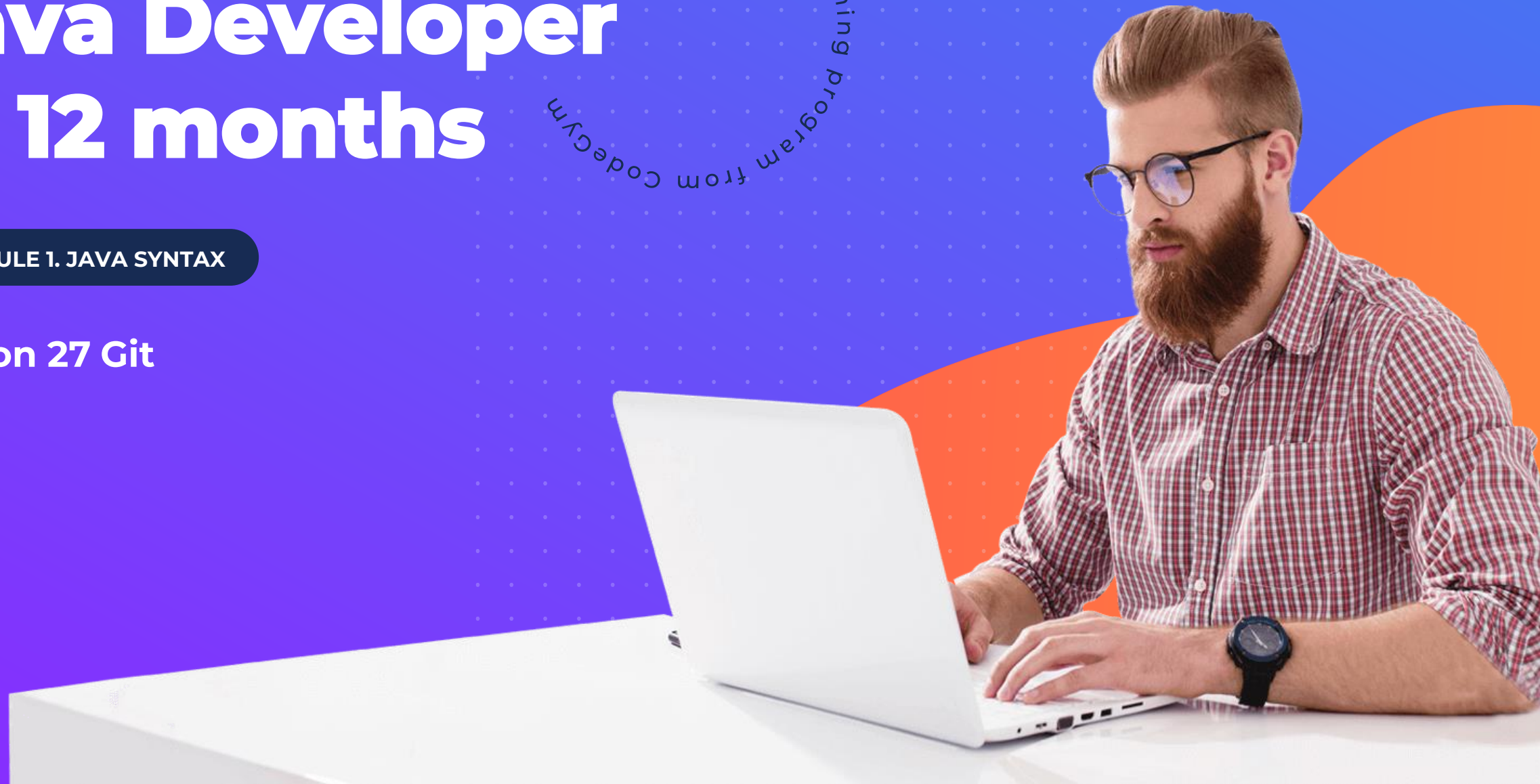
CODEGYM

Mentor-supported training program from CodeGym

Java Developer in 12 months

MODULE 1. JAVA SYNTAX

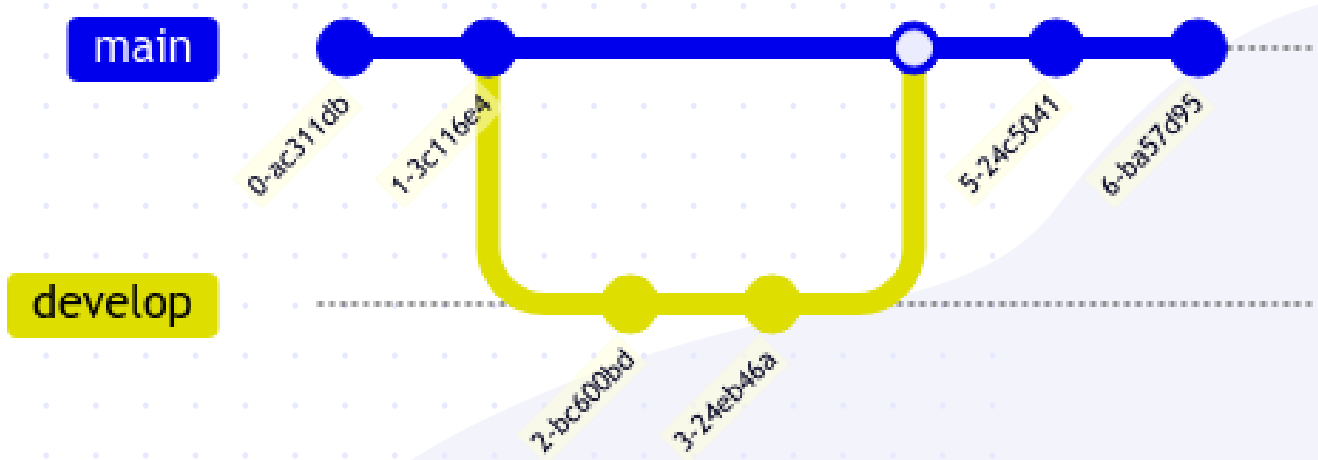
Lesson 27 Git



CODEGYM

Today's Plan

1. What is Git
2. Creating an account on GitHub
    - Creating a repository
3. Working with GitHub from IDE
    - Cloning a project
    - Working with branches
    - Modifying files and commits
    - Bug reports



CODEGYM

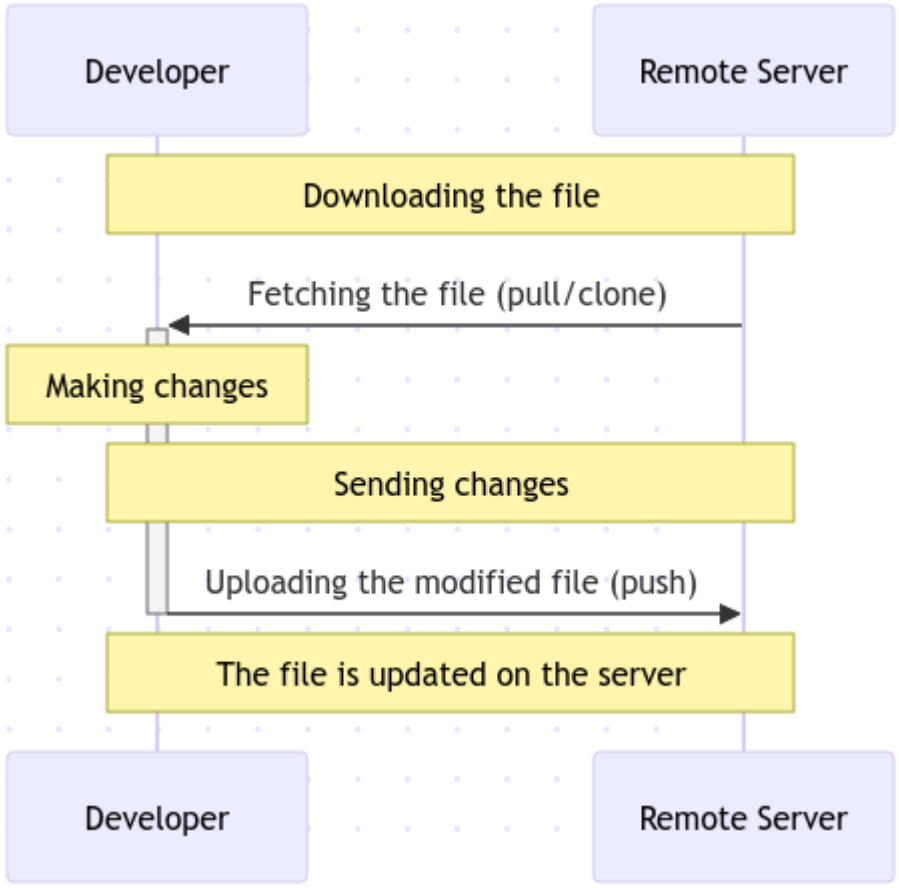
Why do we need GIT?

Git is a system that tracks changes in project files.

Instead of copying project folders, you simply "commit" (save) changes.

If something goes wrong, you can roll back and continue working.

And when working in a team, Git is absolutely essential—otherwise, there would be chaos.



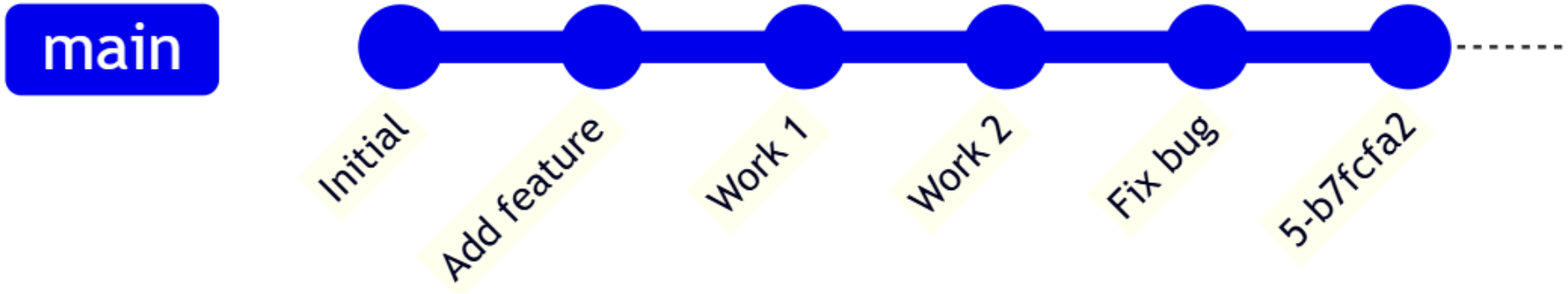
CODEGYM

Git as a Version Control System

When developing a project, Git creates a **timeline** of changes.

Each save (**commit**) becomes a point in history that you can revert to. This is especially useful for debugging or when you need to undo changes.

A commit hash is a unique identifier that Git generates for each commit. 5-b7fcfa2 is a commit hash.



CODEGYM

Key Advantages of Git

- **Commits:** Git allows you to save "snapshots" of your project (commits) with descriptions of changes.
- **Change history:** you can always see who made changes, when, and what was modified in the code.
- **Rollback to previous versions:** in the IDE, you can easily revert to any commit and restore your code.
- **Security:** changes do not overwrite each other, ensuring you don't lose a working version.

CODEGYM

Three Code Storage Locations

Git can be local, centralized, or distributed:

1. **Local** is installed on a single computer and stores files only in one instance within the configured environment—suitable if a programmer is working alone.

2. **Centralized** is located on a shared server and stores all files there.

3. **Distributed** stores data both in a shared cloud repository and on team members' devices.

CODEGYM

Git vs. GitHub: What's the Difference?

It's important to understand that **Git** and **GitHub** are different things.

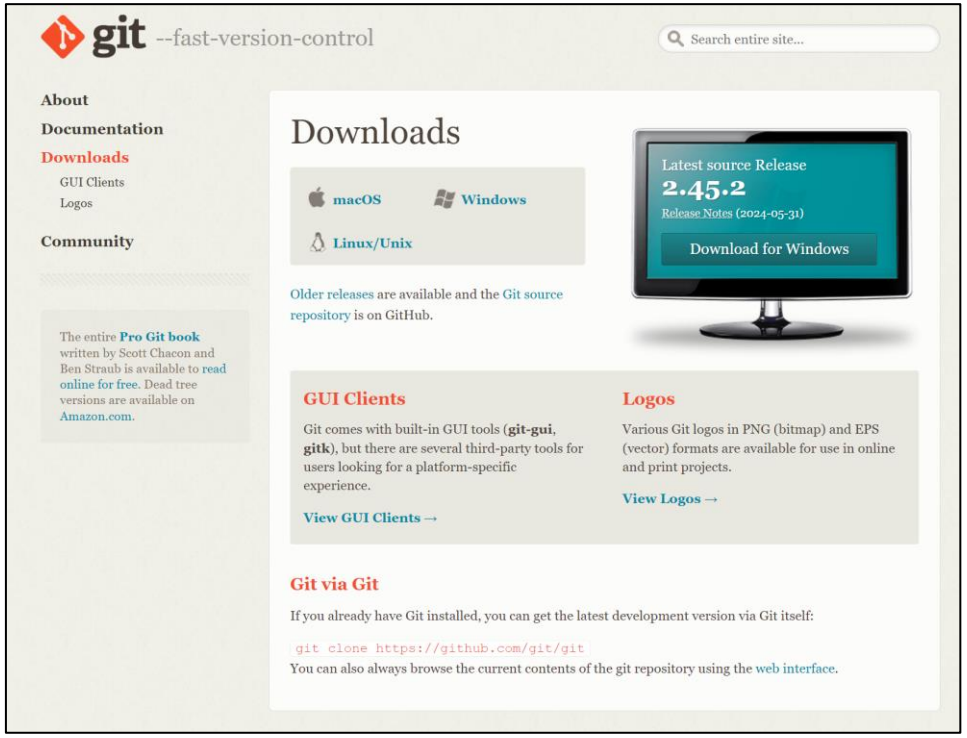
You can use Git locally on your computer without the internet. GitHub, on the other hand, is a platform where you store your projects "in the cloud," share them with others, and collaborate with your team.

There are also GitLab and Bitbucket. But the idea is the same—Git manages versions, while these services allow you to store repositories and work with them conveniently.

CODEGYM

Installing Git and Registering on GitHub

Go to the page
https://git-scm.com/downloads



Windows
As usual, download the .exe file and run it.

Linux
To check if Git is installed, open the terminal and type: git --version.
For Ubuntu and derivative distributions, type: sudo apt-get install git.

macOS
To check if Git is installed, open the terminal and type: git --version.

The easiest way is to download the latest version from :
https://sourceforge.net/projects/git-osx-installer/files/

CODEGYM

Key Terms

Repository ("Repo") – this is where the entire project code is stored, and Git tracks all changes.

Commit – like a save in a game. It saves the current state of the code so that you can return to it later.

Branch – like a parallel universe for your code. By default, there is the main branch (master or main). You can create new branches to develop new features or fix bugs without affecting the main version of the code.

Merge – the process of combining changes from one branch into another.

Conflict – occurs when two people modify the same line of code in different branches.

Push – sending your commits to the server.

Pull – downloading the latest changes from the server to your local machine.

HEAD indicates where you are currently working, which branch and commit.

CODEGYM

Cloning a Repository

To start working with a project that is already stored on GitHub, you need to "clone" it. Cloning creates a local copy of the repository on your computer.

Difference between cloning via link and through an account in IDE (authentication)

Method	Access	Convenience	Security
Cloning via link (HTTPS)	Public/Private (with authentication)	Less convenient (need to enter URL and password)	Less secure (password is transmitted with every action)
Cloning via link (SSH)	Public/Private (with authentication)	Convenient (after setting up SSH)	More secure (uses SSH keys)
Through account in IDE	Your repositories + those available to you	Most convenient	Depends on account security settings and trust in the IDE

CODEGYM

Forking a Project (Fork)

A **Fork** is the creation of a full copy of someone else's repository in your account on GitHub/GitLab/Bitbucket. You get your own repository, independent of the original.

Why Forking is Needed:

- You can freely modify the code in your fork and then propose changes to the original repository's author through a **Pull Request**.
- A fork can be used as a base for your own project, without affecting the original repository.
- Forking is a great way to study someone else's code while making your own edits.
- How to Fork: On the repository page on GitHub/GitLab/Bitbucket, there is a "Fork" button.

Connection to the Original Repository:

- A fork remembers the original repository (upstream).
- You can pull changes from the original repository into your fork to keep it up to date.

CODEGYM

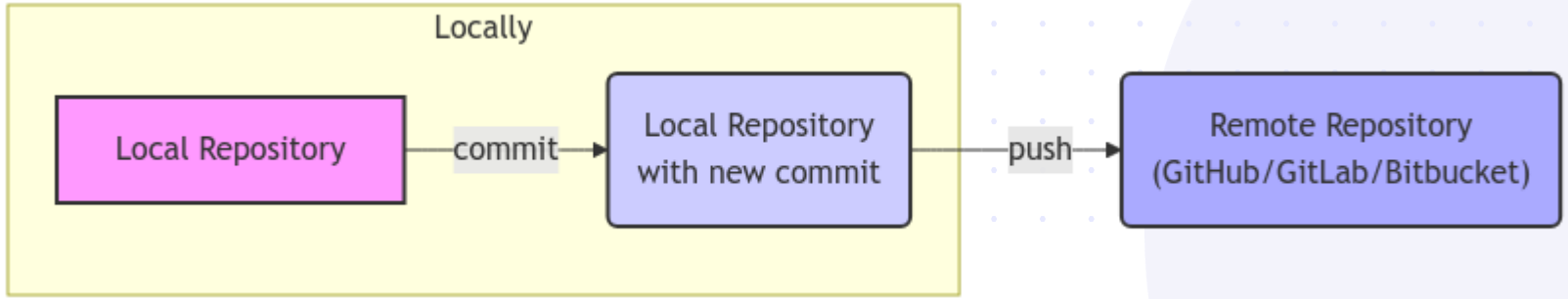
Commit and Push

After making changes to the project files, you need to commit those changes and push them to the remote server.

To do this, there are two main commands: **commit** *и* **push**. In the IDE, all of this is done via buttons, no need to type commands.

Commit Messages:

- Always write clearly and understandably!
- **Use the imperative mood in the present tense.**
- Write as if you're giving a command: "Add", "Fix", "Delete", "Change", not "Added", "Fixed", "Deleted". This is the standard convention in Git.



CODEGYM

Commit Types

Many teams use a convention for commit types that are specified at the beginning of the commit header. This helps quickly understand which category the commit belongs to. Examples:

- **feat** - new functionality (feature).
- **fix** - bug fix.
- **docs** - changes in documentation.
- **style** - changes that do not affect the meaning of the code (formatting, spaces, semicolons, etc.).
- **refactor** - code refactoring.
- **test** - adding or modifying tests.
- **chore** - changes that do not relate to source code, tests, or documentation (e.g., updating dependencies).

Example: feat: add user profile page

CODEGYM

Pull/Update Project

To avoid working with an outdated version, you need to regularly “**pull**” changes to your local repository.

In the IDE (JB), this is done via the menu: VCS/Git -> Pull. It may also be called "Update Project" – the idea is the same. The IDE will automatically download all the latest changes and merge them with your code.

Important Notes:

- Make sure to pull/Update Project regularly, especially before starting work on a new task.
- Always do a pull/Update Project before pushing your changes to the remote repository. This will help avoid merge conflicts.
- If merge conflicts arise, the IDE will help you resolve them.

CODEGYM

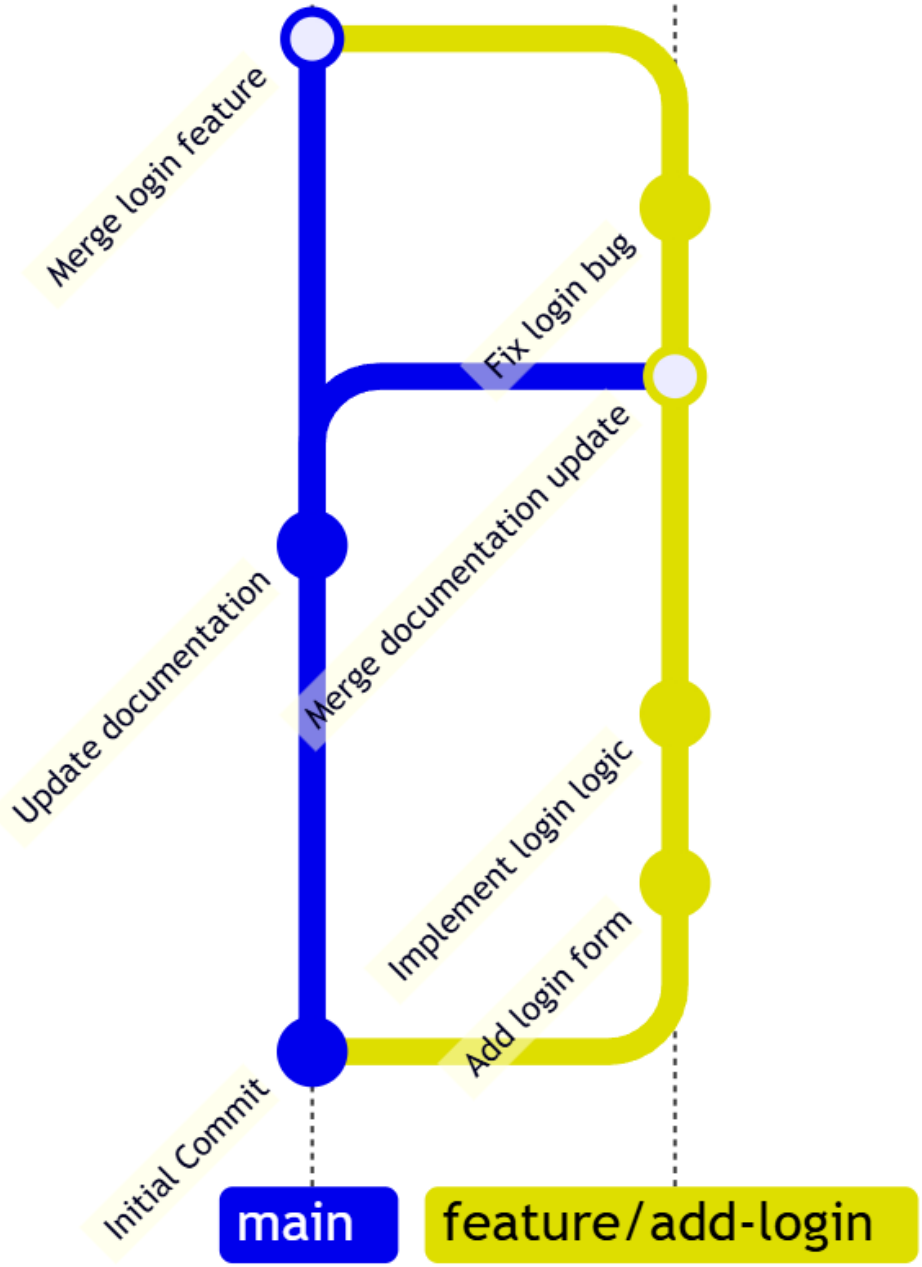
Branch and merge

Branches are like parallel universes for your code.

Merge is when you take code from one branch and add it to another branch.

Explanation of the visualization:

- **main** - the main branch
- **feature/add-login** - a branch for developing a new feature.
- **commit** - commits in different branches.
- **checkout** - switching between branches.
- **branch** - creating a branch
- **merge** - merging the feature/add-login branch into main (after the feature development is completed).
- Work is done simultaneously in **main** and **feature/add-login**.



CODEGYM

Merge Conflict

A merge conflict occurs when Git cannot automatically merge changes from two branches.

The conflict arises because the same line of code in the same file has been modified in two different branches (HEAD and feature/change-header).

Git cannot automatically determine which of these two header variants should be in the final file after the merge. It doesn't know which text is "more correct" or "more important."

```
<<<<<<< HEAD
<div class="header">
  <h1>My Awesome Website</h1>
</div>
=======
<div class="header">
  <h1>The Best Website Ever</h1>
</div>
>>>>>>> feature/change-header
```

CODEGYM

Deleting Commits from a Branch

If you've made several commits locally in your branch (e.g., feature/my-feature), **but haven't pushed them to GitHub yet,** and realize that the last two commits are incorrect and you want to delete them.

IDE JetBrains:

- Go to the "Log" tab
- Find the commit you want to reset to (i.e., the commit that should be the last one in your branch).
- Right-click on that commit and select "**Reset Current Branch to Here...**".
- Choose the "Hard" option. This will irreversibly delete the changes from the commits you want to remove.
- Click "Reset": confirm the reset of the branch.

CODEGYM

Changes after Push

If you have already pushed a commit to GitHub and later realize that you need to make further changes (fix an error, add something, improve the code), the best and safest approach is to make those changes and create a new commit.

You should not try to modify a commit that has already been pushed (this is complex and could lead to issues).

CODEGYM

.gitignore

This is a file that contains names of other files that **git should not track**, and which will not be included in the GitHub repository.

Rules for .gitignore:

- * - replaces any number of any characters.
  For example, ***.log** ignores all files with the **.log** extension.
- ? - replaces a single character.
- [] - specifies a range of characters.
  For example, **[abc]** matches any of the characters a, b or c.
- ! - excludes files that match the pattern.
  For example, **!important.txt** will track **important.txt**, even if there's a rule ***.txt**.
- / - at the beginning of the pattern indicates the root of the repository. **/temp/** ignores the **temp** folder in the root of the repository.
- ** - matches any number of nested folders.
  For example, ****/temp** ignores **temp** folders at any level of nesting.

Gitignore.io
You can visit the website and create a .gitignore for your project
https://www.toptal.com/developers/gitignore

CODEGYM

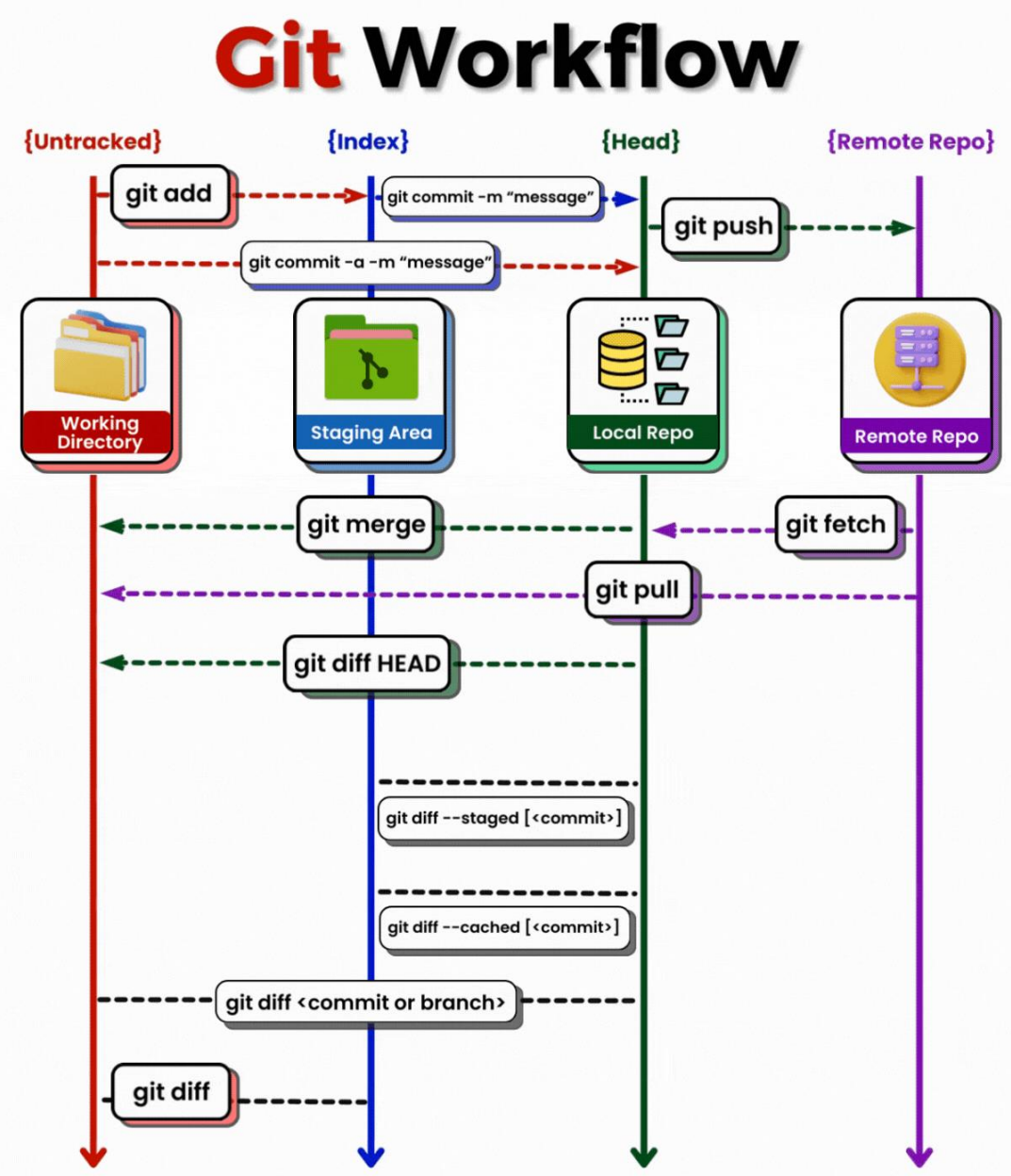
Git WorkFlow

Or how everything works “under the hood”

Git has several states:

- untracked;
- modified;
- staged;
- committed.

[Gif]
[Gif 2]



CODEGYM

Working from the Console

1. Free book Pro Git https://git-scm.com/book/en/v2
2. Free online course https://githowto.com/
3. Interactive visual online trainer https://learngitbranching.js.org

CODEGYM

That’s good!

Me: Take a look at my GitHub, there are great projects!
My projects:

CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX