



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

A Beginner's Workshop on GIT

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



WORKSHOP OBJECTIVES

- Understand what version control is and why it matters
- Learn the difference between Git and GitHub
- Practice the basic Git workflow locally
- Connect a local repository to GitHub
- Collaborate with a team using branches and pull requests



LEAGUE OF OUTSTANDING PROGRAMMERS

MEET THE SPEAKER

Ready to take your first steps into version control? Meet the speaker who will guide you through the essentials of Git!

We're excited to introduce Sir Tj! With his guidance, you'll learn the fundamentals of Git—from tracking changes and managing repositories to collaborating effectively on projects. Whether you're completely new to Git or looking to strengthen your foundation, this workshop is a great place to start.

**TJAKOEN
STOLK**



Software Engineer
Chief Technical Officer
Computer Science Instructor



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

Introduction to VCS

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



THE PROBLEM

Have You Ever Done This?
You changed your code but wanted
to keep the old version.

So you end up with:

```
Car.java  
Car1.java  
CarFINAL.java  
CarFINALFINAL.java  
CarFINALFINAL2.java
```

As projects become larger:

- Changes become difficult to track
- Finding errors becomes harder
- Old versions become difficult to manage
- Collaboration becomes more complicated

So... how do we manage all these changes?



WHAT IS VERSION CONTROL?

Version Control System (VCS)

- A tool used in software development to track and manage changes to source code.
- VCS keeps a record of modifications to the code.
- Lets you go back and compare versions when something breaks

Why Version Control Matters

- Track → what changed?
- Compare → what was different?
- Restore → what did the project look like before?
- Collaborate → how can multiple developers work together?
- It is a backbone of modern software development because it supports both change tracking and collaboration.



CONCEPTS TO REMEMBER

1 REPOSITORY

A repository is where project files are stored. It also keeps a history of all changes made to the project.

2 COMMIT / REVISION

A commit is a specific saved version of a file or project. It records the changes made at a particular point in time.

3 BRANCH

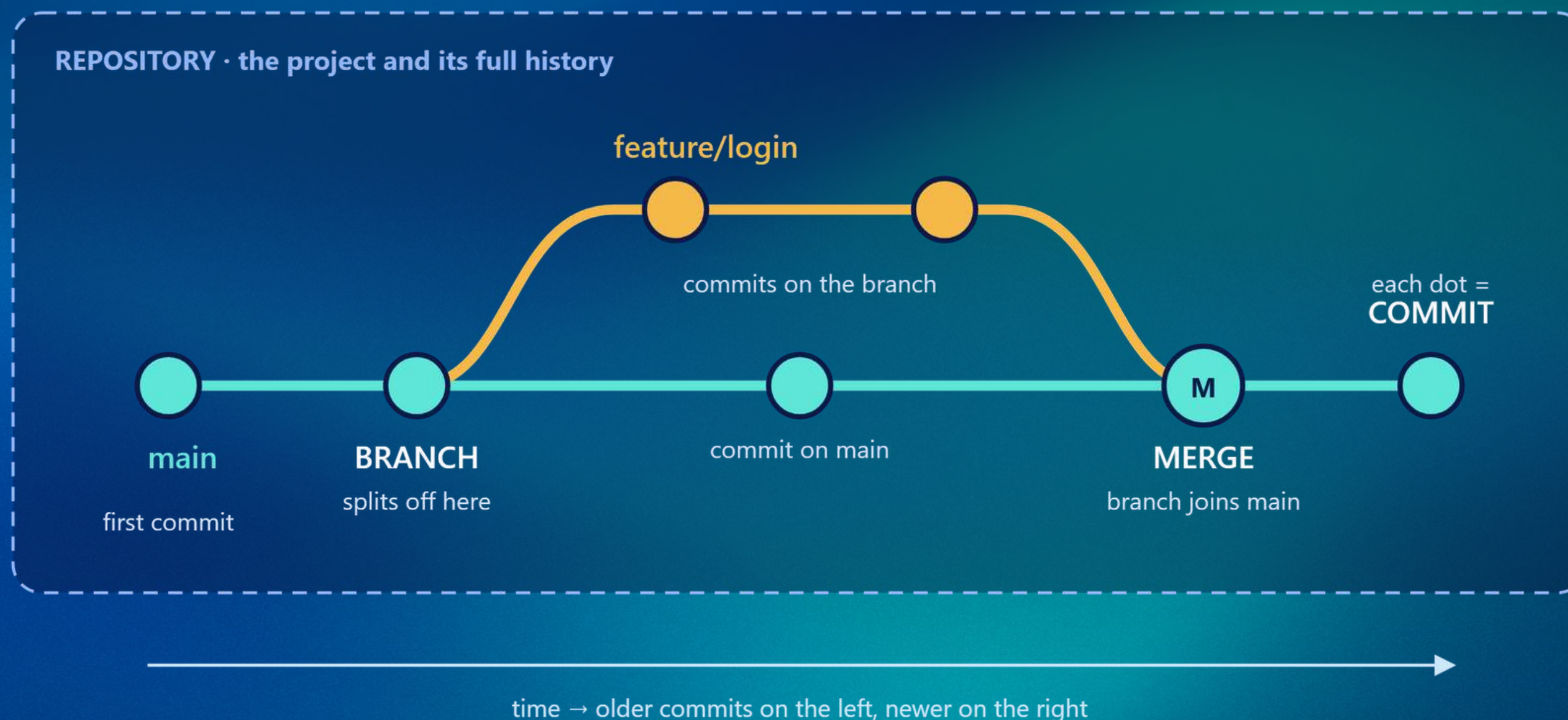
A branch is a separate line of development within a project. It allows you to work on features, fixes, or documentation without affecting the main branch.

4 MERGE

A merge combines changes from one branch into another. It allows completed work from different branches to become part of the main project.



HOW IT FITS TOGETHER





LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

Introduction to Git & GitHub

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



GIT

vs.

GITHUB

- A version control system that tracks project changes.
- It runs directly on your computer.
- It tracks changes made to files.
- It works with repositories stored locally.
- It manages the history of your project.

Git tracks the changes.

- An online platform for storing and sharing repositories.
- It is hosted online and accessed through the internet.
- It stores and shares project repositories.
- It supports collaboration between users.
- It provides tools for online collaboration.

GitHub stores and shares those changes online.



EXAMPLE SCENARIO

Imagine you are building an app with 3 friends

Each developer can have a copy of the project on their own computer using Git.

Developer A
Login page

Developer B
Bug fixes

Developer C
Other pages

When their work is ready:

Local Git
Repository



Push



GitHub



Team



KNOW THESE TERMS

1 LOCAL REPOSITORY

A Git repository stored on your computer where you can make changes, commits, and branches locally.

2 REMOTE REPOSITORY

A Git repository stored elsewhere, usually on GitHub, where your project can be shared and accessed online.

3 CLONE

Downloads an existing repository to your computer along with its files, Git history, and branches.

4 PUSH

Sends your local commits to a remote repository to update the online project.

5 PULL

Gets the latest changes from the remote repository and applies them to your local repository.



KNOW THESE TERMS

BASIC WORKFLOW

[Clone] → [Edit] → [Add] → [Commit] → [Push] → [GitHub]



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

Hands-On Git Fundamentals

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



CREATE YOUR FIRST GIT REPOSITORY

You will:

- Create a project folder
- Initialize Git
- Create a README file
- Stage the file
- Make your first commit
- View your commit history

STEP 0: OPEN GIT BASH OR THE VS CODE TERMINAL.



LEAGUE OF OUTSTANDING
PROGRAMMERS

BEFORE WE START

Run these once, or your first commit will fail.

1. Install Git from git-scm.com (on Windows, this includes Git Bash)
2. `git --version` (check that it installed)
3. `git config --global user.name "Your Name"`
4. `git config --global user.email "you@example.com"`
5. Use the same email as your GitHub account



STEP 1: NAVIGATE TO DESKTOP

Navigate to your Desktop:

```
cd ~/Desktop
```

Then create your project folder:

```
mkdir git-workshop  
cd git-workshop
```

Expected Location:

Desktop/
└── git-workshop/



STEP 2: INITIALIZE GIT

Create Your Local Repository

Inside the git-workshop
folder, run:

```
git init
```

What happened?
Git creates a repository inside
your project folder.

```
git-workshop/  
└── .git/
```

Your project is now being
tracked by Git.



STEP 3: CREATE README.MD

Create Your First File

Run:

```
echo "# Git Workshop" > README.md
```

Then check the repository:

```
git status
```

Note: The file has not been committed yet.



STEP 4: STAGE THE FILE

Add README.md to the Staging Area

Run:

```
git add README.md
```

Then:

```
git status
```

Remember:

Working Directory



git add



Staging Area

git add prepares changes for
a commit.



STEP 5: MAKE YOUR FIRST COMMIT

Save Your First Version

Run:

```
git commit -m "Add README"
```

Then check your history:

```
git log
```



STEP 6: UPDATE README.MD

Open the Project in Visual Studio Code

Open:

```
git-workshop/
```

Then edit README.md.

Add:

```
Name: [Your Name]  
Program: [Your Program]  
Year Level: [Your Year Level]
```

Save the file



STEP 7: CHECK YOUR CHANGES

Run:

```
git status
```

Now prepare the update:

```
git add README.md
```

What Changed?

Git should recognize that README.md has been modified.



STEP 8: CREATE YOUR SECOND COMMIT

Run:

```
git commit -m "Update README"
```

Then:

```
git log
```

You should now have 2 commits:

Commit #2 — Update README

Commit #1 — Add README



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

10-Minute Break

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

GitHub Integration

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



FROM LOCAL TO GITHUB

So far, everything exists on your computer.

But what if:

- You want to back up your project?
- You want to share it?
- You want other developers to work on it?

FIRST, GO TO GITHUB.



CONNECT YOUR REPOSITORY TO GITHUB

Step 1: Create a GitHub Repository

Then copy the repository URL.

Step 2: Connect Your Local Repository

```
git remote add origin <repository-url>
```



CONNECT YOUR REPOSITORY TO GITHUB

Step 3: Rename Branch

```
git branch -M main
```

Step 4: Push Your Commits

```
git push -u origin main
```



LEAGUE OF OUTSTANDING
PROGRAMMERS

SIGNING IN TO GITHUB

Your first git push will ask you to sign in.

1. A window opens: choose “Sign in with your browser”
2. Log in to GitHub and click Authorize
3. Typing your GitHub password in the terminal will not work
4. No window? Make a personal access token in GitHub settings and paste it when Git asks for a password



PUSH

and

PULL

Local → GitHub

```
git push
```

Used to upload your commits to the remote repository.

GitHub → Local

```
git pull origin main
```

Used to retrieve and integrate changes from the remote repository.

If another developer already made changes and uploaded them to GitHub, we can retrieve those changes using: `git pull origin main`.



IMPORTANT COMMANDS

```
git remote add origin <repository-url> → Connect to GitHub  
git push → Upload changes  
git pull → Get latest changes  
git clone → Copy an existing repository
```

On the other hand, if the GitHub repository already exists and we do not have it yet on our computer, we can use: `git clone <repository-url>`. This creates a local copy of the repository, including its files and Git history.



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

Branching and Collaboration

 LOOP.SOC.HAU@GMAIL.COM

 [HAUSOC.LOOP](https://www.facebook.com/HAUSOC.LOOP)

 [HAUSOC.LOOP](https://www.instagram.com/HAUSOC.LOOP)



WHY DO WE NEED BRANCHES?

Imagine 4 Developers:

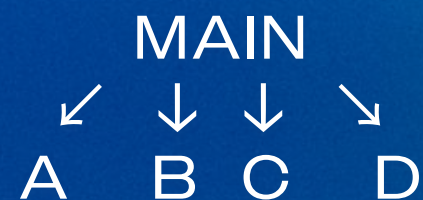
Developer A → Login

Developer B → Registration

Developer C → Home Page

Developer D → Bug Fix

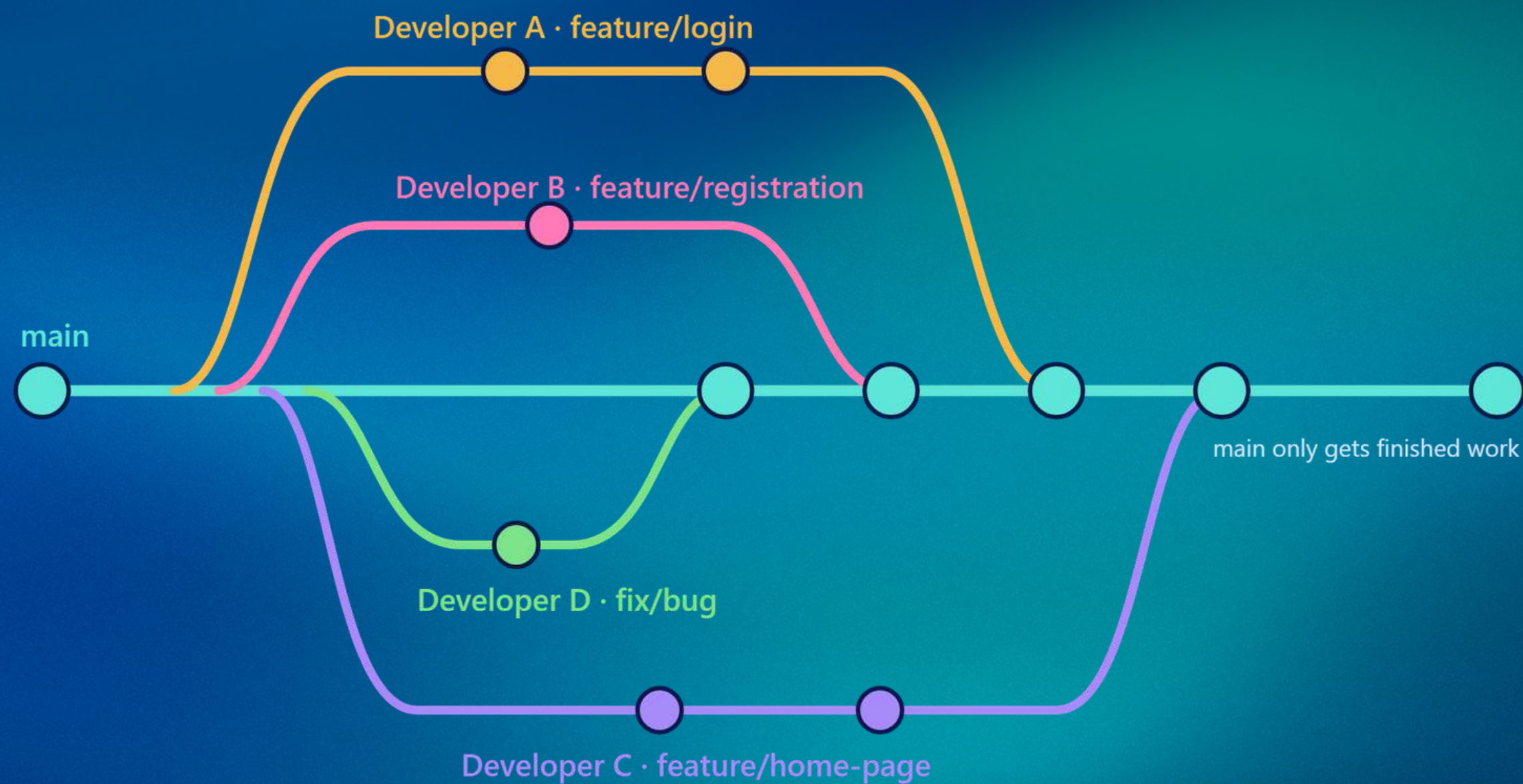
If everyone directly changes main:



This can become difficult to manage, especially when unfinished or broken code immediately enters the main version of the app. Because of this, we need to use branches.



EVERYONE GETS A BRANCH





1 MAIN BRANCH

Contains the main working version of the project.

2 FEATURE BRANCH

Used to develop something separately.

Example:

main



feature/login

BRANCHES

Commands

Create and switch to a branch:

```
git switch -c feature/login
```

View branches:

```
git branch
```



WORKFLOW

Feature Branch



Push



Pull Request



Review



Merge



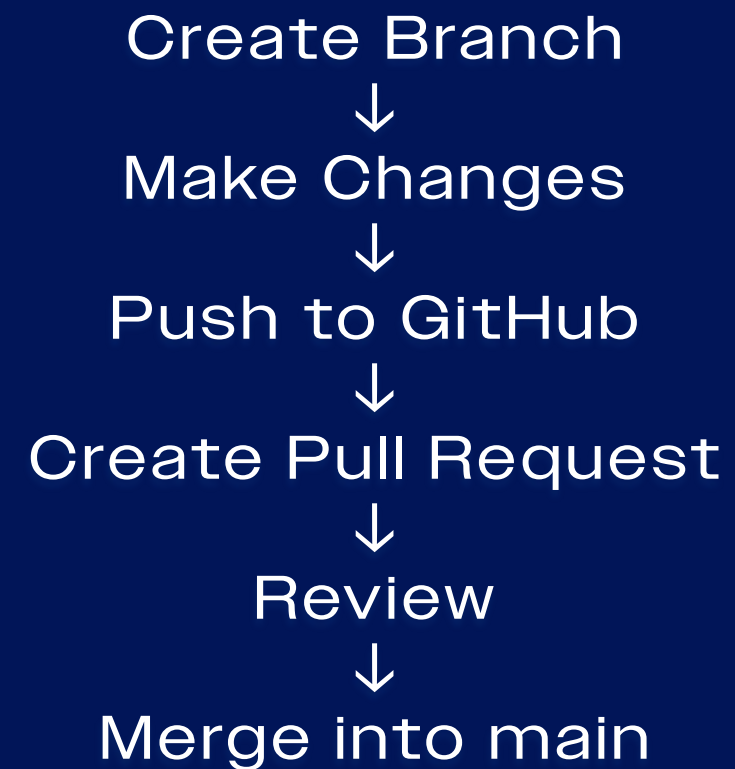
main

PULL REQUEST (PR)

Once the feature is ready, a Pull Request (PR) can be created on GitHub. A Pull Request allows other developers to review the changes before they are merged into the main branch.



FROM BRANCH TO MAIN

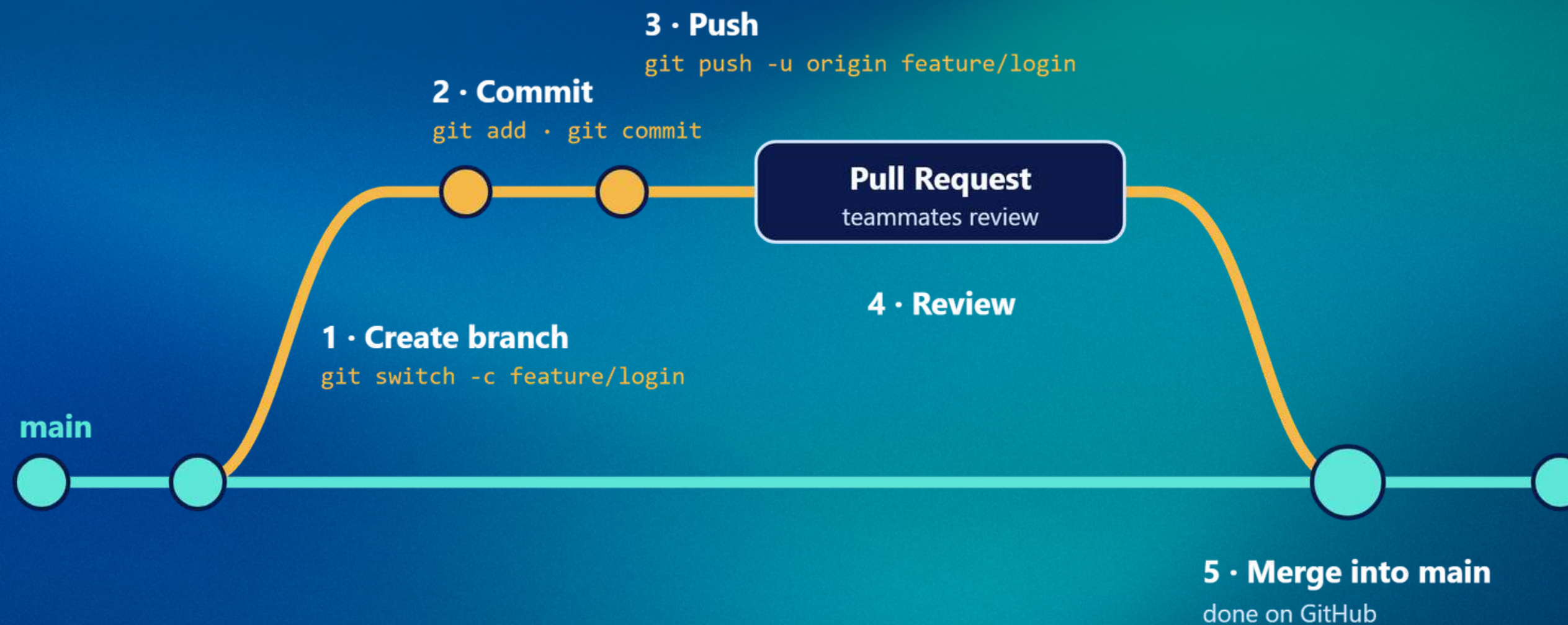


BRANCH COLLABORATION WORKFLOW

In this way, developers work on different parts of the project without immediately affecting each other's work.



FROM BRANCH TO MAIN





LEAGUE OF OUTSTANDING
PROGRAMMERS

WHAT IS A MERGE CONFLICT?

A merge conflict can happen when two developers modify the same part of the same file differently.

Example:

Developer A

```
String message = "Welcome!";
```

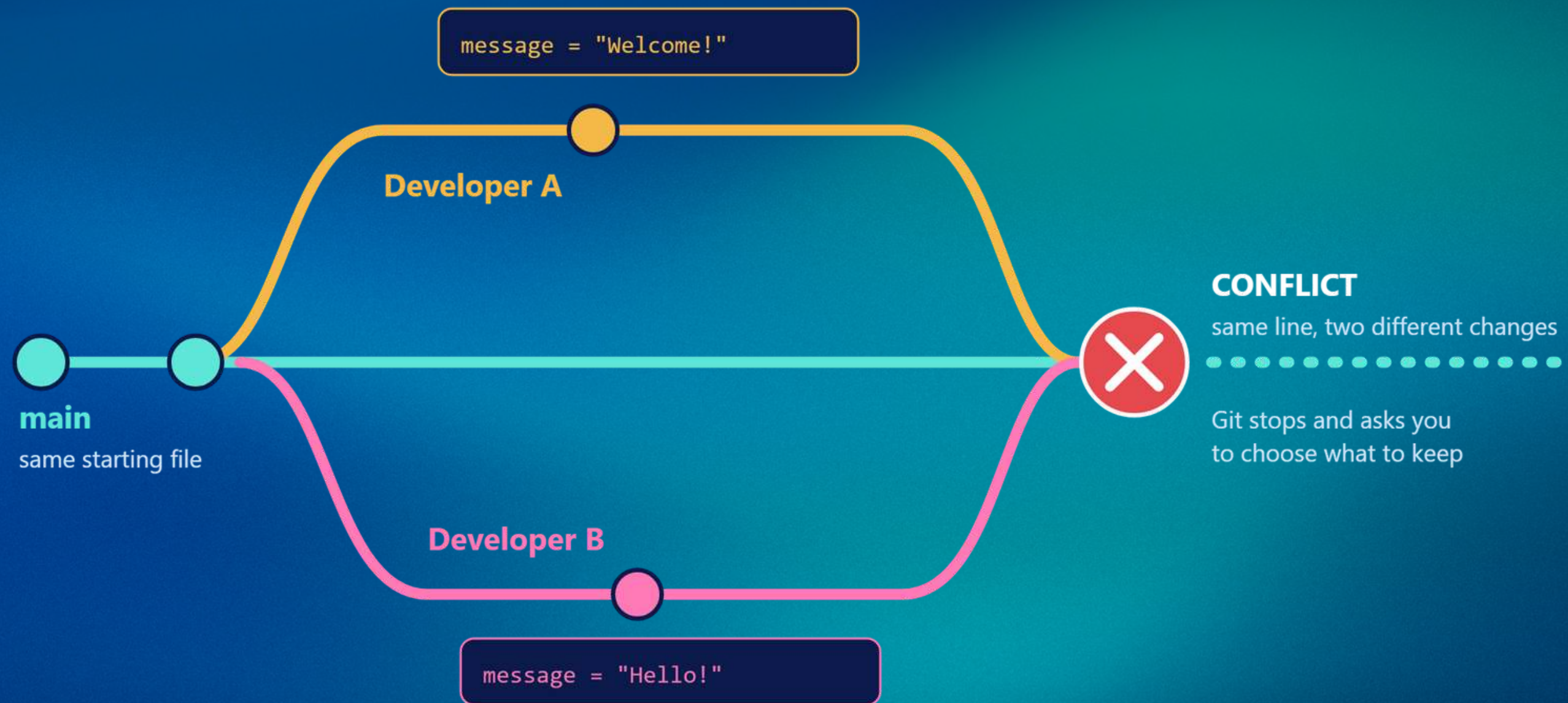
Developer B

```
String message = "Hello!";
```

Git cannot automatically determine which change should be kept.
The developers must then resolve it manually.



WHY CONFLICTS HAPPEN





LEAGUE OF OUTSTANDING
PROGRAMMERS

HOW TO RESOLVE A CONFLICT

When Git can't merge on its own, it marks the conflict inside the file.

How to fix it:

1. Open the file and find the <<<<<<, ===== and >>>>>> lines.
2. Keep the code you want and delete the marker lines.
3. Run git add, then git commit, to finish the merge.



LEAGUE OF OUTSTANDING PROGRAMMERS

VERSION ONE:

Group Collaboration Activity

 LOOP.SOC.HAU@GMAIL.COM

 HAUSOC.LOOP

 HAUSOC.LOOP



GROUP COLLABORATION ACTIVITY

Form Your Groups
10 Groups Total

- Based on seating arrangement
- At least 4 members per group
- Each group will receive a shared GitHub repository

Your task is to work together to complete 4 Python programming exercises.



LEAGUE OF OUTSTANDING
PROGRAMMERS

HOW YOUR GROUP WORKS

One repository per group. One branch per person.

1. One member creates the repository on GitHub
2. Settings > Collaborators: add everyone's GitHub username
3. Everyone accepts the invite, then runs `git clone <url>`
4. Pick an exercise and branch: `git switch -c exercise-1`
5. Commit, then `git push -u origin exercise-1`
6. Open a pull request. A teammate reviews and merges it
7. Before the next exercise: `git switch main`, then `git pull`



WHEN THINGS GO WRONG

Read the error first. It usually tells you what to do.

1. “not a git repository”: wrong folder, cd into your project
2. “Please tell me who you are”: run the git config commands
3. “rejected ... fetch first”: run git pull, then push again
4. “Authentication failed”: sign in through the browser window
5. “CONFLICT”: fix the marked lines, then git add and git commit



GIT CHEAT SHEET

Today's commands, in the order you'll use them.

- 1.git config --global user.name "Your Name" (set up once)
- 2.git init or git clone <url> (start a repository)
- 3.git status (see what changed)
- 4.git add <file> (stage your changes)
- 5.git commit -m "message" (save a version)
- 6.git push / git pull (sync with GitHub)
- 7.git switch -c <branch> (create a branch)