1

The Core Concept
(Git vs. GitHub)

Analogy:

```
git merge [branch_name]
```

Git
(The Coffee)

The local tool on your computer that tracks every change.

GitHub
(The Coffee Shop)

The cloud-based home where you store and share your work with

Key Difference: → **Git runs locally;**
GitHub is a remote → **online server.**

2

The Three Stages of Git

(Internal Architecture)

Git moves your work through three distinct areas before it goes to the cloud:

Working Directory

Your local folder where you write code (The "Work" zone)

Staging Area

The intermediate "Check-point" where you prepare files for saving

Local Repository

The digital cabinet where all versions are saved on your PC

Remote Repository

The final destination (GitHub) for backup and sharing.

3

Getting Started (init vs. clone)

Command your work through three distinct areas before it goes to the cloud;

Command	Usage	When to use it
✓ git init	git init	When starting a brand new project locally from scratch.
✓ git clone [URL]	git clone [URL]	When you want to copy an existing project from GitHub to your PC.

✓ **Working Directory**

Your local folder where you write code (The "Work" zone)

✓ **Local Repository**

The digital cabinet where all versions are saved on your PC.

4

Checking the Status (status)

Command: git status

```
git status
```

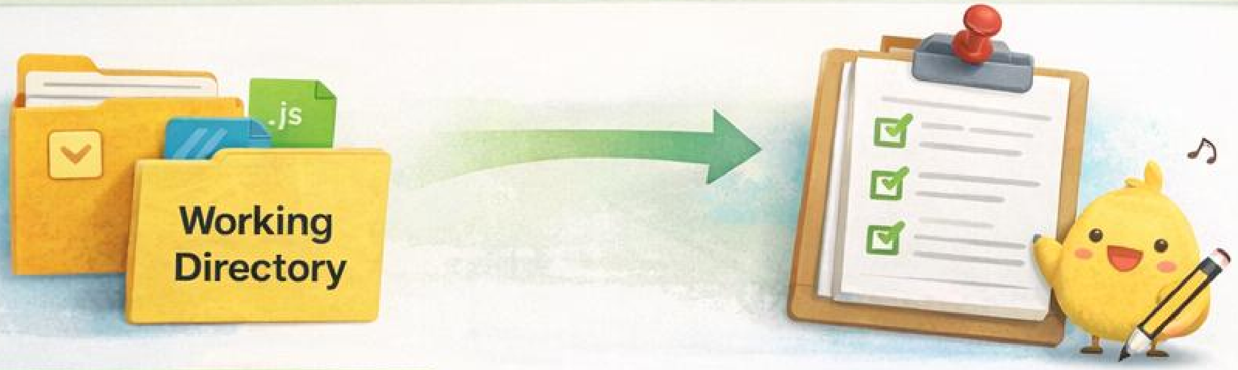
✓ **Red files:** Modified but not yet added to the staging area.

✓ **Green files:** Staged and ready to be committed (saved)

Untracked files: New files Git doesn't know about yet.

5 Staging Your Work (add)

The process of moving changes from the Working Directory to the Staging Area.



git add Commands

- `git add filename.txt` - Stages one specific file.
- `git add .` - Stages everything in the current directory.
- `git add -A` or `git add --all` - Stages every change across the entire project.
- `git add *.txt` - Stages all files with a specific extension.
- `git add *.txt` - Stages all files with a specific extension.

REMEMBER:
The staging area is like a clipboard where you gather changes before committing them to the repository.

6

Saving a Version (commit)

Command: git commit -m

```
git commit -m "Your message here"
```

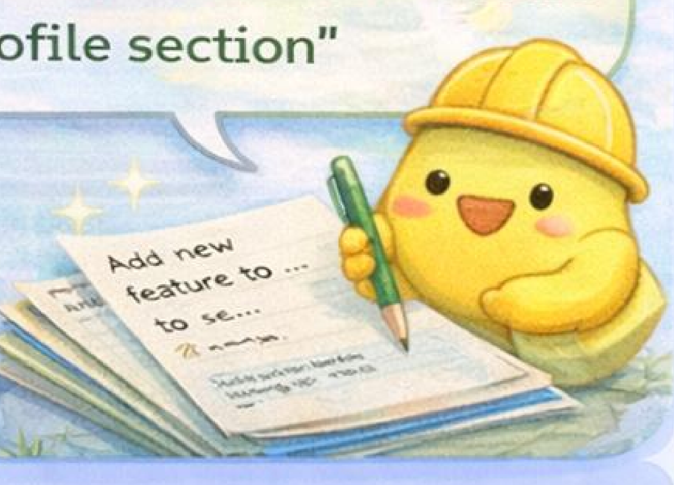
Definition:

"***Commit***" means permanently saving those staged changes into the project's history.

The Message:

Always include a clear description of what you changed so your future self (or team) understands the version.

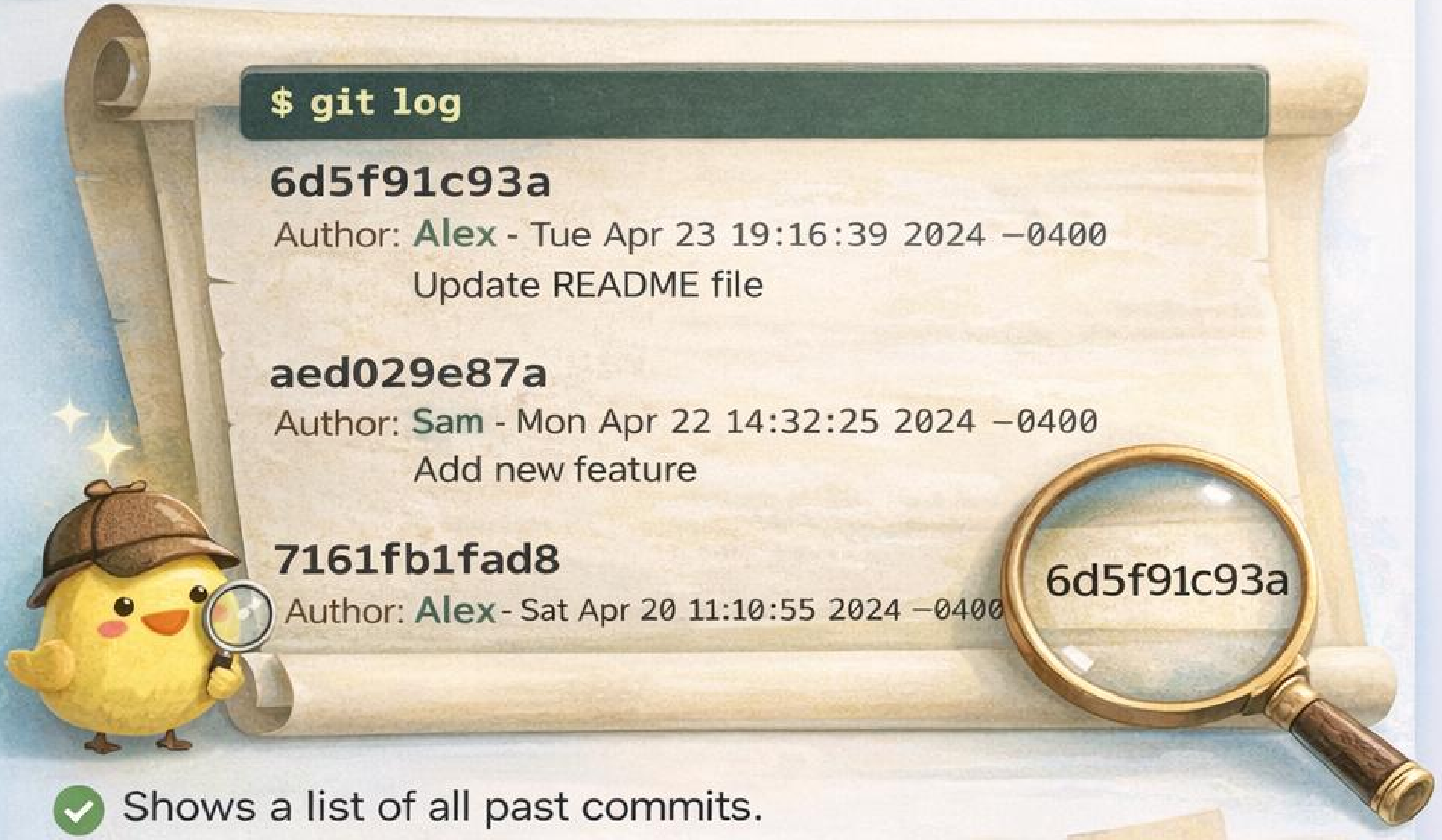
```
git commit -m "Add new feature to user profile section"
```



7

View the History (log)

Command: **git log**



```
$ git log
6d5f91c93a
Author: Alex - Tue Apr 23 19:16:39 2024 –0400
        Update README file

aed029e87a
Author: Sam - Mon Apr 22 14:32:25 2024 –0400
        Add new feature

7161fb1fad8
Author: Alex - Sat Apr 20 11:10:55 2024 –0400
```

- ✓ Shows a list of all past commits.
- ✓ `git log –oneline` Shows a simplified, one-line version of the history.

Commit ID: Each commit has a unique alphanumeric ID (Hash) used for navigation and undoing.

8 Undoing and Reverting

Command What it does

Command	What it does
git reset HEAD~	Undoes the very last commit and brings files back to the working directory.
git reset --hard	**Caution!** Completely deletes all uncommitted changes and reverts to the last save.
git restore [file]	Discards local changes in a specific file to match the last commit.
git revert [ID]	Creates a new commit that reverses the changes of an old commit (best for shared projects).

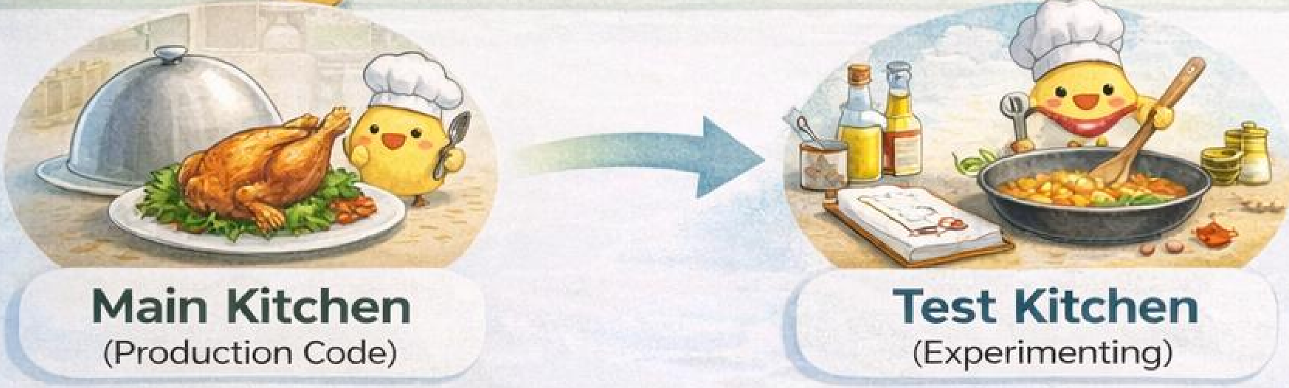
CAUTION!

This command is destructive & cannot be undone!

- ✓ Shows a list of all past commits.
- ✓ `git reset --hard` Caultɩeall/ Deletes all uncommitted changes
- ✓ **git revert [ID]** Creates a new commit that reverses the changes of an old commit (best for shared projects).

9 Branching (The Test Kitchen)

Analogy:



- ✓ **Main Branch:** The “Main Kitchen” (Production code).
- ✓ **Feature Branch:** A “Test Kitchen” where you experiment without breaking the main branch

- ✓ **git branch [name]**: Create a new branch
- ✓ **git checkout [name]:** Switch to that branch
- ✓ **git branch:** List all current branches

10

Merging and Conflicts

Command: **git merge [branch_name]**

```
git merge [branch_name]
```

Definition:

Combining changes from a feature branch back into the main branch.

Merge Conflict:

Occurs when the same line of code is changed in two different ways. Git stops and asks you to pick which version to keep.

Merge Conflict:

Occurs when the same line of code is changed in two different ways. Git stops and asks you to pick which version to keep.

```
git commit -m "Add new feature to user profile section"
```

```
<<<< HEAD
Call new function();
=======
chicken riding();
>>>>>> feature branch.
```

11

Stashing (stash)

Command: **git stash**

✓ **Scenario:**

You have unfinished work but need to switch branches quickly.

✓ **Action:**

git stash "hides" your current work in a temporary drawer.

✓ **Retrieve:**

git stash pop brings your work back when you return.

12

Syncing with GitHub
(push & pull)

The bridge between your local PC and the Cloud.

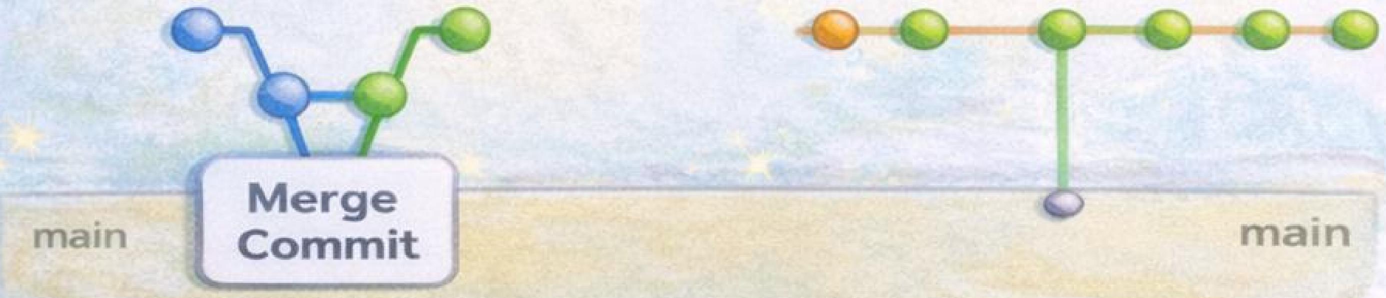
✓ git push origin main	Sends your local commits to GitHub.
✓ git fetch	Checks if there are any new changes on GitHub without applying them.
✓ git pull	Downloads new changes from GitHub AND merges them into your code automatically.

13 Rebase vs. Merge

Merge:

Combines branches and creates a "**Merge Commit.**"

It keeps the history of both branches exactly as they happered.



Rebase (git rebase):

Moves your branch to the "tip" of the main branch. It makes your project history look like one straight line (cleaner but more advanced)

14

The Pull Request (PR)

The GitHub Workflow:

✓ **Branch:**

Create a branch for a new feature.

✓ **Commit:**

Make your changes locally.

✓ **Push:**

Send the branch to GitHub.

✓ **Push:**

Send the branch to GitHub.

✓ **Pull Request:**

Open a "Request" on GitHub for the team to review your code before it is merged into the main project.