Lesson: Git's Mental Model

What you'll learn

- Why version control exists and what problems it solves on a real team.
- The four core places your work lives: the working tree, the staging area, the local repository, and (later) a remote.
- What a commit actually is — a *snapshot*, not a list of changes — and why that matters.
- What the hidden `.git` directory holds and why you must never delete it by accident.
- How to create a repository with `git init` and set your identity with `git config`.

By the end you will be able to explain, in your own words, what happens when you "save" work in Git — the foundation everything else in this module builds on.

The lesson

1. What problem does version control solve?

Imagine you wrote a backup script last week. Today you "improved" it, but now it is broken, and you cannot remember exactly what you changed. You start making copies: `backup.sh`, `backup-final.sh`, `backup-final-REALLY.sh`. This is chaos, and it gets far worse when two people edit the same file.

A **version control system** (VCS) — a tool that records changes to files over time — fixes this. It lets you:

- See exactly what changed, when, and who changed it.
- Return to any earlier state of your project.
- Work on a new feature without breaking the working version.
- Combine your changes with a teammate's safely.

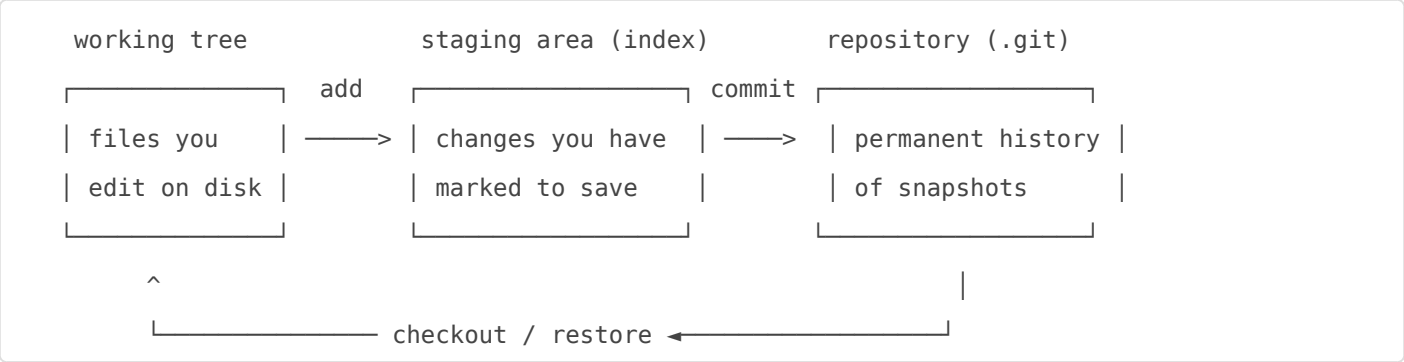
Git is the VCS the whole industry uses. It is *distributed*: every copy of a project is a full repository with the complete history, not just a checkout from a central server.

2. The repository (repo)

A **repository** is a project folder that Git is tracking. Physically, it is your normal files *plus* one hidden subfolder named `.git`. That hidden folder is the actual database of your history. Delete `.git` and you have an ordinary folder again — all history is gone. The visible files are just one version Git has written out for you to edit.

3. The three areas you work in

Git work flows through three areas inside one repo. Picture it like this:



1. **Working tree** — the real files in your folder that you open and edit.
2. **Staging area** (also called the **index**) — a holding zone. You *stage* a change to say "I want this to be part of my next save." This lets you commit some changes now and others later.
3. **Repository** — when you **commit**, Git takes everything currently staged and stores it permanently in `.git`.

The staging area is the part beginners find surprising. It exists so you can craft a clean, deliberate commit instead of dumping every random edit together.

4. A commit is a snapshot, not a diff

Many people think Git stores "the lines you changed." It does not. Each **commit** is a complete **snapshot** — a photo of what every tracked file looked like at that moment. Git is smart about storage (unchanged files are reused, not copied), but mentally you should think: *every commit is a full picture of the project.*

Each commit has:

- A unique ID (a long **hash** like `a1b2c3d…`) computed from its contents.
- The author, a date, and a **commit message** describing the change.
- A pointer to its **parent** — the commit that came before it.

Because each commit points to its parent, the history forms a chain:

```
  (A) ──> (B) ──> (C) ──> (D)   <- HEAD (where you are now)
first                  latest
```

`HEAD` is simply Git's bookmark for "the commit you are currently on."

5. Inside the `.git` directory

You rarely touch `.git` by hand, but knowing it is real helps. It contains the object database (your commits, file contents, and directory listings), the **refs** (names like branches pointing at commits), the `HEAD` file, and your repo-local config. Everything Git "remembers" lives here. Treat it as read-only and let Git commands manage it.

6. Create your first repository

Open a terminal on your lab VM and run:

```
mkdir my-scripts
cd my-scripts
git init
```

`git init` creates the `.git` folder — that folder is now a repository. You can confirm it exists:

```
ls -a        # you should see  .  ..  .git
git status   # Git tells you it's a fresh repo with nothing committed yet
```

7. Tell Git who you are

Every commit records an author. Set this once per machine (the `--global` flag writes to your user config, used by all your repos):

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

A couple of friendly defaults worth setting now:

```
git config --global init.defaultBranch main   # name the first branch "main"
git config --global core.editor nano           # editor Git opens for messages
```

Check what you set at any time:

```
git config --global --list
```

That is the entire mental model. Three areas, snapshots linked by parents, all stored in `.git`, with your identity stamped on each commit. In the next lesson you will actually move work through those three areas with `add` and `commit`.

Dig deeper

- Pro Git — Getting Started (Git Basics)
- Pro Git — Recording Changes to the Repository
- Atlassian — What is version control?
- Official git config documentation

Search terms

- `what is git version control for beginners`
- `git working tree staging area explained`
- `why git stores snapshots not diffs`
- `git init and git config first time setup`
- `what is inside the .git directory`

Check yourself

1. Name the three areas a change passes through and what each one is for.
2. What is actually stored in a single commit — a snapshot or a list of changed lines?
3. What happens to your project history if you delete the `.git` folder?
4. Which command turns an ordinary folder into a Git repository?
5. Why do you set `user.name` and `user.email`, and what does `--global` mean?

Revision #3
Created 2026-05-17 12:00:00 UTC by Eslam Shapsough
Updated 2026-05-17 12:04:00 UTC by Eslam Shapsough