Lesson: Git & GitHub from Zero to First Repo

Prerequisite
Have access to a computer and an internet connection.
That's it—every other tool you need will be installed step-by-step in this lesson.

Learning Goals

By the end, you will be able to

1. Install Git on **Windows** or **macOS/Linux**.
2. Create a GitHub account & Personal Access Token (PAT).
3. Configure Git with your identity and store credentials securely.
4. Clone an existing repository to your computer.
5. Create a brand-new repo, add a file, commit, and push it to GitHub.
6. Pull changes and resolve simple merge conflicts.
7. Explain (in plain English) what each Git command in your workflow does.

Check your understanding with the 10-question quiz at the end of Part 2.
You'll need **80 %** to unlock the Portfolio Workshop.

Pre-reading: The Big Picture (complete *before* you install Git)

Before diving into Git or terminals, take 30 minutes to read these beginner-friendly guides. Each one introduces **what GitHub is**, **how to use a terminal**, and **why it all matters**—even if you're brand new to tech.

Why you're reading	Title & Link	What you'll learn
1 — What *is* GitHub?	"Hello World" Quick Start — GitHub Docs	A hands-on walkthrough of GitHub's interface, how to create your first repository, and what a commit or pull request looks like in plain English.
2 — Terminal 101	"Command Line for Beginners" — freeCodeCamp	Learn what a terminal is, why developers use it, and how to navigate folders and files with commands like `cd`, `ls`, and `mkdir`.

Optional but eye-opening
"Why Git is the most important thing you need to learn as a Student" — medium.com
This article explains how GitHub helps students in *any* major build a digital portfolio, collaborate on real-world projects, and become job-ready early.

Your task:
Read all required articles before starting **Install Git** below.
Optional reading is encouraged if you're curious about long-term value.

Install Git

Before we can explore branches, commits, or portfolios, we need **Git**—the free tool that remembers every version of your files.
Think of Git as **"Track Changes on steroids."** Once it's on your computer, everything else in this module becomes a click-along exercise rather than abstract theory.

Good news: installation is usually a one-time, 5-minute job.
If you get stuck, post a screenshot in the Canvas **#git-help** thread and tag a TA; we'll unblock you quickly.

What *is* Git, anyway? *(skip if you're eager to install)*

- A **version-control system**: it stores snapshots of your project so you can rewind or compare any point in time.
- A **collaboration diary**: when you work with partners, Git shows who changed what (and when), preventing accidental overwrites.
- A **requirement for this course**: your assignments will be graded by cloning (downloading) your repo, so Git has to live on your laptop.

Choose your path

Your computer	Follow these steps
macOS or Linux	Path A — macOS & Linux
Windows 10/11	Path B — Windows

(If you prefer a graphical approach, you can skip ahead to GitHub Desktop, but we still recommend learning the basics of the command line first.)

Path A — macOS & Linux

You'll open something called a **terminal**. Picture it as *texting* your computer rather than clicking icons.
Every command appears in a gray box you can **copy-paste**. Nothing here can break your machine.

Open the Terminal

OS	How to open
macOS	Space → type **Terminal** → Return
Linux	Ctrl Alt T *or* search "Terminal" in your apps menu

A blinking prompt such as `jane@MacBook ~ %` (mac) or `jane@ubuntu:~$` (linux) means you're in.

Meet the Everyday Commands

Command	Plain-English meaning	Example	Why you need it
`pwd`	"Where am I?" (Print Working Directory)	`pwd` → `/Users/jane`	Prevents getting lost.
`ls`	"Show me what's here." (List)	`ls` → `Documents` `Downloads`	Like opening Finder/Files.
`ls -l`	Same, but with sizes & dates	`ls -l`	Helps spot newest file.
`ls -a`	Same, but include hidden files	`ls -a`	Lets you see `.git`, `.Rproj`, etc.
`cd folder`	"Go into that folder." (Change Directory)	`cd Documents`	Navigate folders.
`cd ..`	"Go up one level."	`cd ..`	Backtrack if you dove too deep.
`cd ~`	"Jump home."	`cd ~`	Quick reset to your base folder.
`cd -`	"Back to where I was."	`cd -`	Toggles between two folders.
`mkdir`	"Make a new folder." (Make Directory)	`mkdir ds101`	Organize assignments.
`touch`	"Create an empty file."	`touch notes.txt`	Quick placeholder.
`cat`	"Show the whole file here."	`cat notes.txt`	Peek at small text files.
`less`	"Open file in scrollable view."	`less README.md` (press **Q** to quit)	Browse large files safely.
`cp`	"Copy a file."	`cp a.txt backup/a.txt`	Keep backups.
`mv`	"Move or rename a file."	`mv draft.txt final.txt`	Rename files.
`rm`	"Delete a file." (no undo)	`rm trash.txt`	Clean up clutter.
`clear`	"Wipe the screen."	`clear`	Removes scroll clutter.
`exit`	"Close the terminal."	`exit`	Ends session.

Bookmark this table—you'll use these verbs all term.

Quick Navigation Walk-Through

Try these in your terminal now:

```
pwd                     # where am I?
ls                      # what's here?
mkdir ds101-sandbox     # create a folder
cd ds101-sandbox        # go inside it
touch hello.txt         # make a file
ls                      # confirm it's there
```

You just navigated, made a folder, and created a file *without* using the mouse.

Do I already have Git?

```
git --version
```

If you see something like `git version 2.43.0`*, you're done—skip to "Verify."*
If you get "command not found," continue to **Install Git**.

Install Git

macOS (option 1 – easiest)

```
xcode-select --install              # pops up "Command Line Tools"
```

macOS (option 2 – Homebrew)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
brew install git
```

Ubuntu / Debian

```
sudo apt update
sudo apt install git
```

Fedora / RHEL

```
sudo dnf install git          # Fedora
sudo yum install git          # RHEL 8-
```

What does `sudo` mean?
"Super-User DO"—temporarily grants admin rights (like Windows "Run as administrator").
You'll type your password; it won't show on screen—that's normal.
Use `sudo` **only** when a trusted guide (like this lesson) tells you to.

Verify the Install

```
git --version
```

Seeing a version number (e.g., `2.44.0`) means success.
If not, re-check the install steps or post a screenshot in **#git-help**.

Path B — Windows 10 / 11

Goal: install Git, learn the *Git Bash* terminal, and understand why Windows shows paths like `C:\Users\You\Desktop` while Git commands prefer `C:/Users/You/Desktop`.

Download Git for Windows

1. Open https://git-scm.com/download/win.

2. The download starts automatically (a file named `Git-<version>-64-bit.exe`).

Run the Installer — What to Click

Screen	Recommended choice	Why
Select Components	Leave defaults	Git Bash, Git GUI, and context-menu entries are handy.
Choosing the default editor	Nano (default) or VS Code	Pick the one you're comfortable with.
Adjusting your PATH	"Git from the command line and 3rd-party software"	Lets tools like VS Code find Git.
Configuring line endings	"Checkout Windows-style, commit Unix-style (CRLF → LF)"	Prevents stray `^M` characters when collaborating with Mac/Linux users.
The rest	Default	Safe to accept.

Click **Install** → **Finish**.
A shortcut called **Git Bash** now lives in your Start menu.

Launch Git Bash

Start *Git Bash* → Enter.
You'll see something like:

```
you@DESKTOP MINGW64 ~
$
```

That `$` is your prompt.

File-Path Cheat Sheet — Backslash (\) vs Forward (/)

Where you are	Path looks like	Example
File Explorer / PowerShell	**Backslashes** \	`C:\Users\You\Desktop`
Git Bash / Linux / macOS	**Forward slashes** /	`C:/Users/You/Desktop`

Git Bash understands both, but its own output always shows **forward slashes**.
So when this guide says:

```
cd C:/Users/You/Documents
```

you **could** type:

```
cd C:\Users\You\Documents
```

and it would still work.

> **Pro tip:** If the path has spaces, wrap it in quotes:
> `cd "C:/Users/You/My Projects/ds101"`.

Basic Navigation Commands in Git Bash

Command	What it does	Windows-Explorer analogy
`pwd`	Shows the current folder	Address bar
`ls`	Lists files/folders	File pane
`cd Desktop`	Enters Desktop	Double-click Desktop
`cd ..`	Goes "up" one folder	Back arrow
`mkdir ds101`	Makes a new folder	*New Folder*

> **Heads-up:** Windows' built-in `dir`, `copy`, etc. still exist in *PowerShell* or *Command Prompt*,
> but inside **Git Bash** you'll use the Unix-style commands above.

Verify Git Installed Correctly

```
git --version
```

You should see something like `git version 2.44.0.windows.1`.
If Bash says "command not found," restart your PC and try again.

Optional: Windows Terminal + Git Bash Profile

If you prefer a tabbed terminal, install **Windows Terminal** from the Microsoft Store.
It auto-detects Git Bash, letting you switch between PowerShell, CMD, and Bash tabs.

GitHub Desktop GUI *(optional)*

Still nervous about commands? Download **GitHub Desktop** (https://desktop.github.com/).
It provides buttons for **Clone, Commit, Push, Pull** while sharing the same Git history.
We'll demonstrate both Desktop *and* terminal workflows in class.

You did it

Git now lives quietly on your machine. Keep the terminal (or GitHub Desktop) open—we'll configure your name and connect to GitHub in **Section 2 — Set Up Your GitHub Account**.

> **Checkpoint:** Does `git --version` print a number? If yes, you're set.
> Still stuck? Post a screenshot in **#git-help**; a TA will respond within 24 h.

Create / Confirm Your GitHub Account

Why this matters Your GitHub profile becomes your public résumé of code & data projects.
Employers and research mentors *will* Google it—so let's set it up right the first time.

GUI Walk-through using the Web Browser

1. **Open** https://github.com in Chrome, Firefox, or Edge.
2. Click **Sign up** (top-right).
3. **Email address** — use a **personal** account you'll still own after graduation.
4. **Password** — pick something strong (password manager recommended).
5. **Username** — choose something professional (e.g., `jane-doe`, not `gamer420`).
6. You'll be sent a **verification email**. Click the link inside to activate your account.
7. Optional—but recommended:
   - Add a friendly **profile photo** (headshot or avatar, nothing NSFW).
   - Fill in **Name**, **Bio** ("First-year DS student at OSU"), and **Location**.

Configure Git on Your Computer

You're telling Git, "Whenever I save a snapshot, tag it with *this* name & email, use *this* editor, call the branch *main*, and remember my PAT so you don't nag me."
You run these **once per machine** (or whenever you want to change a setting).

What are these tables?
Each table below is a "cheat-sheet."
* **Command** – the exact text you type (copy-paste is fine).
* **Plain-English meaning** – what you should hear in your head when you run it.
* **Example** – a realistic usage with placeholders filled in.
* **Why you need it** – the practical reason first-year DS students will care.
Skim the **meaning** column first; if it sounds useful, glance right for the example

and try it in your own terminal. Replace anything in *italics* or `<angle-brackets>` with your info.

Command	Plain-English meaning	Example	Why you need it
`git --version`	"Do I have Git, and which one?"	`git --version` → `git version 2.44.0`	Quick sanity check after install.
`git config --global user.name "Jane Doe"`	"Stamp my work with this name."	`git config --global user.name "Brian C. Alvarez"`	Commits show up under your real name on GitHub.
`git config --global user.email "you@example.com"`	"...and this email."	`git config --global user.email "brian@example.com"`	Matches commits to your GitHub account for contribution graphs.
`git config --global credential.helper store`	"Remember my PAT so I type it once."	*(run once, then push)*	Avoids re-entering token each push.
`git config --global init.defaultBranch main`	"Call new branches *main* instead of *master*."	`git init` then `git branch` → `main`	Keeps local + GitHub branch names consistent.
`git config --global core.editor "nano -w"`	"Open Nano when Git needs me to edit."	set VS Code instead: `git config --global core.editor "code --wait"`	Pick an editor you're comfortable with for merge messages.
`git config --global color.ui auto`	"Color-code Git's output."	*(run once)*	Highlights adds (green) & deletes (red) in `git diff`.
`git config --list --show-origin`	"Show every setting & where it lives."	`git config --list --show-origin`	Troubleshoot if something seems off.

Command	Plain-English meaning	Example	Why you need it
`git config --edit --global`	"Open the config file in my editor."	`git config --edit --global`	Manual tweaks without memorizing commands.

TIP: The `--global` flag writes to a single file (`~/.gitconfig` on mac/Linux, `%USERPROFILE%\.gitconfig` on Windows). Remove `--global` if you ever need a per-project override.

GitHub-specific Remote Commands *(you'll use these in Parts 2 & 4)*

Read this table like a recipe card:
You'll run these whenever you want to *connect* your local folder to the cloud, *sync* changes, or *grab* someone else's work. Replace `URL` with the HTTPS link you copy from GitHub (it always ends in `.git`).

Command	Plain-English meaning	Example	Why you need it
`git remote add origin URL`	"Bookmark my repo on GitHub."	`git remote add origin https://github.com/you/ds101.git`	Links **local** folder **remote** repo.
`git remote -v`	"Show which remotes I have."	`git remote -v`	Sanity check before pushing.
`git push -u origin main`	"Upload commits and set *origin/main* as default."	*(after first commit)*	The `-u` flag means future `git push` can omit branch name.
`git pull origin main`	"Grab teammates' latest work."	`git pull origin main`	Always pull before you start editing.
`git clone URL`	"Copy the whole project (and its history)."	`git clone https://github.com/octocat/Hello-World.git`	First step when joining an existing repo.

Keep this table handy; these six commands make up **90 %** of your day-to-day Git-for-GitHub life.

Option #1: Using GitHub Desktop

1. Launch **GitHub Desktop**.

2. Menu **File Options...** (**GitHub Desktop Preferences...** on macOS).

3. In **Git** tab, enter
    - **Name:** the exact name you want on commits (e.g., *Jane Doe*).
    - **Email:** the **same** personal email you used on GitHub.com.
4. Click **Save**.

Option #2 Using RStudio

1. **Tools** → **Global Options** → **Git/SVN** (or **Version Control** on newer builds).
2. Fill in **Name** and **Email** → **Apply** → **OK**.

Terminal Method (works everywhere)

```
git config --global user.name  "Jane Doe"
git config --global user.email "jane.doe@example.com"
```

`--global` = use these values for **all** future repositories on this computer.

Generate & Store a Personal Access Token (PAT)

> GitHub no longer accepts account passwords for pushing code;
> a **PAT** acts like a long, single-purpose password just for Git.

GUI Path (Recommended)

1. On GitHub.com: click your **profile picture** → **Settings**.
2. Sidebar **Developer settings** → **Personal access tokens** → **Tokens (classic)**.
3. Press **Generate new token (classic)**.
4. Fill the form:
    - **Note:** `ds101-pat`
    - **Expiration:** 90 days is fine (you can always regenerate).

- **Scopes:** tick **repo** (gives push/pull rights).

5. Click **Generate token** → copy the long string that appears (starts with `ghp_`).

6. **Store it once** when your tool asks:

   - **GitHub Desktop:** it prompts automatically on first push—paste token as *password*.

   - **RStudio:** run `gitcreds_set()` (see below) and paste when prompted.

   - **Windows Credential Manager / macOS Keychain:** Git stores it after your first successful push, so you're not asked again.

Terminal Storage (universal fallback)

```
git config --global credential.helper store          # remember credentials
# The next time Git asks for "password", paste the PAT.
```

R code helper (for RStudio fans)

```
install.packages("usethis")
library(usethis)
gitcreds_set()   # Paste PAT when prompted; it's written to an encrypted store
```

> **Never** put your PAT inside a repository, email, or chat post. Treat it like a credit-card number.

Clone Your First Repository

We'll practice with GitHub's sample repo **"Hello-World."**

Option #1: Using GitHub Desktop

1. Open **GitHub Desktop** → **File** **Clone repository...**

2. In the **URL** tab, paste

   `https://github.com/octocat/Hello-World.git`

3. Choose any **local folder** (e.g., `Documents\ds101`) → **Clone**.

4. Once cloned, click **Open in VS Code** or **Open in Finder/Explorer** to peek at the files.

Option #2: Using RStudio

1. **File New Project Version Control Git**.

2. Paste the same URL.

3. Pick a local directory → **Create Project**.

4. RStudio reloads with a new project. The **Git** tab lists all version-controlled files.

Option #3: Terminal Method

```
cd ~                                    # or any workspace directory
git clone https://github.com/octocat/Hello-World.git
cd Hello-World
```

What just happened?

Step	Meaning
clone	Copied the entire project—including **every past version**—to your computer.
`.git` folder	A hidden directory where Git keeps its history; deleting it would "un-version" the project.
cd Hello-World	"Change Directory" so future commands run *inside* the project.

Checkpoint: Open the newly cloned folder; you should see `README.md`, `index.html`, etc. If so, you've successfully installed Git, configured your identity, authenticated with GitHub, and cloned your first repo—great work!

Next up: making your *own* repository and pushing changes.

The Core Workflow — *Change → Stage → Commit → Push*

You make a change, tell Git you're ready, write a short diary note, then send it to the cloud.
Same idea, two ways: **GUI first**, **terminal second**.

GUI (GitHub Desktop)

1. **Open** the *Hello-World* repo in **GitHub Desktop**.
2. In your file explorer (Finder / Explorer) open `README.md`, add this line at the bottom:

   `Hello DS 101!`
3. **Save** the file; GitHub Desktop now shows the change in the *Changes* tab.
4. Type a **summary**: `Add friendly greeting` → click **Commit to main**.
5. Click **Push origin** (upper-right).
6. Wait for the green check that says **"Published"**.

6.2 Terminal

```
echo "Hello DS 101!  " >> README.md   # add a line
git status                            # Git sees it as 'modified'
git add README.md                     # stage the change
git commit -m "Add friendly greeting" # snapshot saved locally
git push origin main                  # upload to GitHub
```

If Git prompts, **username = your GitHub handle**, **password = PAT**.

Create Your Repository (Local Remote)

GUI — One-click Web Creation & Desktop Clone

1. On GitHub.com click **New repository**.
2. **Name:** `ds101-playground` · **Public** · *Add a README* → **Create repository**.
3. In the new repo page click **Code** **Open with GitHub Desktop**.
4. Pick a folder (e.g., `Documents/ds101`) → **Clone**.
5. **Open** `README.md`, change the heading to `# My DS 101 Playground`, **Save**.
6. Back in GitHub Desktop → **Commit** `Update title` → **Push origin**.

Your repo is live at `https://github.com/<you>/ds101-playground`.

Terminal — Manual Init & First Push

```
cd ~
mkdir ds101-playground && cd ds101-playground
echo "# My DS 101 Playground" > README.md
git init
git add README.md
git commit -m "First commit  "
git branch -M main
git remote add origin https://github.com/<you>/ds101-playground.git
git push -u origin main
```

Keep in Sync — *Pull → Edit → Push*

GUI

1. **Fetch origin** (top-left) before you start editing.
2. If changes exist, click **Pull**.
3. Edit files → **Commit** → **Push**.

Terminal

```
git pull origin main          # grab teammates' latest work
# make edits...
git add <file>
git commit -m "Describe change"
git push                      # send your commits
```

> **Conflict?** GitHub Desktop shows red "Conflict" badges; click each file, pick **Use mine**, **Use theirs**, or **Merge**. In terminal you'll open the file, delete the `<<<<<<< HEAD … >>>>>>>`, then `add` + `commit`.

(Optional) GitHub Desktop Cheat-Sheet

Action	Button / Menu
Clone repo	**File** **Clone repository…**
Stage file	Check the box beside the file
Commit	Type message → **Commit**
Push / Pull	**Push origin** / **Fetch origin**
Switch branch	Top-center dropdown

Remember: anything you do here can be done later in terminal with the same Git history.

Common Pitfalls & Quick Fixes

Symptom	Likely Reason	Fix
Re-enter PAT every push	Helper not set	`git config --global credential.helper store` and push once more

Symptom	Likely Reason	Fix
`fatal: not a git repository`	You're outside the project folder	`cd` into the folder that contains a hidden `.git`
Merge conflict markers `<<<<<<<`	Two people edited the same line	Decide which text to keep → save → `add` → `commit`
`brew: command not found` (mac)	Homebrew missing	Install via https://brew.sh
`git: command not found` (Linux)	Git not installed	`sudo apt-get install git` (Debian) / `sudo dnf install git` (Fedora)

Ready for the Quiz

Great job! Take the **10-question quiz** to lock in these ideas.
Score **80 %** and Part 3 (*Portfolio Workshop*) unlocks automatically.

Where to Next?

- **Part 3 – Portfolio Workshop** — convert your repo into a live Quarto site.
- **Part 4 – Assignments** — solo and team practice, plus extra credit for polishing your portfolio.

Questions? Post in the Canvas discussion **"GitHub Help Thread."**
A TA will respond within 24 hours (often sooner).