Introduction to GitHub Workflows

Version Control for Collaborative Research

Prof. Maithreyi Gopalan

2026-01-07

class: inverse, center, middle

Why Version Control?

The Problem

Have you ever…

- Had multiple versions of the same file? (`thesis_final.docx`, `thesis_final_v2.docx`, `thesis_final_ACTUAL.docx`)
- Lost track of what changes you made when?
- Worked with collaborators and struggled to merge your work?
- Wanted to go back to a previous version but couldn't remember which one?

Version control systems like Git solve these problems!

What is Git?

Git is a distributed version control system that:

- Tracks changes to your files over time
- Allows multiple people to work on the same project
- Lets you experiment without fear of breaking things
- Maintains a complete history of your project

GitHub is a web-based platform that:

- Hosts Git repositories online
- Facilitates collaboration
- Provides additional tools for project management

Basic Git/GitHub Workflow

The Core Workflow

Local (Your Computer)

1. **Clone** - Download repository
2. **Edit** - Make changes
3. **Stage** - Select changes
4. **Commit** - Save snapshot
5. **Push** - Upload to GitHub

Remote (GitHub)

- **Repository** - Project storage
- **Fork** - Personal copy
- **Pull Request** - Propose changes
- **Merge** - Integrate changes

Key Concepts

Repository (Repo): A project folder containing all files and their complete history

Commit: A snapshot of your project at a specific point in time

Branch: A parallel version of your repository (we'll use `main` for now)

Fork: Your personal copy of someone else's repository

Clone: Downloading a repository to your local computer

Push: Uploading your local commits to GitHub

Pull: Downloading updates from GitHub to your local computer

Part 1: Creating Your First Repository

Step 1: Create a Repository on GitHub

1. Go to github.com and sign in
2. Click the **"+"** icon in the top right
3. Select **"New repository"**
4. Fill in the details:
    - **Repository name**: `my-first-repo`
    - **Description**: Optional but recommended
    - **Public** or Private
    - ✅ **Initialize with a README**
5. Click **"Create repository"**

What Just Happened?

You created a **remote repository** on GitHub with:

- A unique URL (e.g., `github.com/yourusername/my-first-repo`)
- A `README.md` file (Markdown format)
- A default branch called `main`

The README file is displayed on your repository's home page and should describe:

- What the project is
- How to use it
- How to contribute

Your First Edit on GitHub

Let's edit the README directly on GitHub:

1. Click on `README.md` in your repository
2. Click the **pencil icon** (Edit this file)
3. Add some text:

   ```
   # My First Repository

   This is my first GitHub repository where I'm learning version control.

   ## About Me
   I'm learning data science and research methods.
   ```

4. Scroll down to **"Commit changes"**
5. Add a commit message: "Update README with introduction"
6. Click **"Commit changes"**

Part 2: Cloning to Your Local Computer

Getting Your Repository Locally

Cloning creates a local copy of a remote repository on your computer.

Steps:

1. On your GitHub repository page, click the green **"Code"** button
2. Copy the HTTPS URL (looks like `https://github.com/username/my-first-repo.git`)
3. Open your terminal/command prompt
4. Navigate to where you want to store the repository:

   ```
   cd ~/Documents  # or wherever you want
   ```

5. Clone the repository:

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

What Cloning Does

```
$ git clone https://github.com/username/my-first-repo.git
Cloning into 'my-first-repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
```

You now have:

- A local copy of all files
- Complete commit history
- Connection to the remote repository
- Ability to work offline

Navigate into your repository:

```
cd my-first-repo
```

Checking Your Repository Status

Use `git status` to see the current state:

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

This tells you:

- Which branch you're on (`main`)
- Whether you have uncommitted changes (none yet)
- Whether your local branch is ahead/behind the remote

Part 3: Making Changes and Committing

The Git Workflow

```
Working Directory → Staging Area → Repository
     (edit)             (add)         (commit)
```

1. **Working Directory**: Where you make changes
2. **Staging Area**: Where you prepare changes for commit
3. **Repository**: Where committed changes are permanently stored

Creating a New File

Let's create a learning log:

```
# Create a new file
touch learning-log.md

# Or use your text editor to create and edit:
# nano learning-log.md
# code learning-log.md  (VS Code)
```

Add content to `learning-log.md`:

```
# Learning Log

## January 6, 2026

Today I learned:
- How to create a GitHub repository
- How to clone a repository to my computer
- The difference between Git and GitHub
```

Save the file.

Checking Status Again

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present
```

Untracked means Git sees the file but isn't tracking changes to it yet.

Staging Your Changes

Stage the file to prepare it for commit:

```
$ git add learning-log.md
```

Check status again:

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   learning-log.md
```

The file is now in the **staging area**, ready to be committed.

Committing Your Changes

A **commit** creates a permanent snapshot with a message:

```
$ git commit -m "Add initial learning log entry"
[main 1a2b3c4] Add initial learning log entry
 1 file changed, 8 insertions(+)
 create mode 100644 learning-log.md
```

Writing Good Commit Messages

✅ **Good**:

- "Add data cleaning script"
- "Fix bug in regression analysis"
- "Update README with installation instructions"

❌ **Bad**:

- "stuff"
- "changes"
- "asdf"

Pushing to GitHub

Your commit is local. To upload it to GitHub:

```
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/username/my-first-repo.git
   a1b2c3d..e4f5g6h  main -> main
```

Now go to your GitHub repository in your browser and refresh - you'll see your new file!

The Complete Workflow

```
# 1. Make changes to files
echo "New content" >> learning-log.md

# 2. Check what changed
git status

# 3. Stage changes
git add learning-log.md
# Or stage all changes: git add .

# 4. Commit with a message
git commit -m "Add new learning entry"

# 5. Push to GitHub
git push origin main
```

Part 4: Forking and Contributing

What is Forking?

Forking creates your own copy of someone else's repository on GitHub.

Use cases:

- Contributing to open source projects
- Creating your own version of a project
- Experimenting without affecting the original
- Collaborating with classmates

Fork ≠ Clone:

- **Fork**: Makes a copy on GitHub (remote to remote)
- **Clone**: Downloads to your computer (remote to local)

How to Fork a Repository

1. Go to the repository you want to fork (e.g., a classmate's repo)
2. Click the **"Fork"** button in the top right
3. GitHub creates a copy under your account:
    - Original: `classmate/my-first-repo`
    - Your fork: `yourusername/my-first-repo`
4. You now have complete control over your fork

Working with a Fork

After forking, clone YOUR fork to your computer:

```
# Clone your fork (not the original)
git clone https://github.com/yourusername/my-first-repo.git
cd my-first-repo

# Check the remote
git remote -v
origin  https://github.com/yourusername/my-first-repo.git (fetch)
origin  https://github.com/yourusername/my-first-repo.git (push)
```

Make changes just like before:

1. Edit files
2. Stage changes (`git add`)
3. Commit (`git commit`)
4. Push to YOUR fork (`git push origin main`)

Contributing Back: Pull Requests

A **Pull Request (PR)** asks the original owner to accept your changes.

Workflow:

1. Fork the repository
2. Clone your fork
3. Make improvements
4. Push to your fork
5. Create a Pull Request
6. Original owner reviews and (hopefully) merges

Creating a Pull Request

Let's say you want to add feedback to a classmate's repo:

1. Create a new file `peer-review.md` in your forked copy

```
# Peer Review

Great work on your repository! I especially liked how you
organized your learning log. One suggestion: consider adding
dates to each entry for easier tracking.
```

2. Commit and push to YOUR fork:

```
git add peer-review.md
git commit -m "Add peer review feedback"
git push origin main
```

Submitting the Pull Request

1. Go to YOUR fork on GitHub
2. Click **"Contribute"** → **"Open pull request"**
3. GitHub will show:
    - **Base repository**: The original (your classmate's)
    - **Head repository**: Your fork
    - Changes you made
4. Add a title and description explaining your changes
5. Click **"Create pull request"**

Your classmate will be notified and can review, comment, and merge your changes!

Part 5: Best Practices

Git Best Practices

1. **Commit often**: Small, focused commits are better than large ones
2. **Write clear commit messages**: Others (including future you!) need to understand what changed
3. **Pull before you push**: Always sync with the remote before pushing

   ```
   git pull origin main
   git push origin main
   ```

4. **Don't commit sensitive data**: Passwords, API keys, personal data
5. **Use .gitignore**: Exclude files you don't want to track
   - Large data files
   - Temporary files
   - System files (`.DS_Store`, `Thumbs.db`)

Creating a .gitignore File

Create a file named `.gitignore` in your repository root:

```
# R specific
.Rproj.user/
.Rhistory
.RData
.Ruserdata

# Data files (if large)
.csv
data/

# System files
.DS_Store
Thumbs.db

# Temporary files
.tmp
~
```

Git will automatically ignore these files and directories.

Markdown Tips for README Files

Markdown is a simple formatting language used in README files:

```
# Header 1
## Header 2
### Header 3

*bold text**
italic text*

- Bullet point 1
- Bullet point 2

1. Numbered item 1
2. Numbered item 2

[Link text](http://example.com)

`inline code`
```

code block

\`\`\`

Practical Exercise

Your Assignment

Complete all five exercises in the problem set:

1. **Create** your own repository with a README
2. **Clone** your repository locally
3. **Make changes** and commit them
4. **Fork** a classmate's repository
5. **Contribute** with a pull request

Submission:

- URL to your repository
- Screenshot of your commit history
- URL to your fork of a classmate's repository

Time: ~30-45 minutes

Resources

Documentation:

- GitHub Docs
- Git Documentation
- GitHub Learning Lab

Quick Reference:

- Git Cheat Sheet
- Markdown Guide

Getting Help:

- Check `git status` frequently
- Read error messages carefully
- Search for errors online
- Ask classmates and instructor

Common Terminal Commands

```
# Navigation
pwd             # Print working directory
ls              # List files
cd folder       # Change directory
cd ..           # Go up one directory

# File operations
touch file      # Create new file
mkdir folder    # Create new folder
rm file         # Delete file
cat file        # View file contents

# Git commands
git status      # Check status
git log         # View commit history
git add .       # Stage all changes
git commit -m "message"  # Commit
git push        # Push to remote
git pull        # Pull from remote
```

Questions?

Key Takeaways

1. **Version control** tracks changes to your files over time
2. **Git** is the version control system; **GitHub** is the hosting platform
3. The basic workflow is:
    - Edit → Stage → Commit → Push
4. **Forking** lets you create your own copy of repositories
5. **Pull requests** allow you to contribute to others' projects
6. **Commit often** with clear messages
7. Git takes practice - don't worry if it feels confusing at first!

Now let's practice!

Open your terminal and let's create your first repository together.