01_git_basics

GitHub Complete Beginner's Guide

Table of Contents

1. Introduction to Version Control and GitHub
2. Installing Git
3. Initial Git Configuration
4. Understanding Git Concepts
5. Creating Your First Repository
6. Basic Git Workflow
7. Connecting to GitHub
8. Troubleshooting Common Issues

---

1. Introduction to Version Control and GitHub

What is Version Control?

Version control is a system that records changes to files over time so you can recall specific versions later. Think of it as a "time machine" for your code.

Why do we need it?

- Track changes in your code
- Collaborate with team members
- Revert to previous versions if something breaks
- Maintain multiple versions of your project
- Keep a history of who changed what and when

What is Git?

Git is a **distributed version control system** created by Linus Torvalds in 2005. It runs locally on your computer and tracks changes to your files.

What is GitHub?

GitHub is a **cloud-based hosting service** for Git repositories. It provides:

- Remote storage for your code
- Collaboration tools
- Project management features
- A platform to showcase your work

Analogy: If Git is like Microsoft Word's "Track Changes" feature, GitHub is like Google Drive where you store and share those documents.

---

2. Installing Git

For Windows:

1. **Download Git**
   - Visit: https://git-scm.com/download/win
   - Download the latest version (e.g., 64-bit Git for Windows Setup)
2. **Run the Installer**
   - Double-click the downloaded `.exe` file
   - Follow the installation wizard:
     - Click "Next" through most screens
     - **Important:** Select "Git from the command line and also from 3rd-party software"
     - Choose "Use bundled OpenSSH"
     - Select "Use the OpenSSL library"
     - Choose "Checkout Windows-style, commit Unix-style line endings"
     - Select "Use MinTTY (the default terminal of MSYS2)"
     - Click "Install"
3. **Verify Installation**

```shell
git --version
```

You should see something like: `git version 2.42.0.windows.1`

For macOS:

1. **Using Homebrew (Recommended)**

```shell
brew install git
```

1. **Or download from:** https://git-scm.com/download/mac
2. **Verify Installation**

```shell
git --version
```

For Linux (Ubuntu/Debian):

```shell
sudo apt-get update
sudo apt-get install git
```

Verify:

```shell
git --version
```

---

3. Initial Git Configuration

Before using Git, you need to configure your identity. This information will be attached to all your commits.

Set Your Global Username

```shell
git config --global user.name "Your Name"
```

Example:

```shell
git config --global user.name "John Doe"
```

Set Your Global Email

```shell
git config --global user.email "your.email@example.com"
```

Example:

```shell
git config --global user.email "john.doe@email.com"
```

Note: Use the same email address you'll use for your GitHub account.

Verify Your Configuration

```shell
git config --global --list
```

You should see:

```
user.name=John Doe
user.email=john.doe@email.com
```

Why Global Configuration?

The `--global` flag means these settings apply to all repositories on your computer. You only need to do this once per computer.

---

4. Understanding Git Concepts

Before we start using Git, let's understand some key concepts:

Repository (Repo)

A repository is a project folder that Git tracks. It contains:

- Your project files
- A hidden `.git` folder (where Git stores all version history)

Working Directory

This is your project folder where you create, edit, and delete files. It's what you see in your file explorer.

Staging Area (Index)

Think of this as a "preparation area" where you select which changes you want to include in your next commit. It's like putting items in a shopping cart before checkout.

Commit

A commit is a snapshot of your project at a specific point in time. Each commit has:

- A unique ID (hash)
- Author information
- Timestamp
- Commit message describing the changes

Remote Repository

A version of your repository hosted on the internet (like GitHub). This allows collaboration and backup.

The Git Workflow Diagram

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

---

5. Creating Your First Repository

Let's create a project folder called `python_api_dev` and initialize it as a Git repository.

Step 1: Create the Project Folder

Open your terminal/command prompt and navigate to where you want to create your project:

```shell
# Navigate to your desired location (e.g., Desktop)
cd Desktop

# Create the folder
mkdir python_api_dev

# Navigate into the folder
cd python_api_dev
```

Explanation:

- `mkdir` = "make directory" (creates a new folder)
- `cd` = "change directory" (moves into a folder)

Step 2: Verify Your Location

```shell
pwd  # On Mac/Linux (prints working directory)
cd   # On Windows (shows current directory)
```

You should see a path ending with `python_api_dev`.

Step 3: Initialize Git Repository

```shell
git init
```

Output:

```
Initialized empty Git repository in /Users/yourname/Desktop/python_api_dev/.git/
```

What just happened? Git created a hidden `.git` folder in your project directory. This folder contains all the version control information. Your folder is now a Git repository!

To see the hidden .git folder:

```shell
ls -la              # Mac/Linux
dir /a              # Windows Command Prompt
Get-ChildItem -Force  # Windows PowerShell
ls -Force           # Windows PowerShell (short version)
```

Step 4: Rename Branch to 'main'

By default, Git might create a branch called 'master'. GitHub's standard is now 'main', so let's rename it:

```shell
git branch -M main
```

Explanation:

- `git branch -M main` renames your current branch to 'main'
- `-M` flag forces the rename even if the branch already exists
- This ensures compatibility with GitHub's default branch name

Note: Modern Git installations (2.28+) can be configured to use 'main' by default:

```shell
git config --global init.defaultBranch main
```

---

6. Basic Git Workflow

Now let's add some files and track them with Git.

Step 1: Create Some Files

Let's create a few Python files for our API development project:

```shell
# Create a main Python file
echo "print('Hello, API World!')" > main.py

# Create a requirements file
echo "flask==2.3.0" > requirements.txt

# Create a README file
echo "# Python API Development Training" > README.md
```

Or create them manually:

- Open a text editor
- Create `main.py` with some Python code

- Create `requirements.txt` with package names
- Create `README.md` with project description

Step 2: Check Repository Status

```shell
git status
```

Output:

```
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        main.py
        requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Note: If you see `On branch master` instead of `On branch main`, run:

```shell
git branch -M main
```

Understanding the Output:

- **Untracked files:** Git sees these files but isn't tracking changes to them yet
- **Red color** (if your terminal supports colors): Indicates files not staged

Step 3: Add Files to Staging Area

You have two options:

Option A: Add individual files

```shell
git add main.py
git add requirements.txt
git add README.md
```

Option B: Add all files at once (recommended)

```shell
git add .
```

Explanation:

- `git add` tells Git to start tracking these files
- The `.` means "add everything in the current directory"
- Files are now in the **staging area**

Step 4: Verify Staging

```shell
git status
```

Output:

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
        new file:   main.py
        new file:   requirements.txt
```

Green color: Indicates files are staged and ready to commit.

Step 5: Create Your First Commit

```shell
git commit -m "Initial commit: Add main.py, requirements.txt, and README"
```

Explanation:

- `git commit` creates a snapshot of your staged changes
- `-m` flag allows you to add a message inline
- The message should be descriptive and explain **what** changed

Output:

```
[main (root-commit) a1b2c3d] Initial commit: Add main.py, requirements.txt, and
README
 3 files changed, 3 insertions(+)
 create mode 100644 README.md
 create mode 100644 main.py
 create mode 100644 requirements.txt
```

What happened?

- Git created a snapshot of your project
- Each file's current state is now saved
- You got a unique commit ID (e.g., `a1b2c3d`)

Step 6: View Commit History

```shell
git log
```

Output:

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0 (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Mon Oct 6 10:30:00 2025 +0530

    Initial commit: Add main.py, requirements.txt, and README
```

Shorter version:

```shell
git log --oneline
```

Output:

```
a1b2c3d (HEAD -> main) Initial commit: Add main.py, requirements.txt, and README
```

---

7. Connecting to GitHub

Now let's push our local repository to GitHub so it's backed up and shareable.

Step 1: Create a GitHub Account

1. Visit: https://github.com
2. Click "Sign up"
3. Follow the registration process
4. Verify your email address

Step 2: Create a New Repository on GitHub

1. **Log in to GitHub**
2. **Click the "+" icon** in the top-right corner
3. **Select "New repository"**
4. **Fill in the details:**
   - **Repository name:** `python_api_dev`
   - **Description:** (optional) "Training repository for Python API development"

- **Visibility:** Public or Private
- **DO NOT** check "Initialize this repository with a README" (we already have files)

5. **Click "Create repository"**

Step 3: Copy the Repository URL

After creating the repository, GitHub shows you a setup page. You'll see a URL like:

```
https://github.com/yourusername/python_api_dev.git
```

Copy this URL.

Step 4: Add Remote Repository

Back in your terminal, link your local repository to GitHub:

```shell
git remote add origin https://github.com/yourusername/python_api_dev.git
```

Explanation:

- `git remote add` links your local repo to a remote one
- `origin` is the default name for your primary remote repository
- Replace `yourusername` with your actual GitHub username

Verify the remote:

```shell
git remote -v
```

Output:

```
origin  https://github.com/yourusername/python_api_dev.git (fetch)
origin  https://github.com/yourusername/python_api_dev.git (push)
```

Step 5: Push Your Code to GitHub

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

Explanation:

- `git push` uploads your commits to GitHub
- `-u` sets up tracking (you only need this the first time)
- `origin` is the remote name
- `main` is the branch name

First-time push: You may be asked to authenticate:

- Enter your GitHub username
- Enter your **personal access token** (not your password)

Output:

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 450 bytes | 450.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/yourusername/python_api_dev.git
 * [new branch]      main → main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Step 6: Verify on GitHub

1. Go to your repository URL in a browser
2. You should see your files: `main.py`, `requirements.txt`, and `README.md`
3. Click on files to view their contents

Congratulations! Your code is now on GitHub!

---

8. Making Changes and Pushing Again

Let's practice the full workflow with a change.

Step 1: Modify a File

Edit `main.py` to add more code:

SHELL

```
# Open in your text editor and add:
# def greet(name):
#     return f"Hello, {name}!"
#
# print(greet("Developer"))
```

Step 2: Check Status

SHELL

```
git status
```

Output:

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

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Red "modified": Git detected changes but they're not staged yet.

Step 3: View Changes

```shell
git diff main.py
```

This shows exactly what changed in the file (lines added/removed).

Step 4: Stage the Changes

```shell
git add main.py
```

Step 5: Commit the Changes

```shell
git commit -m "Add greet function to main.py"
```

Step 6: Push to GitHub

Since we already set up tracking with `-u`, we can now simply use:

```shell
git push
```

Output:

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 280 bytes | 280.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/yourusername/python_api_dev.git
   a1b2c3d..e4f5g6h  main → main
```

9. Essential Git Commands Summary

Configuration

```shell
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
git config --global --list
git config --global init.defaultBranch main  # Set 'main' as default branch
```

Repository Setup

```shell
git init                    # Initialize a new repository
git branch -M main          # Rename branch to 'main' (if needed)
git remote add origin <url> # Link to GitHub repository
```

Daily Workflow

```shell
git status                  # Check repository status
git add <file>              # Stage a specific file
git add .                   # Stage all changes
git commit -m "message"     # Commit staged changes
git push                    # Upload commits to GitHub
git pull                    # Download changes from GitHub
```

Viewing Information

```shell
git log                     # View commit history
git log --oneline           # Compact commit history
git diff                    # View unstaged changes
git diff <file>             # View changes in specific file
git remote -v               # View remote repositories
```

Undoing Changes

```shell
git restore <file>          # Discard changes in working directory
git restore --staged <file> # Unstage a file
git reset HEAD~1            # Undo last commit (keep changes)
```

10. Best Practices

Commit Messages

- **Be descriptive:** "Add user authentication" not "Update code"
- **Use present tense:** "Add feature" not "Added feature"
- **Keep first line under 50 characters**
- **Add details in body if needed**

Good examples:

```
Add login functionality for users
Fix bug in payment processing
Update README with installation instructions
```

When to Commit

- **Commit often:** Small, logical chunks
- **Each commit should work:** Don't commit broken code
- **One feature per commit:** Makes tracking easier

What NOT to Commit

Create a `.gitignore` file to exclude:

- Passwords and API keys
- Large binary files
- Temporary files
- Dependencies (node_modules, venv)

Example .gitignore for Python:

```
__pycache__/
*.pyc
venv/
.env
*.log
```

---

11. Troubleshooting Common Issues

Issue 1: "Author identity unknown"

Error:

```
*** Please tell me who you are.
```

Solution:

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

Issue 2: "Failed to push"

Error:

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

Solution: Pull changes first, then push:

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

Issue 3: "Authentication failed"

Problem: GitHub no longer accepts password authentication.

Solution: Create a Personal Access Token:

1. Go to GitHub Settings → Developer settings → Personal access tokens → Tokens (classic)
2. Generate new token
3. Select scopes (at minimum: `repo`)
4. Copy the token
5. Use this token instead of your password

Issue 5: "src refspec main does not match any" when pushing

Error:

```
error: src refspec main does not match any
```

Problem: You're trying to push to 'main' branch but your local branch is named 'master'.

Solution:

```shell
git branch -M main  # Rename your branch to main
git push -u origin main  # Now push
```

Issue 6: PowerShell Commands Not Working

Problem: Commands like `dir /a` don't work in PowerShell.

Solution: PowerShell uses different commands:

```
Get-ChildItem -Force  # Show hidden files (full command)
ls -Force             # Show hidden files (short version)
```

---

12. Practice Exercise

Try this complete workflow:

1. Create a new file called `api_routes.py`
2. Add some Python code to it
3. Check the status
4. Stage the file
5. Commit with message "Add API routes module"
6. Push to GitHub
7. Verify on GitHub website

Commands:

SHELL

```
echo "# API Routes" > api_routes.py
git status
git add api_routes.py
git commit -m "Add API routes module"
git push
```

---

13. Next Steps

Once comfortable with basics, explore:

- **Branching:** Work on features without affecting main code
- **Pull Requests:** Collaborate with others
- **Merge Conflicts:** Resolve conflicting changes
- **GitHub Actions:** Automate testing and deployment
- **.gitignore:** Exclude files from tracking
- **Git stash:** Temporarily save changes

Key Takeaways

1. **Git is local, GitHub is remote** - Git works on your computer, GitHub stores it online
2. **The workflow is:** edit → add → commit → push
3. **Commit often** with clear messages
4. **Always pull before pushing** when collaborating
5. **Check status frequently** to know what's happening

Remember: Git seems complex at first, but with practice, these commands become second nature. Don't worry about making mistakes - that's what version control is for!