Git Setup + Pehla Repo

Install Git, set your name and email once, create your first repo with git init, and walk the full status → add → commit → log journey.

Introduction

Before Git can remember your project's history, it needs two things: to be **installed**, and to know **who you are** — because every commit is signed with a name and email. This lesson does the full setup once, properly, then builds your first real repository slowly: `init`, watch a file with `status`, stage it with `add`, save it with `commit`, and read the story back with `log`. Every later lesson in this series stands on these five commands. *(Roman Urdu: Pehle Git install karein, phir apna naam aur email aik dafa set karein — us ke baad har repo ka pehla safar bas yehi paanch commands hain.)*

Key points

- **Install once, from the official source:** Windows and macOS get Git from git-scm.com; most Linux systems use their package manager. `git --version` is the proof the install worked.
- **Config is a one-time signature:** `git config --global user.name` and `user.email` stamp every commit you ever make on this machine. Use the same email as your GitHub account, or your commits will not link to your profile.

  (Roman Urdu: Naam aur email aik dafa set hota hai — phir har commit par aap ke dastakhat khud lag jate hain.)
- **`git init` creates exactly one thing:** a hidden `.git` folder inside your project. Delete that folder and the history is gone (the files stay). Nothing else about your folder changes.
- **`git status` is your dashboard:** it always tells the truth — which files are untracked, which are modified, which are staged — and it even hints the command you probably want next. When in doubt, run `git status`.
- **The three areas:** working folder (what you see) → staging area (what the next commit will contain, filled by `git add`) → repository history (filled by `git commit`). `git log` reads that history back, newest first.

Code examples

Example 1 — Install check + one-time identity

```
$ git --version
git version 2.45.1.windows.1
$ git config --global user.name "Ayesha Khan"
$ git config --global user.email "ayesha@example.com"
$ git config --global --list
user.name=Ayesha Khan
user.email=ayesha@example.com
```

What it does: `git --version` proves Git is installed and shows its version. The two `config` commands save your identity globally (for every repo on this machine), and `--list` reads it back so you can be sure there is no typo — a wrong email is the number-one reason commits appear as a stranger on GitHub.

What you will see: the version line (yours may end in `.windows.1`, `.macOS` style text, or a plain number on Linux — all fine), then your two settings printed back exactly as typed. No output at all from the two middle commands: in Git, silence means success.

Example 2 — init + status: meeting an untracked file

```
$ mkdir notes-app && cd notes-app
$ git init
Initialized empty Git repository in C:/projects/notes-app/.git/
$ echo "# My Notes App" > README.md
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
```

What it does: `git init` starts the repo; a new file `README.md` appears; `git status` reports it as **Untracked** — Git can see it, but is not following it yet, and Git itself suggests the cure: `git add`.

What you will see: the phrases "No commits yet" and "Untracked files" — this exact screen greets every brand-new repo. Notice Git speaks in instructions: nearly every status output tells you the next command.

Example 3 — add → commit → log: the full journey

```
$ git add README.md
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

$ git commit -m "Pehla commit: README jora"
[main (root-commit) 7c8d9e0] Pehla commit: README jora
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
$ git log --oneline
7c8d9e0 (HEAD -> main) Pehla commit: README jora
```

What it does: `git add` moves the file into "Changes to be committed" (the staging area). `git commit` saves it as the root commit — the very first checkpoint. `git log --oneline` reads the history in compact form: short ID, a pointer (`HEAD -> main`) showing where you are, and your message.

What you will see: after the commit, `git status` would say "nothing to commit, working tree clean" — the cleanest state a repo can be in. Edit the file, run `git status` again, and it reappears under "Changes not staged for commit" — the loop has begun.

Quick recap

- Setup is once per machine: install Git, `git --version` to prove it, `git config --global` for name + email (GitHub's email!).
- `git init` → the `.git` folder → history lives there. `git status` is the dashboard you will run a hundred times a day.
- The journey of every change: **working folder** → **`git add`** → **staging** → **`git commit`** → **history**, and `git log` reads the story back.

Common mistakes

1. **Running `git init` in the wrong folder** — in your home folder or inside another repo. The fix: `pwd` (or look at your terminal path) before init, and if `git status` already works in a folder, it already has a repo — do not nest another inside it.
2. **A typo in the config email** — commits then show on GitHub as an anonymous author, unlinked to you. The fix: `git config --global --list` to check, and re-run the config command with the correct email; it overwrites cleanly.

3. **Forgetting `git add` before commit** — Git answers "nothing added to commit" and the new file stays untracked. The fix: build the habit status → add → status → commit; the second status confirms exactly what is about to be saved.

Practice

1. Do the full setup on your own machine: install Git, set your real name and the email of your GitHub account, and print `git config --global --list` to verify both lines.
2. Create a repo `my-subjects`, add three small text files (one per subject you study), but stage and commit them in TWO commits: first one file alone, then the other two together. Read `git log --oneline` — you should see a two-chapter story, newest on top.

Practice quiz

1. What exactly does git init create?

- A GitHub repository online
- One hidden .git folder inside your project; delete it and the history is gone, the files stay
- A README file
- A remote called origin

Answer: One hidden .git folder inside your project; delete it and the history is gone, the files stay

2. Why set git config --global user.name and user.email, and which email?

- For fun, any email works
- They stamp every commit you ever make on this machine; use the same email as your GitHub account, or commits will not link to your profile
- They are required only once a year
- They set your GitHub password

Answer: They stamp every commit you ever make on this machine; use the same email as your GitHub account, or commits will not link to your profile

3. What is git status for?

- It pushes your code
- It is your dashboard: it tells the truth about untracked, modified and staged files, and even hints the command you probably want next
- It deletes untracked files
- It shows only the last commit

Answer: It is your dashboard: it tells the truth about untracked, modified and staged files, and even hints the command you probably want next

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/git-setup-first-repo · More free tools and notes at https://scopetreasure.com