FROM ZERO TO CONFIDENT · BEGINNER SERIES

Git & uv Quick Start

Basic Instructions for Beginners

The essential commands to save your work, share it, and run Python projects

JUST THE BASICS

WRITTEN BY

Ayman Hamed

www.aymanhamed.com

Git & uv Quick Start

Basic Instructions for Beginners

Ayman Hamed

Contents

Introduction

This is a short guide for absolute beginners. It covers only the basic commands you need to start using Git and uv. No advanced features, no deep theory – just what you need to get going.

You will learn how to:

- Save versions of your work with Git
- Ignore files you do not want to share
- Put your work on GitHub
- Handle merge conflicts when they happen
- Create a Python project with uv
- Install packages and run your code

You need a terminal and basic typing ability. Nothing else.

Git Basics

What Git does

Git saves snapshots of your files. Every time you commit, Git remembers exactly how all your files looked at that moment. You can go back to any saved version at any time.

Install Git

Linux:

```
sudo apt install git
```

macOS:

```
brew install git
```

Windows: Download from https://git-scm.com/download/win

Set your name and email

Do this once on your computer:

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

Start a repository

A repository is just a folder that Git tracks:

```
mkdir my-project
cd my-project
git init
git branch -M main
```

The last line names your branch `main`. Some Git versions call it `master` by default. This line makes sure it is always `main`, which matches GitHub.

Ignore files before your first commit

Before you add anything, create a `.gitignore` file. This tells Git which files to skip. Create `.gitignore` with this content:

```
__pycache__/
*.pyc
.venv/
.env
dist/
.DS_Store
```

Warning

Do this first. If you commit before creating `.gitignore`, you may accidentally share passwords, API keys in `.env`, or junk folders like `__pycache__`. Always ignore before your first commit.

Now commit the `.gitignore` itself:

```
git add .gitignore
git commit -m "Add .gitignore"
```

The three commands you will use every day

```
git add .                                # stage all changes
git commit -m "describe what you did"    # save a snapshot
git log --oneline                        # see your history
```

Edit files, `git add .`, `git commit`. Repeat. That is the core loop.

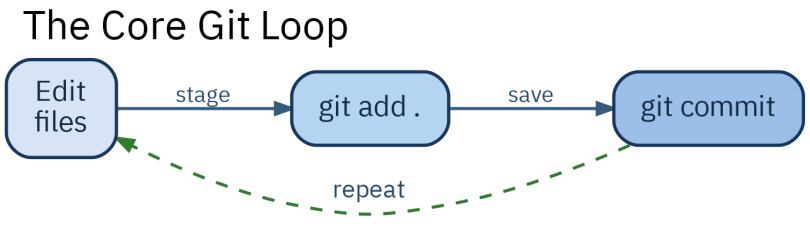

Figure 1: The core Git loop: edit, stage, commit, repeat.

Check what changed

```
git status
```

This shows what files changed and what is ready to commit. Run it anytime you are not sure.

Branches

A branch lets you work on something new without breaking your main work.

Create a branch

```
git switch -c my-feature
```

Work on it

Edit files, then:

```
git add .
git commit -m "add my feature"
```

Merge it back

```
git switch main
git merge my-feature
```

Delete it when done

```
git branch -d my-feature
```

> Tip
>
> **Use a branch for every new thing.** It keeps your main work safe. You can always delete a branch if the idea did not work out.

Undoing Changes

Unstage a file

If you ran `git add` by accident:

```
git restore --staged file.txt
```

Throw away edits

If you do not want your changes anymore:

```
git restore file.txt
```

Undo a commit safely

If you committed something you did not mean to, `git revert` creates a new commit that undoes the previous one. Your history stays intact:

```
git revert HEAD
```

Git opens your editor for a message. Save and close. The bad commit is cancelled but still visible in history, which is safe.

> Tip
>
> **Prefer `git revert`.** It undoes a commit without erasing history. This is safe even if you have already pushed to GitHub.

GitHub

GitHub is a website where you store your Git repository online so others can see it and contribute.

Put your project on GitHub

1. Go to https://github.com and create a new repository.
2. Copy the URL GitHub gives you.
3. Connect your local repository:

```
git remote add origin https://github.com/yourname/my-project.git
git push -u origin main
```

Send your latest work to GitHub

```
git push
```

Get the latest work from GitHub

```
git pull
```

Copy someone's repository to your computer

```
git clone https://github.com/yourname/my-project.git
```

Tip

Always pull before you push. If someone else added changes to GitHub, `git push` will be rejected. Run `git pull` first.

Merge Conflicts

When you `git pull` and someone else changed the same line you did, Git pauses and says there is a conflict. This is normal, not dangerous.

What it looks like

Git prints:

```
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Open the file. You will see conflict markers:

```
<<<<<<< HEAD
Your version of the line.
=======
Their version of the line.
>>>>>>> origin/main
```

How to fix it

Edit the file to keep the version you want. Delete the markers (<<<<<<<, =======, >>>>>>>) completely. Then:

```
git add README.md
git commit -m "Resolve merge conflict in README"
```

If you want to start over

```
git merge --abort
```

This cancels the merge and puts everything back the way it was.

Tip

Conflicts are just Git asking for your decision. Edit the file, pick the version you want, stage, and commit. That is all.

Git Command Summary

Command	What it does
`git init`	Start a new repository
`git branch -M main`	Name the branch `main`
`git add .`	Stage all changes
`git commit -m "message"`	Save a snapshot
`git status`	See what changed
`git log --oneline`	See history
`git switch -c branch`	Create and switch to a branch
`git switch branch`	Switch to a branch
`git merge branch`	Merge a branch into the current one
`git branch -d branch`	Delete a branch
`git revert HEAD`	Undo the last commit safely
`git restore file`	Discard edits to a file
`git restore --staged file`	Unstage a file
`git push`	Send commits to GitHub
`git pull`	Get commits from GitHub
`git clone url`	Copy a repository
`git merge --abort`	Cancel a conflicted merge

uv Basics

What uv does

uv replaces pip and virtualenv with one fast tool. It creates Python projects, installs packages, and runs your code – all in one command.

Install uv

Linux and macOS:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Windows:

```
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Verify it works:

```
uv --version
```

Create a Python project

```
uv init my-app
cd my-app
```

This creates `main.py`, `pyproject.toml`, and a virtual environment. You do not need to activate anything.

Managing Packages

Add a package

```
uv add requests
```

uv installs the package, updates your project file, and locks the version.

Remove a package

```
uv remove requests
```

Install all dependencies

If you cloned a project, install everything it needs:

```
uv sync
```

Add a testing tool

```
uv add --dev pytest
```

The `--dev` flag means it is only used during development, not in production.

Running Your Code

Run a file

```
uv run main.py
```

You do not need to activate a virtual environment. uv handles it automatically.

Run tests

```
uv run pytest
```

Run any tool once

You can run a Python tool without installing it:

```
uvx ruff check main.py
```

This is useful for trying a tool before adding it to your project.

uv Command Summary

Command	What it does
`uv init name`	Create a new project
`uv add package`	Add a dependency
`uv add --dev package`	Add a dev-only dependency
`uv remove package`	Remove a dependency
`uv sync`	Install all dependencies from lock file
`uv run main.py`	Run a file in the project environment
`uv run pytest`	Run tests
`uvx tool args`	Run any tool once without installing it
`uv build`	Build a package for distribution
`uv publish`	Publish to PyPI

Putting It All Together

Here is a complete workflow using both Git and uv from start to finish.

Start a project

```
uv init my-app
cd my-app
git init
git branch -M main
```

Add a .gitignore before anything else

Create `.gitignore` with:

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

```
git add .gitignore
git commit -m "Add .gitignore"
```

Add a package and write code

```
uv add requests
```

Edit `main.py`:

```
import requests

def main():
    response = requests.get("https://httpbin.org/get")
    print(response.json())
```

```
if __name__ == "__main__":
    main()
```

Run and commit

```
uv run main.py
git add .
git commit -m "Add main script with requests"
```

Put it on GitHub

```
git remote add origin https://github.com/yourname/my-app.git
git push -u origin main
```

Someone else can now run it

```
git clone https://github.com/yourname/my-app.git
cd my-app
uv sync
uv run main.py
```

Create, code, commit, push, clone, sync, run. That is the whole workflow.

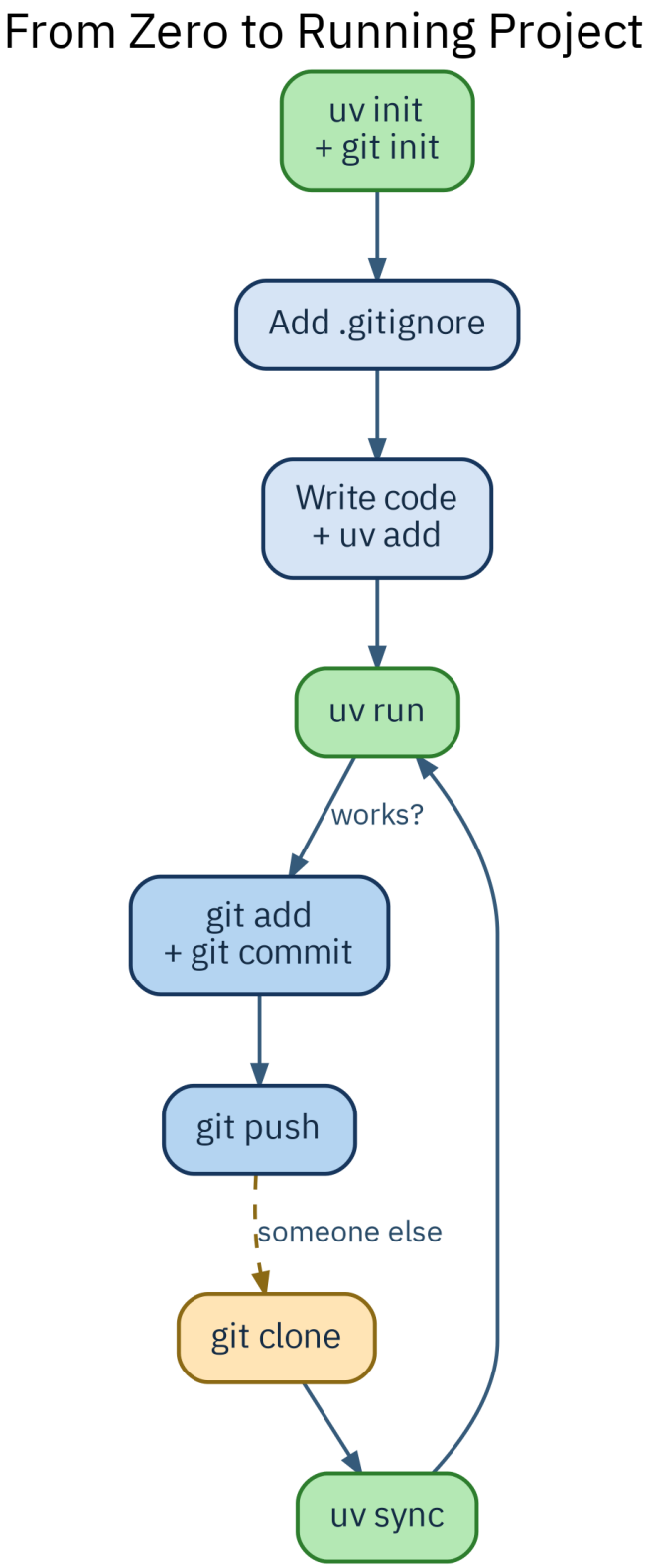

Figure 2: The full workflow from zero to running project.

What's Next

You now know enough to save your work with Git, share it on GitHub, and build Python projects with uv. The best way to solidify this is to use it on a real project.

If you want to go further and build AI agents with Python, these guides pick up where this one leaves off:

- **Serving AI Agents with FastAPI** – build and deploy production-ready agent APIs
- **Google ADK 2.0 Workflows** – orchestrate multi-agent systems with Google's Agent Development Kit

Both assume you know the basics covered here and take you straight into building real applications.

Written by Ayman Hamed

https://www.aymanhamed.com