Block 5: Essential Tools for Reproducible Research

Python Module for Incoming ISE & OR PhD Students

Will Kirschenman

August 7, 2025 | 2:00 PM - 2:50 PM

North Carolina State University

NC STATE UNIVERSITY

Welcome to Block 5!

- **Goal:** Master version control for your coding projects
- **Duration:** 50 minutes of hands-on Git and GitHub
- **Format:** Presentation + interactive demonstrations

What We'll Cover

Why version control matters · Git fundamentals · GitHub workflow · VS Code integration

Session Learning Objectives

By the end of Block 5, you will:

1. Understand **why version control is essential** for coding projects
2. Know the **core Git concepts** (commits, branches, repositories)
3. Be able to use **essential Git commands** for daily work
4. Understand **GitHub** for hosting and sharing code
5. See how **VS Code integrates** with Git for seamless workflow
6. Have a **professional coding portfolio** foundation

Focus
Everything we learn applies to Python scripts, Jupyter notebooks, and coding assignments!

Why Version Control?

The Problem: Life Without Version Control

Sound familiar?

Your Project Folder

- `assignment1.py`
- `assignment1_v2.py`
- `assignment1_final.py`
- `assignment1_final_REAL.py`
- `assignment1_final_submitted.py`
- `assignment1_backup.py`
- `assignment1_working.py`
- `assignment1_before_refactor.py`

Common Problems

- Which version actually works?
- What changed between versions?
- How to merge teammate's changes?
- Accidentally deleted working code!
- Can't remember why you made changes
- Email attachments everywhere

There's a better way!

Git: The Solution to Version Chaos

What Git Does

- **Tracks every change** you make
- **Explains why** changes were made
- **Shows who** made each change
- **Allows time travel** to any version
- **Merges** changes from teammates
- **Backs up** your code online

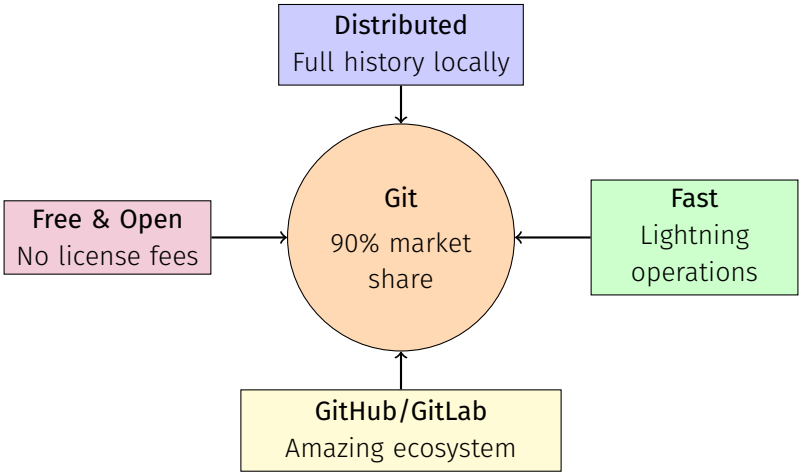
Real Benefits for PhD Students

- Never lose working code again
- Track progress on assignments
- Collaborate on group projects
- Build coding portfolio
- Show code evolution to instructors
- Professional skill for industry

The Bottom Line

Git turns chaos into organized history. It's like "Track Changes" for code!

Why Git Dominates Version Control

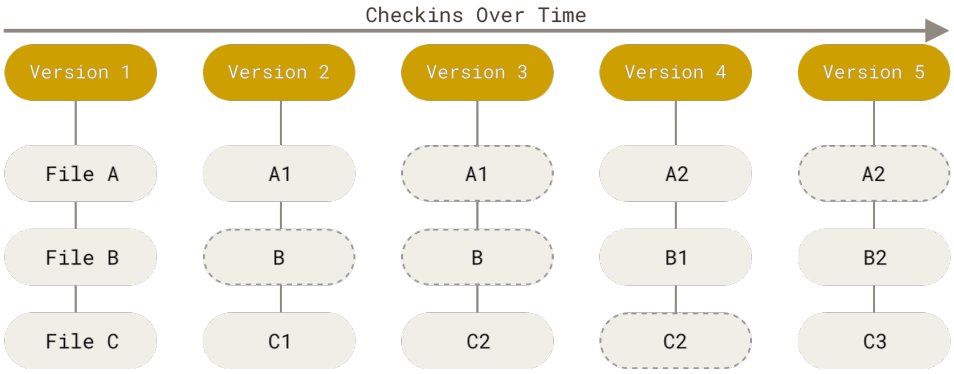


Industry Standard

Every tech company uses Git. Learning it now prepares you for internships and careers!

Git Fundamentals

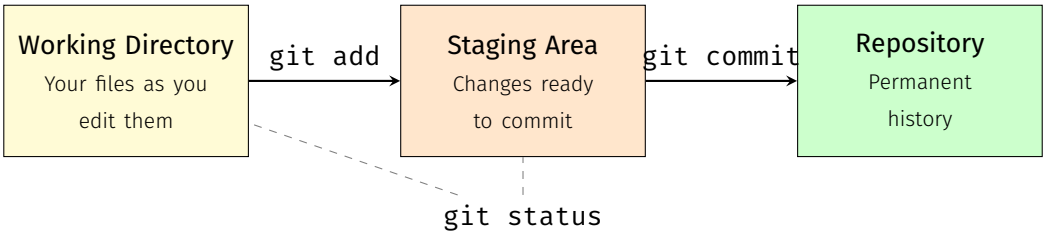
How Git Stores Your Code



Key Insight: Git Stores Snapshots, Not Differences

Unlike other version control systems, Git takes a ”snapshot” of your entire project at each commit. This makes Git incredibly fast and powerful!

Git's Mental Model: Three Key Areas



Edit
Write your Python code

Stage
Select changes to save

Commit
Save snapshot forever

Core Git Commands You'll Use Daily

Starting Out

```
# Initialize new repository
git init

# Clone existing repository
git clone <url>

# Check current status
git status
```

Viewing History

```
# See commit history
git log --oneline

# See what changed
git diff

# See staged changes
git diff --staged
```

Making Changes

```
# Stage specific file
git add mycode.py

# Stage all changes
git add .

# Commit with message
git commit -m "Add␣data␣analysis"
```

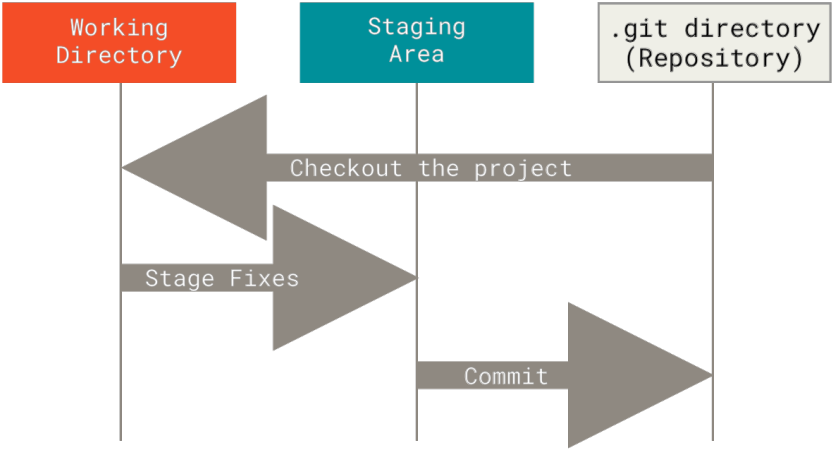
Working with GitHub

```
# Download updates
git pull

# Upload your commits
git push

# Set up remote
git remote add origin <url>
```

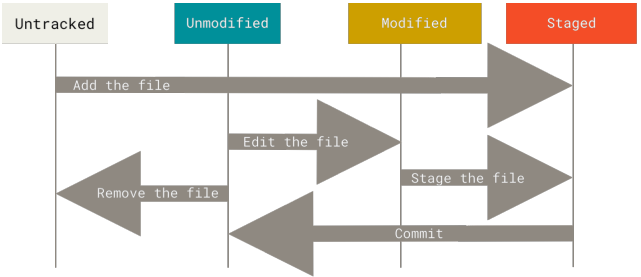
The Git Areas in Detail



Remember the Flow

Working Directory → Staging Area → Repository (Git Directory)

File Lifecycle in Git



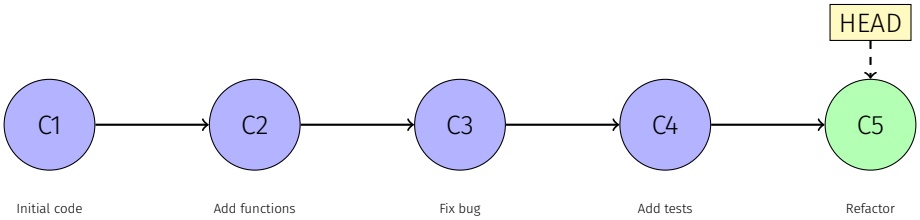
File States

- **Untracked**: New files Git doesn't know about
- **Unmodified**: Tracked, no changes
- **Modified**: Changed but not staged
- **Staged**: Ready to commit

Commands

- `git add`: Stage changes
- `git commit`: Save to repository
- `git rm`: Remove files
- `git status`: Check states

Understanding Commits: Your Code's Time Machine



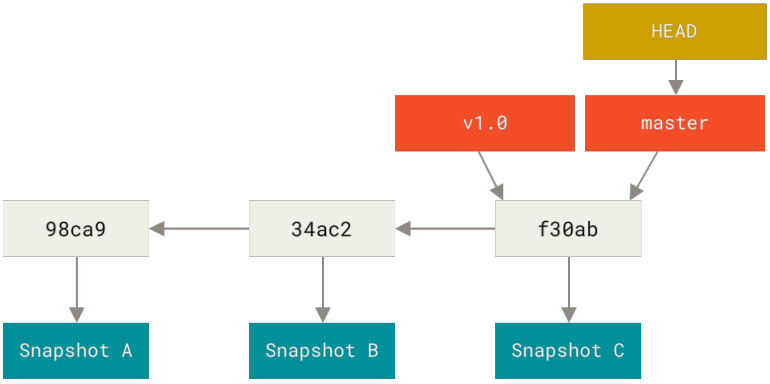
Each Commit Contains

- Snapshot of all files
- Author information
- Timestamp
- Commit message
- Link to parent commit

Good Commit Messages

- "Add data cleaning function"
- "Fix off-by-one error in loop"
- "Implement gradient descent"
- "Update requirements.txt"
- "Refactor for readability"

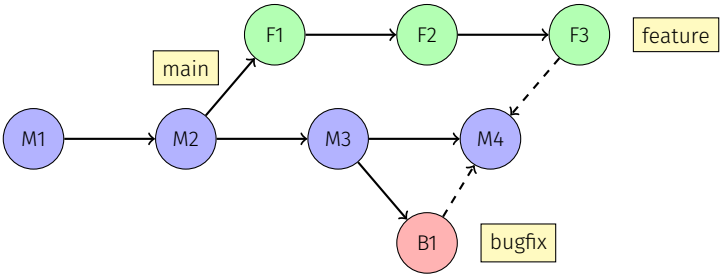
How Git Tracks History



Branches Are Lightweight Pointers

A branch in Git is simply a movable pointer to a specific commit. Creating branches is instant and takes minimal space!

Branches: Parallel Universes for Your Code



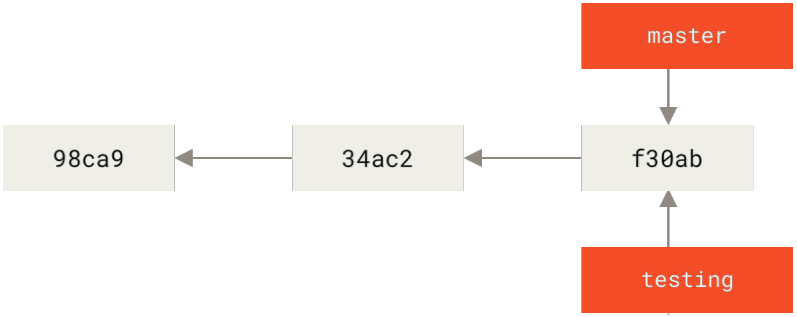
Why Use Branches?

- Experiment without breaking main
- Work on features independently
- Try different solutions
- Keep main branch stable

Common Branch Names

- `main` - stable code
- `feature/add-plotting`
- `bugfix/array-index`
- `experiment/new-algo`

Multiple Branches in Action



Creating Branches

```
# Create new branch
git branch testing

# Switch to branch
git checkout testing
```

Branch Strategy

- `main`: Stable code
- `develop`: Integration
- `feature/*`: New features
- `bugfix/*`: Bug fixes

GitHub Integration

GitHub: Your Code's Home in the Cloud

What is GitHub?

- Cloud hosting for Git repos
- Social network for coders
- Portfolio for your work
- Collaboration platform
- Free for public & private repos

Student Benefits

- GitHub Student Pack (free!)
- Private repos for coursework
- Show projects to employers
- Join open source projects

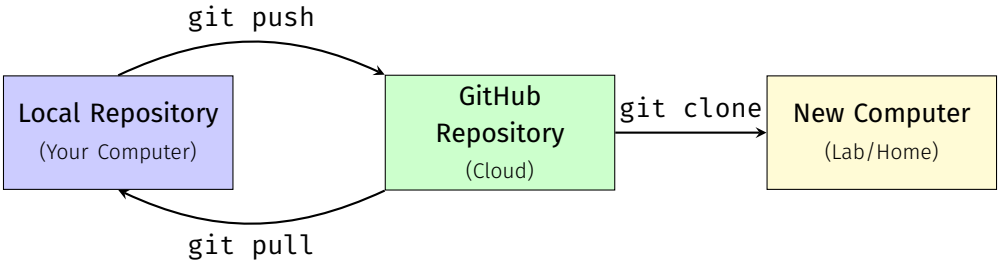
GitHub $\neq$ Git

- **Git**: Version control system
- **GitHub**: Website that hosts Git repositories
- Like email vs Gmail!

Alternatives

- GitLab (also popular)
- Bitbucket
- Self-hosted options

GitHub Workflow: Local and Remote



Typical Workflow

1. Create repository on GitHub
2. Clone to your computer
3. Make changes and commit locally
4. Push commits to GitHub
5. Pull changes when working from different computer

(Optional) Live Demo: Let's Create a Repository!

1. Go to `github.com` and sign in
2. Click the green "New" button
3. Name it: `python-coursework`
4. Add a README and .gitignore for Python
5. Create repository
6. Clone to your computer:

```
git clone https://github.com/YOUR-USERNAME/python-coursework.git
cd python-coursework
```

Pro Tip
Use descriptive repository names that explain the content!

Forks: Contributing to Other Projects

What is a Fork?

- Your personal copy of someone else's repository
- Lives in your GitHub account
- Independent from original
- Can sync with original repo
- Essential for open source contribution

When to Fork

- Contributing to open source
- Experimenting with others' code
- Building on existing projects
- Class assignments from template repos

Fork Workflow

1. Fork repository on GitHub
2. Clone YOUR fork locally
3. Create feature branch
4. Make changes and commit
5. Push to YOUR fork
6. Create Pull Request to original

Staying Updated

```
# Add original as upstream
git remote add upstream <original-url>

# Fetch and merge updates
git fetch upstream
git merge upstream/main
```

VS Code Integration

VS Code: Git Integration Built-In

Source Control Panel

- See all changed files
- Stage/unstage with clicks
- Write commit messages
- Push/pull with buttons
- View diffs side-by-side
- Resolve merge conflicts

Helpful Extensions

- GitLens (see blame, history)
- Git Graph (visualize branches)
- GitHub Pull Requests

VS Code Git Features

- **Gutter indicators**: See changes in editor
- **Status bar**: Current branch info
- **Timeline**: File history view
- **Inline blame**: See who wrote each line
- **Commit graph**: Visualize branches

Getting Started

- Open Source Control panel
- Initialize repository or clone
- Stage changes with + button
- Commit with checkmark button

Best Practices

Git Best Practices for Coursework

Commit Practices

- Commit early and often
- One logical change per commit
- Write clear commit messages
- Test before committing

What NOT to Commit

- Large data files (use .gitignore)
- API keys or passwords
- `__pycache__` folders
- `.ipynb_checkpoints`
- Virtual environments
- Compiled binaries

Project Structure

```
my-project/
|-- README.md
|-- .gitignore
|-- requirements.txt
|-- src/
|   +-- main.py
|-- data/
|   +-- .gitkeep
+-- tests/
    +-- test_main.py
```

The .gitignore File: Your Project's Bouncer

Purpose: Tell Git which files to ignore

Python .gitignore Template

```
# Python
__pycache__/
*.so
.Python
env/
venv/

# Jupyter
.ipynb_checkpoints/

# Data
*.csv
data/
```

IDE and OS

```
# VS Code
.vscode/
*.code-workspace

# Mac
.DS_Store

# Windows
Thumbs.db

# Secrets
.env
config.ini
```

Pro Tip

GitHub provides templates! When creating a repo, select "Python" for .gitignore.

Practical Application

Real Scenario: Course Project Workflow

Scenario: You're working on a machine learning project for class

1. **Initial Setup**
   - Create GitHub repository
   - Clone locally
   - Add README with project description
   - Create .gitignore for Python
2. **Development Cycle**
   - Create feature branch: `git checkout -b add-preprocessing`
   - Write code, test locally
   - Commit changes: `git commit -m "Add data preprocessing"`
   - Push branch: `git push origin add-preprocessing`
   - Merge to main when complete
3. **Collaboration** (if group project)
   - Team members clone repository
   - Each person works on separate branch
   - Regular pulls to stay synchronized
   - Merge branches via pull requests

Building Your Portfolio Early

Why Start Now?

- Employers check GitHub profiles
- Shows progression over time
- Demonstrates real projects
- Proves you can use tools
- Networking opportunities

What to Include

- Course projects (if allowed)
- Personal experiments
- Tutorial implementations
- Open source contributions
- Documentation practice

Portfolio Tips

- Add README to every project
- Include installation instructions
- Add example usage
- Keep repositories organized
- Pin your best projects

Professional Profile

- Professional username
- Clear bio
- Profile picture
- Contact information
- Link to LinkedIn

Hands-On Practice

Live Coding Exercise: Your First Repository

Let's practice together! We'll create a simple Python project with Git.

Exercise: Statistical Calculator

1. Create a new repository on GitHub
2. Clone it locally
3. Create `stats_calc.py` with basic functions
4. Add and commit the file
5. Create a feature branch for new functionality
6. Add more functions on the branch
7. Merge back to main
8. Push everything to GitHub

We'll Practice

- Basic Git workflow
- Writing good commit messages
- Working with branches
- Using VS Code's Git features

Next Steps

What We Accomplished in Block 5 ✓

- ✓ **Understood** why version control matters
- ✓ **Learned** Git's core concepts and commands
- ✓ **Created** a GitHub account and repository
- ✓ **Practiced** the basic Git workflow
- ✓ **Explored** VS Code's Git integration
- ✓ **Started** building a professional portfolio

Ready for Block 6!
You now have the tools to manage and share your code professionally

Preview: Block 6 (3:00 PM - 3:50 PM)

Advanced Topics & AI for Research Productivity

What's Coming

- High-performance computing preview
- AI-powered coding assistants
- GitHub Copilot demonstration
- Advanced productivity tools
- Q&A about PhD journey

Quick Prep

- Push your practice code to GitHub
- Think about your coding challenges
- Prepare questions about tools
- Get ready for the future of coding!

10-minute break, then we explore the cutting edge!

Resources for Continued Learning

Essential Resources

- **Pro Git Book (Free)**
  git-scm.com/book
- **GitHub Learning Lab**
  lab.github.com
- **Atlassian Git Tutorial**
  atlassian.com/git
- **VS Code Git Guide**
  code.visualstudio.com/docs

Practice Ideas

- Version control all coursework
- Contribute to open source
- Create a personal website with GitHub Pages
- Track your learning projects
- Collaborate with classmates

Remember
The best way to learn Git is to use it daily. Start with your next assignment!

Questions?

See you in 10 minutes for Block 6!