Git & GitHub Workshop for New Interns

Welcome to Version Control 101!

🎯 Workshop Goals

By the end of this session, you'll be able to:

- Understand what Git and GitHub are and why we use them
- Create repositories and make commits
- Push and pull code to collaborate with your team
- Avoid common mistakes and fix them when they happen

---

1. Introduction to Version Control

What is Version Control?

Imagine you're writing an important essay. You save it as `essay_v1.doc`, then `essay_v2.doc`, `essay_final.doc`, `essay_final_FINAL.doc`, `essay_final_for_real_this_time.doc`... Sound familiar?

Version control is like having a smart time machine for your code. Instead of creating multiple copies, you have ONE file, but you can:

- See every change you've ever made
- Go back to any previous version
- Work with teammates without overwriting each other's work
- Understand WHO changed WHAT and WHEN

Why Teams Need It

Picture this: You and three teammates are building a website. Without version control:

- You email code files back and forth (messy!)
- Someone's changes get lost when another person saves
- You can't track who broke what
- Collaboration is a nightmare

With version control:

- Everyone works on the same project simultaneously
- All changes are tracked and attributed
- You can work on different features without conflicts
- If something breaks, you can instantly revert to the working version

Git vs. GitHub

Git = The tool (software on your computer) that tracks changes **GitHub** = The website (cloud platform) where code is stored and shared

Think of it like this:

- **Git** is like Microsoft Word (the program)
- **GitHub** is like Google Drive (where you store and share documents)

---

2. Understanding Git

How Git Works Behind the Scenes

Git creates invisible checkpoints (called **commits**) of your project. Each commit is like a snapshot of your entire project at that moment.

Key Concepts

Repository (Repo): The folder containing your project and all its history. Think of it as a magic folder that remembers everything.

Staging Area: A preparation zone before committing. It's like putting items in a shopping cart before checkout.

Commit History: The timeline of all changes made to your project. Each commit has a unique ID, message, author, and timestamp.

Branching: Creating parallel versions of your code to work on different features. Like creating alternate timelines where you can experiment safely.

---

3. GitHub Overview

What is GitHub?

GitHub is where developers:

- Store code in the cloud
- Collaborate with teammates
- Share projects with the world
- Review each other's code

Getting Started

1. Go to github.com and create a free account
2. Choose a professional username (employers will see this!)
3. Verify your email

Public vs. Private Repositories

Public Repos: Anyone can see your code (great for portfolios!) **Private Repos**: Only invited collaborators can access (for sensitive projects)

---

4. Repositories (Repos)

Creating Your First Repo on GitHub

1. Click the + icon in the top right → "New repository"
2. Name it (e.g., `my-first-project`)
3. Add a description (optional but recommended)
4. Choose Public or Private
5. Check "Add a README file" (creates a starting point)
6. Click "Create repository"

Cloning a Repo

Cloning means downloading a copy of a repository to your computer.

```bash
```

```
git clone https://github.com/username/repo-name.git
```

This creates a folder with all the project files and history.

Important Files

README.md: The front page of your repo. Explains what the project does.

.gitignore: Tells Git which files to ignore (passwords, temporary files, etc.)

---

5. Commits: Saving Your Progress

What is a Commit?

A commit is a saved checkpoint of your work. Each commit includes:

- All the changes you made
- A message describing what you did
- Your name and timestamp
- A unique ID (hash)

The Commit Workflow

Step 1: Check Status

```
bash

git status
```

Shows which files have changed (red = not staged)

Step 2: Stage Your Changes

```
bash

git add filename.txt        # Stage one file
git add .                   # Stage all changed files
```

Staging means "I want to include these changes in my next commit"

Step 3: Commit

```bash
git commit -m "Add login feature"
```

The `-m` flag adds a message describing what you did.

Step 4: View History

```bash
git log
```

Shows all commits with messages, authors, and dates.

Writing Good Commit Messages

❌ Bad: `"fixed stuff"`, `"update"`, `"asdf"` ✅ Good: `"Fix login button alignment"`, `"Add user authentication"`, `"Update README with setup instructions"`

Pro tip: Write messages that complete this sentence: "This commit will..."

---

6. Push and Pull: Syncing with GitHub

Understanding Push and Pull

git pull: Downloads the latest changes from GitHub to your computer **git push**: Uploads your local commits to GitHub

Think of GitHub as a shared Google Doc:

- **Pull** = Refresh to see others' updates
- **Push** = Save your changes so others can see them

The Workflow

Before you start working:

```bash
git pull origin main
```

Gets the latest version from GitHub's `main` branch.

After making commits:

```
bash

git push origin main
```

Uploads your commits to GitHub's `main` branch.

Handling Merge Conflicts

A **merge conflict** happens when:

1. You and a teammate both change the same line of code
2. You try to push without pulling first

Git will mark the conflict in your file:

```
<<<<<<< HEAD
Your version of the code
=======
Their version of the code
>>>>>>> main
```

To fix it:

1. Decide which version to keep (or combine both)
2. Delete the conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`)
3. Save the file
4. Stage and commit the resolved file
5. Push again

---

7. Hands-On Workshop Exercise

Task: Create Your First Repository

Step 1: Create a Local Repository

```
bash
```

```
mkdir intern-training
cd intern-training
git init
```

`git init` turns a regular folder into a Git repository.

Step 2: Create Your First File

```bash
echo "# My Training Notes" > notes.txt
```

Step 3: Make Your First Commit

```bash
git add notes.txt
git commit -m "Add initial notes file"
```

Step 4: Create a GitHub Repository

1. Go to GitHub.com
2. Click + → New repository
3. Name it `intern-training`
4. Don't initialize with README (we already have files)
5. Copy the repository URL

Step 5: Connect Local to Remote

```bash
git remote add origin https://github.com/YOUR-USERNAME/intern-training.git
git branch -M main
git push -u origin main
```

Step 6: Make Another Change

```bash
```

```
echo "Git is awesome!" >> notes.txt
git add notes.txt
git commit -m "Add enthusiastic comment about Git"
git push origin main
```

Refresh your GitHub page—your changes are live! 🎉

Bonus Exercise: Clone a Teammate's Repo

1. Ask a teammate for their repo URL
2. Clone it: `git clone [their-url]`
3. Explore their code
4. This simulates joining an existing project!

---

8. Collaboration Tips

Using Branches

Branches let you work on new features without affecting the main code.

Create a new branch:

```bash
git checkout -b feature-login
```

Work on your feature, commit changes

Switch back to main:

```bash
git checkout main
```

Merge your feature:

```bash
git merge feature-login
```

Pull Requests (PRs)

A Pull Request is a way to propose changes:

1. You work on a branch
2. Push the branch to GitHub
3. Open a PR asking teammates to review
4. After approval, your code gets merged into main

Why PRs matter: Code review catches bugs and shares knowledge!

---

9. Common Mistakes & How to Fix Them

Mistake 1: Forgot to Pull Before Pushing

Error: `! [rejected] main -> main (fetch first)`

Fix:

```bash
git pull origin main
# Resolve any conflicts
git push origin main
```

Prevention: Always pull before you push.

Mistake 2: Committed Sensitive Files (Passwords, API Keys)

Fix:

```bash
# Remove from Git but keep on computer
git rm --cached secrets.txt
echo "secrets.txt" >> .gitignore
git commit -m "Remove sensitive file"
git push origin main
```

Prevention: Create a `.gitignore` file before your first commit!

Mistake 3: Commit Message Typo

Fix:

```bash
git commit --amend -m "Corrected commit message"
```

Warning: Only amend commits that haven't been pushed yet!

Mistake 4: Need to Undo Last Commit

Keep changes but undo commit:

```bash
git reset --soft HEAD~1
```

Throw away commit and changes:

```bash
git reset --hard HEAD~1
```

⚠️ Be careful with `--hard`—you'll lose your work!

Git Commands Cheat Sheet

Setup

```bash
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

Creating Repositories

```bash
git init                    # Create new repo
git clone [url]             # Copy existing repo
```

Basic Workflow

```bash
```

```
git status                  # Check what's changed
git add [file]              # Stage specific file
git add .                   # Stage all changes
git commit -m "message"     # Save changes
git push origin main        # Upload to GitHub
git pull origin main        # Download from GitHub
```

Branching

```
bash

git branch                  # List branches
git branch [name]           # Create branch
git checkout [name]         # Switch to branch
git checkout -b [name]      # Create and switch
git merge [name]            # Merge branch into current
```

History

```
bash

git log                     # View commit history
git log --oneline           # Compact history
git diff                    # See unstaged changes
```

Practice Exercises

Exercise 1: Basic Workflow (15 minutes)

1. Create a new repository called `practice-repo`
2. Add a file called `about.txt` with your name and favorite hobby
3. Make three separate commits:
   - First commit: Add the file
   - Second commit: Add your favorite programming language
   - Third commit: Add today's date
4. Push all changes to GitHub

Exercise 2: Collaboration Simulation (20 minutes)

1. Pair up with another intern
2. Clone their `practice-repo`
3. Create a new file with your name (e.g., `john-was-here.txt`)
4. Commit and push (you'll need to be added as a collaborator first)
5. Have them pull your changes
6. Discuss what you learned!

Exercise 3: Branch Practice (15 minutes)

1. Create a branch called `experimental-feature`
2. Add a file called `experiment.txt`
3. Commit the change
4. Switch back to `main`
5. Merge your experimental branch
6. Push to GitHub

Key Takeaways

✅ Git tracks every change to your code
✅ GitHub stores your code in the cloud
✅ Always pull before you push
✅ Write clear commit messages
✅ Use branches for new features
✅ Don't commit sensitive files
✅ Ask for help when stuck—every developer deals with Git issues!

Resources for Further Learning

- GitHub's Git Handbook
- Visualizing Git
- Oh Shit, Git!?! - Fixing common mistakes

- Our internal Slack channel: #git-help

Questions?

Remember: Everyone struggles with Git at first. It's normal! The best way to learn is by doing, making mistakes, and fixing them. You've got this! 💪

Your next steps:

1. Complete Exercise 1 today
2. Commit to your project repos daily
3. Review your commit history weekly
4. Ask questions in our team chat

Welcome to the team! Let's build something amazing together. 🚀