# Linux Basics and Git Workflow

DevOps Training – Day 3

Welcome students to Day 3. Today we transition from abstract concepts to actual tools. Linux and Git are the bread and butter of DevOps.

# SESSION OBJECTIVES

- Understand Linux terminal basics and CLI philosophy
- Learn essential file and directory manipulation commands
- Practice editing configuration files using the `nano` editor
- Grasp the core concepts of Version Control Systems (VCS)
- Build a complete workflow: Local Git Bash to GitHub Cloud

Briefly outline the roadmap. Emphasis: we are moving from "point-and-click" to "type-and-execute".

# WHAT IS LINUX?

### A Simple Definition

Linux is an open-source Operating System (OS). It is the kernel that powers everything from Android phones to the world's fastest supercomputers.

### Why for DevOps?

Most servers (AWS, Azure, Google Cloud) run on Linux. It is stable, secure, and highly efficient for automation and scripting.

*"Linux isn't just an OS; it's the foundation of the Cloud."*

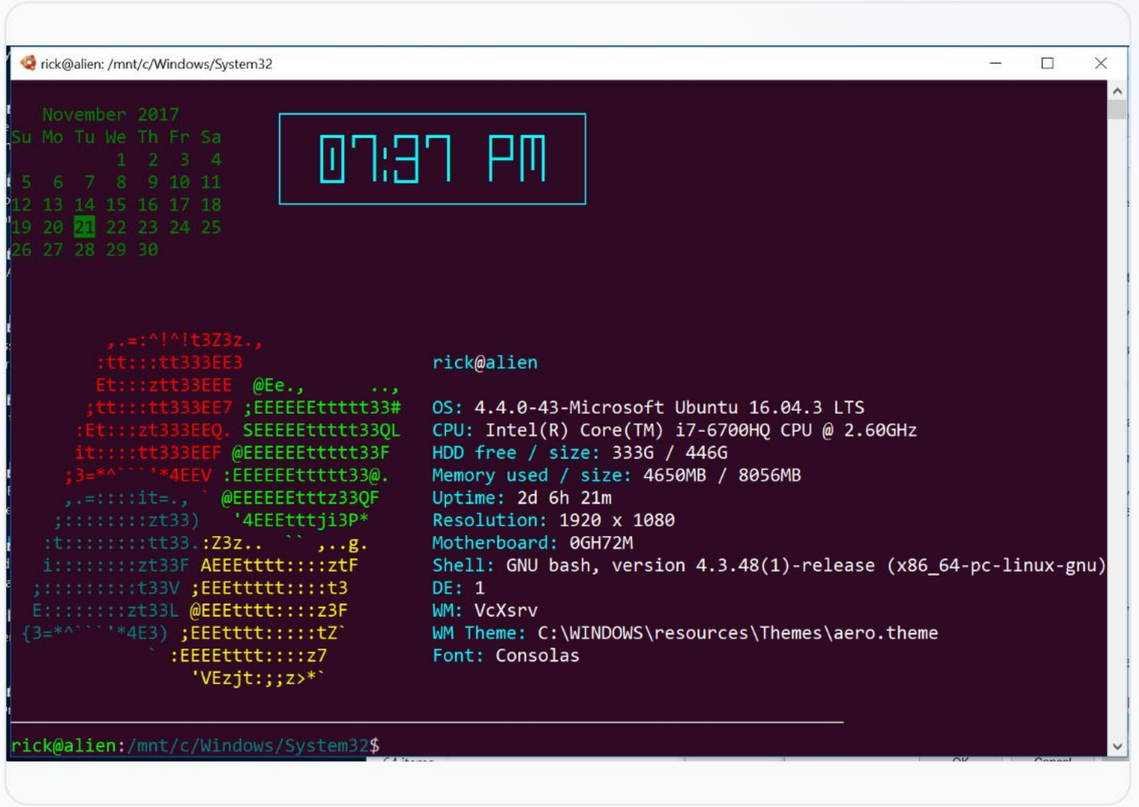
Explain that Linux is everywhere. Compare it to Windows: Windows is user-friendly for desktops, Linux is power-friendly for servers.

# THE TERMINAL (CLI)

### Command Line Interface

Unlike a GUI (Graphical User Interface) where you click icons, the CLI allows you to talk to the OS directly using text commands.

```
username@machine:~$ command -options
```




Explain the anatomy of the prompt: Who you are, where you are, and the '$' which means you are a regular user.

# BASIC NAVIGATION

### pwd

Print Working Directory. Tells you exactly where you are in the folder system.

### ls

List files. Shows everything inside your current folder.

### cd [dir]

Change Directory. Move into a folder or use `cd ..` to go back.

These are the "walking" commands. If you get lost in the terminal, 'pwd' is your GPS.

# MANAGING FILES

### mkdir / touch

`mkdir` creates folders.
`touch` creates empty files.

### cat / echo

`cat` reads file content.
`echo` prints text to the screen.

### rm / rmdir

`rm` removes files. Careful: Linux has no "Recycle Bin" for terminal deletes!

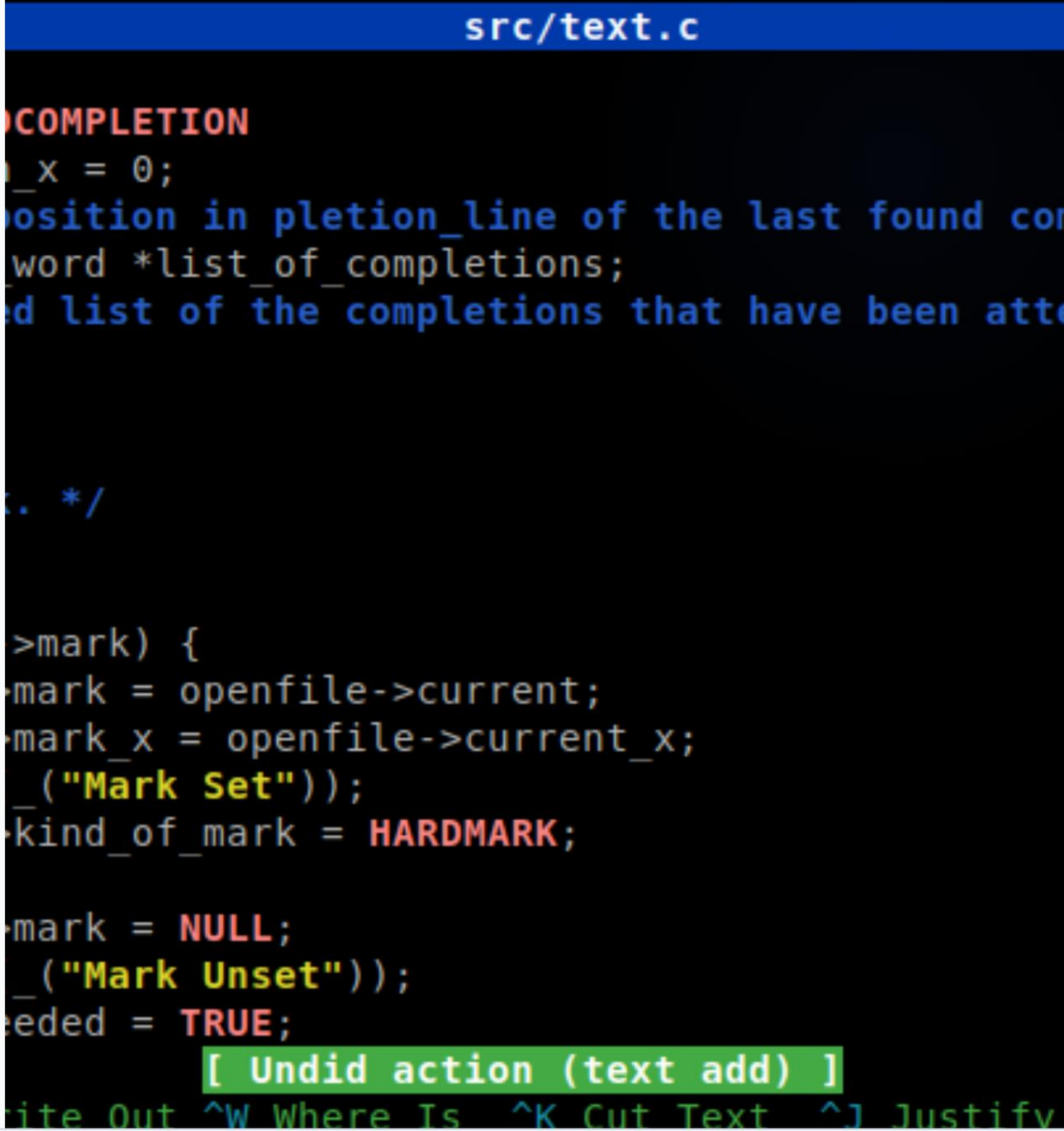
Explain the workflow: Create a folder, enter it, create a file, then read it. Remind them 'rm' is permanent.

# EDITING WITH NANO

Nano is a simple, beginner-friendly text editor built into the terminal.

- Open: `nano filename.txt`
- Save: **CTRL + O** (Write Out)
- Exit: **CTRL + X**

Note: Use keyboard only; no mouse interaction!




Students often get stuck in editors. Tell them the shortcuts are listed at the bottom of the nano screen. '^' means CTRL.

# FILE PERMISSIONS

**755**

**STANDARD PERMISSIONS**

**chmod Concept**

In Linux, every file has permissions for **Owner**, **Group**, and **Others**.

- **r**: Read (4)
- **w**: Write (2)
- **x**: Execute (1)

Command: `chmod 777 script.sh` (Full access for everyone)

Don't go too deep into math. Just explain that 'ls -l' shows a string like '-rwxr-xr-x' which defines who can do what.

# PRACTICE LAB 01

Perform the following tasks in your terminal:

| Step | Action | Command |
| --- | --- | --- |
| 1 | Check current location | `pwd` |
| 2 | Create a folder named 'devops' | `mkdir devops` |
| 3 | Enter the folder | `cd devops` |
| 4 | Create a file and add your name | `nano student.txt` |

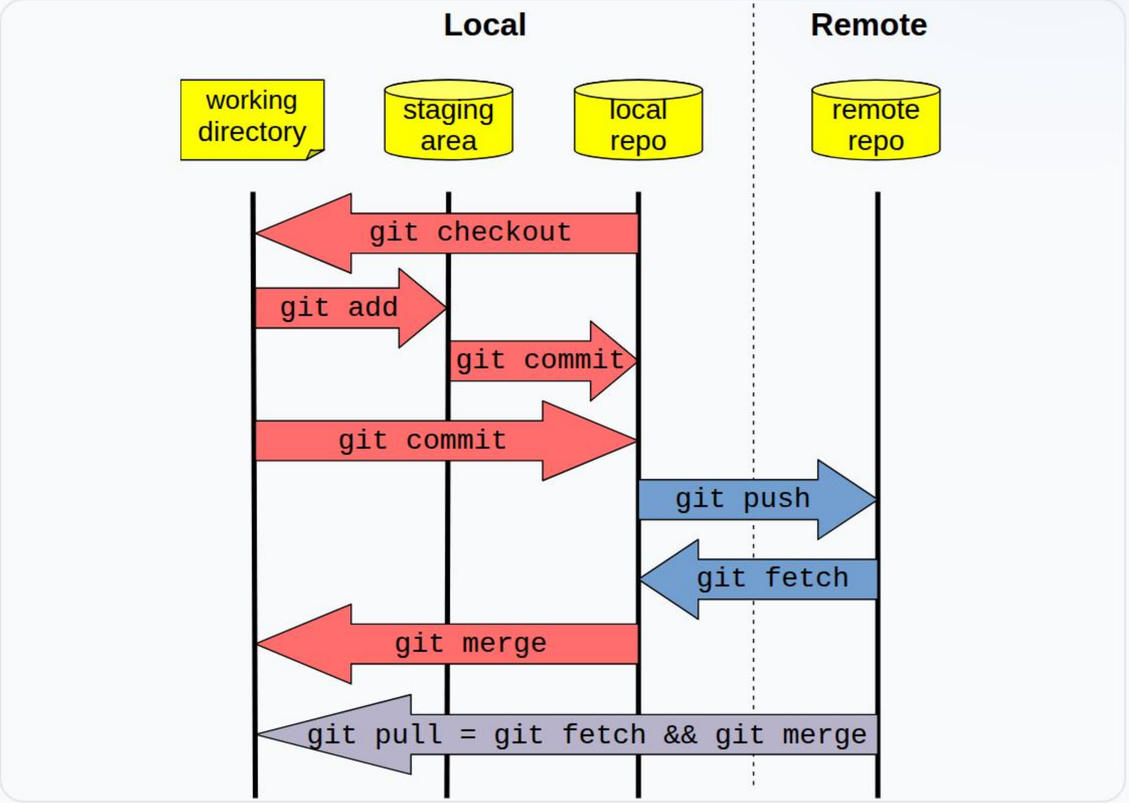
Give them 5 minutes to do this. Walk around and help those struggling with nano or 'cd'.

# INTRODUCTION TO GIT

### Version Control

Git is a time machine for your code. It tracks every change, allowing you to go back to previous versions if things break.

- Track changes locally
- Collaborate with others
- Backup code on the Cloud




Ask them if they've ever named files 'final_v1.zip', 'final_v2_new.zip'. Git solves that mess.

# THE GIT WORKFLOW

1

**Working**

Writing your code in local files.

2

**Staging**

Preparing changes for a commit.

3

**Commit**

Saving a snapshot to local history.

4

**GitHub**

Pushing to the cloud server.

This is the most important slide. Use the 'shopping' analogy: Working = Picking items, Staging = Putting in cart, Commit = Paying/Receipt.

# ESSENTIAL GIT COMMANDS

**git init**

Initialize a new local repository in your current folder.

**git status**

See which files are staged, unstaged, or untracked.

**git add .**

Add all changed files to the staging area.

**git commit -m**

Save changes with a message: `"Added header"`

Explain that 'git status' is your best friend. Tell them to run it after every command to see what happened.

# GITHUB: SETUP

- Login to **github.com**
- Click the **'+'** icon → **New Repository**
- Give it a name (e.g., `my-first-git`)
- Keep it **Public**
- Leave 'README' unchecked for now
- Click **Create Repository**


💬 Wait for everyone to reach the screen with the code snippets after creating the repo. That's the crucial point.

# CONNECTING TO GITHUB

```
# Add the remote address

 git remote add origin [URL]

# Set default branch name

 git branch -M main
```

**What is "origin"?**

It's just an alias for your repository's long URL. It tells Git where to send your files when you are ready to "push".

URL Example: `https://github.com/user/repo.git`

Students often type '[URL]' literally. Remind them to copy the actual link from their browser.

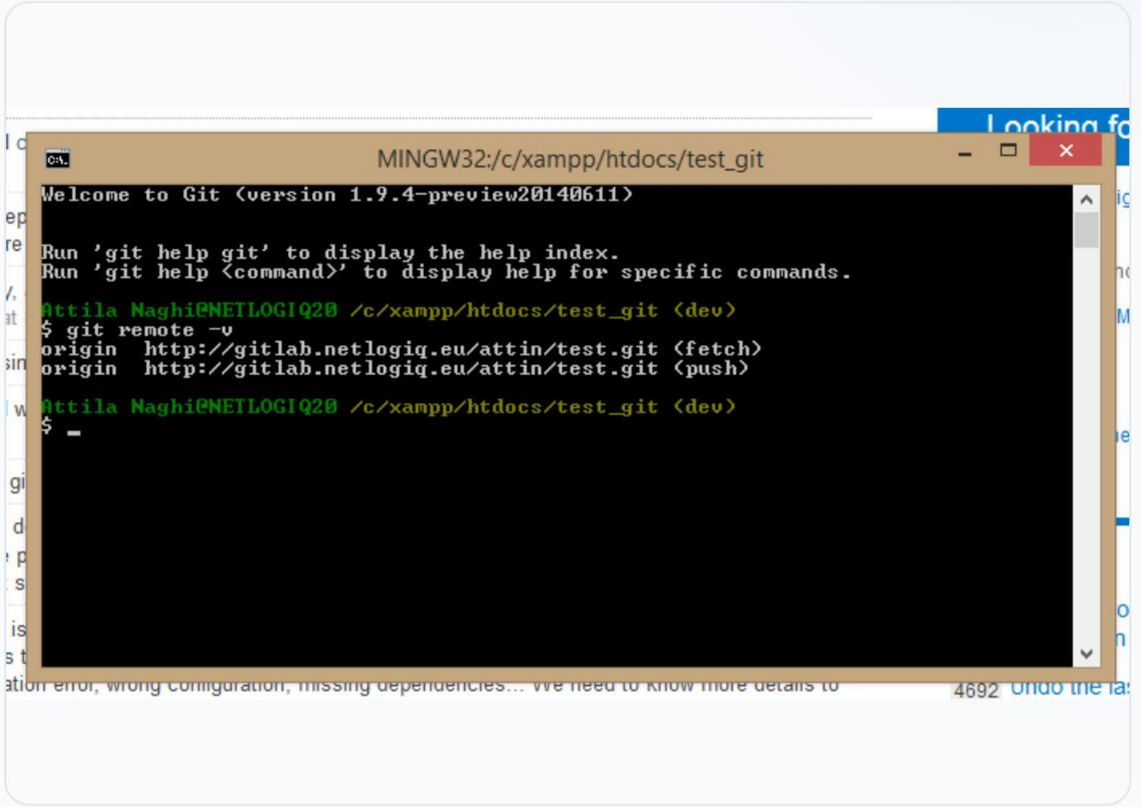
# THE FINAL PUSH

### Sending files to Cloud

```
git push -u origin main
```

The `-u` flag links your local branch to the remote one permanently.

> 🔑 **Authentication:** GitHub will ask you to login via browser or use a "Personal Access Token".




💬 Pushing might fail due to authentication. Mention the GitHub Desktop or Browser login pop-up.

# WATCH OUT! COMMON MISTAKES

### HTML in Terminal

Don't try to write `<>` tags directly in the terminal bash. Terminal is for commands, Nano is for code.

### Wrong Commit Syntax

Forgetting the `-m` or the quotes. Always use: `git commit -m "Your message"`

Mistake #3: Trying to push before they've even committed anything locally. 'git status' would have caught that!

# FINAL ACTIVITY

Complete this full workflow independently:

| Goal | Terminal Commands to use |
| --- | --- |
| Setup | `mkdir website && cd website` |
| Create Content | `nano index.html` (add "Hello DevOps") |
| Init Git | `git init` |
| Save Changes | `git add .` then `git commit -m "First HTML"` |
| Go Cloud | Create GitHub repo, `remote add`, and `push` |

This is their graded/verified task for the day. Ensure they show you the code on GitHub before leaving.

# RECAP: WHAT WE LEARNED

**Linux**

Navigating, creating files, and editing configurations using the terminal.

**Git**

The workflow: Add, Commit, and tracking version history locally.

**GitHub**

Pushing your local projects to the cloud for collaboration.

Ask them: "Why do we use Git?" (To track history). "Why Linux?" (Servers run on it).

# Q&A

Any hurdles with Linux or Git today?

**Next Session Preview:**

**DevOps Concepts & CI/CD Pipelines**

Automating the process from Git commit to Server deployment.

End with a high-energy preview. Tomorrow, we automate everything we did manually today.

# IMAGE SOURCES

https://i.sstatic.net/v1sMa.png

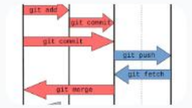
Source: askubuntu.com

---

https://www.nano-editor.org/nano-3.2.png

Source: www.nano-editor.org

---




https://homes.cs.washington.edu/~mernst/advice/version-control-ops-local-vs-remote.svg

Source: homes.cs.washington.edu

---

https://user-images.githubusercontent.com/78680563/228942935-6b80d2a7-d4c9-4930-8c3d-07f03d0c3d26.png

Source: github.com

---

https://i.sstatic.net/P5Zdn.png

Source: stackoverflow.com