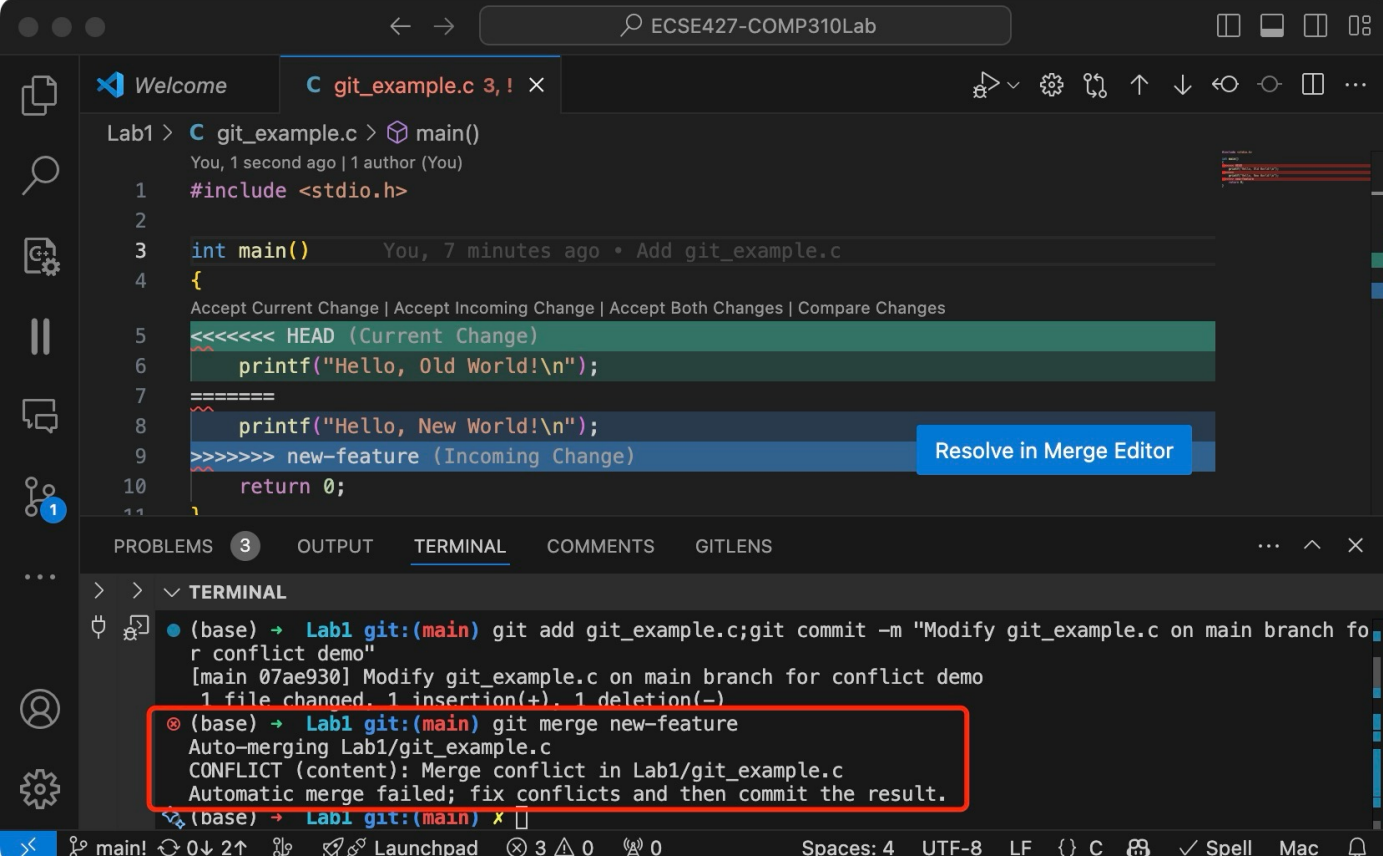
Below the editor is a terminal window with the following output:

```
> > ✓ TERMINAL
• (base) → Lab1 git:(new-feature) git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
• (base) → Lab1 git:(main) git add git_example.c;git commit -m "Modify
git_example.c on main branch for conflict demo"
[main 07ae930] Modify git_example.c on main branch for conflict demo
1 file changed, 1 insertion(+), 1 deletion(-)
• (base) → Lab1 git:(main) █
```

The bottom status bar shows the current branch as `main`, with a commit count of `0↓ 2↑`. The system tray at the bottom includes icons for Launchpad, 0 errors, 0 warnings, 0 suggestions, a C compiler icon, a spell checker icon, and the Mac logo.

Example

- Run `git merge new-feature` to merge the changes from new-feature into main.



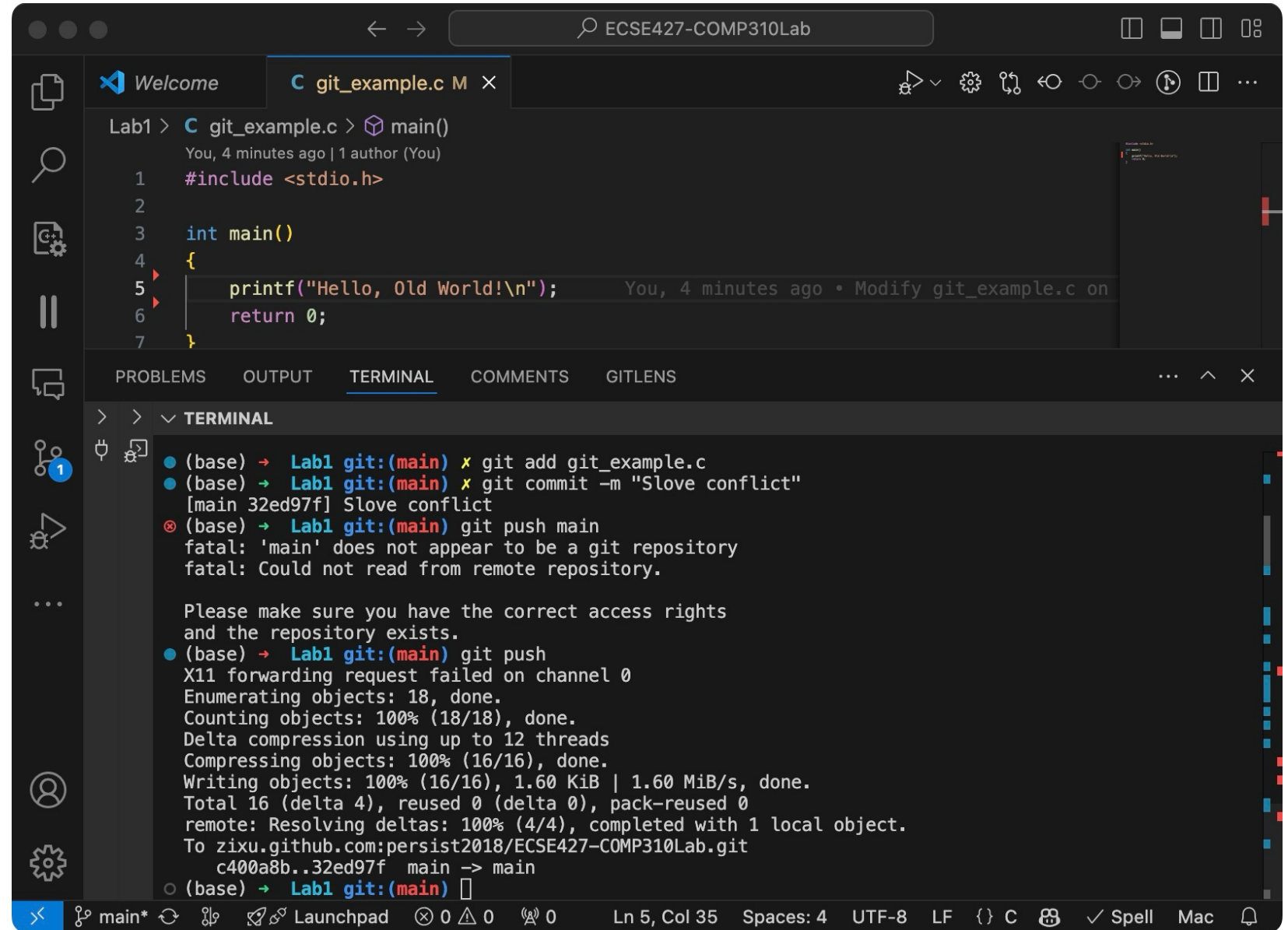
The screenshot shows a Visual Studio Code editor window with a file named `git_example.c` open. The file contains a C program with a `main` function. A merge conflict is visible, with the current change (HEAD) and the incoming change (new-feature) both present. The conflict is resolved in the Merge Editor, showing the current change's `printf("Hello, Old World!\n");` and the incoming change's `printf("Hello, New World!\n");`. A button labeled "Resolve in Merge Editor" is visible.

The terminal window at the bottom shows the following commands and output:

```
(base) → Lab1 git:(main) git add git_example.c;git commit -m "Modify git_example.c on main branch for conflict demo"
[main 07ae930] Modify git_example.c on main branch for conflict demo
1 file changed, 1 insertion(+), 1 deletion(-)
⊗ (base) → Lab1 git:(main) git merge new-feature
Auto-merging Lab1/git_example.c
CONFLICT (content): Merge conflict in Lab1/git_example.c
Automatic merge failed; fix conflicts and then commit the result.
⊗ (base) → Lab1 git:(main) x
```

Example

- Resolve any conflicts that arise, then commit the resolved version.
- git push



The screenshot shows a code editor with a file named `git_example.c` open. The code is a simple C program that prints "Hello, Old World!\n". The terminal window shows the following commands and output:

```
(base) → Lab1 git:(main) x git add git_example.c
(base) → Lab1 git:(main) x git commit -m "Slove conflict"
[main 32ed97f] Slove conflict
⊗ (base) → Lab1 git:(main) git push main
fatal: 'main' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
(base) → Lab1 git:(main) git push
X11 forwarding request failed on channel 0
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (16/16), 1.60 KiB | 1.60 MiB/s, done.
Total 16 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To zixu.github.com:persist2018/ECSE427-COMP310Lab.git
c400a8b..32ed97f  main -> main
○ (base) → Lab1 git:(main) □
```

The status bar at the bottom indicates the current branch is `main`, the editor is `Launchpad`, and the file encoding is `UTF-8`.

Comparing GitHub and GitLab

- **Introduction to GitHub:**

- Founded in 2008, GitHub is the largest host of source code in the world.
- Offers cloud-based Git repository hosting.
- Focuses on simplicity and community collaboration.

- **Introduction to GitLab:**

- Founded in 2011, GitLab is a single application for the entire software development lifecycle.
- Offers both cloud-based and self-hosted options.
- Focuses on integrated CI/CD and comprehensive DevOps solutions.

Key Features and Differences

- **User Interface and Experience:**

- GitHub: User-friendly, intuitive, preferred for project collaboration.
- GitLab: Comprehensive, feature-rich, can be overwhelming but **highly customizable**.

- **CI/CD Integration:**

- GitHub: Offers GitHub Actions for automation but relies heavily on third-party apps.
- GitLab: Built-in CI/CD features, providing a more seamless and integrated experience.

- **Deployment Options:**

- GitHub: Primarily cloud-hosted, with GitHub Enterprise for self-hosting.
- GitLab: Flexible with both cloud and self-hosted options from the start.

- **Open Source vs. Proprietary:**

- GitHub: Primarily proprietary but supports a vast array of open-source projects.
- GitLab: Core is open-source, offering greater transparency and customization.

Ref

1. Version Control with Git. The Carpentries.

<https://swcarpentry.github.io/git-novice/>

2. Subversion – Source Code Control. Doug

Harper. <http://physics.wku.edu/phys316/software/svn/>

3. What Is Git ? – Explore A Distributed Version Control Tool.
Reshma

Ahmed – Edureka. <https://www.edureka.co/blog/what-is-git/>

4. Git Intro – Branching and Merging. Code

Refinery. <https://coderefinery.github.io/git-intro/branches/>