# Git

## Installing Git

- **Windows/Mac**: Download from git-scm.com.
- **Linux**: Install using your package manager:

```
sudo apt install git
```

## Configuring Git

- Set up your global username and email:

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

## Basic Git Commands

- Create a new Git repository:

```
git init
```

- Add files to the staging area:

```
git add <filename>
```

- To add all files:

```
git add .
```

- Commit staged changes with a message:

```
git commit -m "Your commit message"
```

- Check the current status of your repository:

```
git status
```

- View a list of commits:

```
git log
```

## Branching and Merging

- Create a new branch:

```
git branch <branch-name>
```

- Switch to an existing branch:

```
git checkout <branch-name>
```

- Merge a branch into the current branch:

```
git merge <branch-name>
```

## `.gitignore`

- A `.gitignore` file tells Git which files or directories to ignore.
- Files listed in `.gitignore` won't be tracked, staged, or committed.
- Useful for excluding temporary, system-generated, or sensitive files.

### Why Use `.gitignore`?

1. **Prevent Clutter**: Avoid tracking unnecessary files like build outputs, logs, or temporary files.
2. **Protect Sensitive Information**: Keep secrets, credentials, or config files out of version control.
3. **Speed Up Workflows**: Ignoring large or unneeded files can make Git operations faster.

### Basic Structure of `.gitignore`

- Each line specifies a pattern to ignore.
- Blank lines or lines starting with `#` are comments.
- Example:

```
# Ignore all .log files
*.log

# Ignore the node_modules directory
node_modules/

# Ignore all .env files
.env
```

### Ignoring Files by Extension

- Ignore all files of a specific type (e.g., `.log`, `.pyc`):

```
*.log
*.pyc
```

### Ignoring Directories

- Ignore entire directories (e.g., `node_modules`, `build/`):

```
node_modules/
build/
```

### Ignoring System-Specific Files

- Ignore files specific to your operating system:

```
# macOS system files
.DS_Store

# Windows system files
Thumbs.db
```

## How to Create and Use `.gitignore`

1. **Create a `.gitignore` File**:
   - In the root of your repository, create a new file named `.gitignore`.
   - List all files and directories you want Git to ignore.
2. **Check Ignored Files**:
   - Use `git status` to verify which files are being ignored.

## Example `.gitignore` for Python Projects

```
# Byte-compiled files
*.pyc
__pycache__/

# Virtual environment directory
venv/

# Logs
*.log

# Environment variables
.env

# Jupyter Notebook checkpoints
.ipynb_checkpoints/
```

## Global `.gitignore`

- You can set a global `.gitignore` to exclude patterns across all repositories.
- Example:

```
git config --global core.excludesfile ~/.gitignore_global
```

  - Add patterns to `~/.gitignore_global` as you would in a regular `.gitignore`.

## Tips and Best Practices

1. **Add `.gitignore` Early**: Add it before committing unnecessary files.
2. **Use Templates**: Use pre-built `.gitignore` templates for common languages and frameworks (gitignore.io).
3. **Check Before Committing**: Make sure sensitive files aren't committed before adding them to `.gitignore`.

## Example of Converting Sk-stepwise to git

- add `.gitignore` file to the root of the project
- Run commands:

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
```

- add to GitHub project

# Pull Requests with GitHub

- A **Pull Request (PR)** is a mechanism to propose changes to a repository.
- It allows for code review and discussion before merging changes into the main codebase.
- Essential for **collaboration** and maintaining code quality.

## Why Use Pull Requests?

1. **Collaboration**: Enables multiple developers to work together efficiently.
2. **Code Review**: Reviewers can suggest changes, catch bugs, and discuss improvements.
3. **History**: Keeps a clear, documented history of why changes were made.
4. **Testing**: Automatically run tests on changes before merging.

## Creating a Pull Request

1. **Fork or Clone the Repository**:
   - Fork if you don't have write access.
   - Clone to your local machine to make changes.
2. **Create a Branch**:
   - Use branches to isolate your changes:

```
git checkout -b <branch-name>
```

3. **Make Changes**:
   - Edit, add, or delete files as needed.
   - Stage and commit your changes:

```
git add .
git commit -m "Description of changes"
```

4. **Push to GitHub**:
   - Push the branch to your GitHub repository:

```
git push origin <branch-name>
```

5. **Open a Pull Request**:
   - Navigate to your GitHub repository.
   - Click **New Pull Request** and choose the branch you pushed.
   - Add a title and description explaining the changes.

## Reviewing and Merging a Pull Request

1. **Assign Reviewers**:
   - Collaborators or maintainers can be assigned to review your PR.
   - Reviewers can leave comments, suggest changes, and approve the PR.
2. **Discussion and Changes**:
   - Discuss the changes directly in the PR.
   - If requested, push additional changes to the same branch, and the PR will update automatically.
3. **Merging the Pull Request**:
   - Once approved, the PR can be merged into the target branch (usually `main` or `develop`).
   - Options for merging:
     - **Merge Commit**: Keeps all commits and creates a merge commit.
     - **Squash and Merge**: Combines all commits into one before merging.
     - **Rebase and Merge**: Reapplies commits on top of the base branch.

### Best Practices for Pull Requests

1. **Small, Focused PRs**:
   - Keep PRs small and focused on one task or feature.
   - Easier to review and faster to merge.
2. **Detailed Descriptions**:
   - Clearly explain the **purpose**, **changes**, and **impact** of the PR.
   - Link related issues or tasks.
3. **Run Tests**:
   - Ensure all tests pass before submitting a PR.
   - Use GitHub Actions or other CI tools to automate this.
4. **Respond to Feedback**:
   - Address comments or requested changes promptly.
   - Be open to suggestions and improvements.

### Example of Creating a Pull Request for Sk-stepwise

Add `readme.md` file to the root of the project and push the changes to the remote repository. Then create a pull request on GitHub.

## GitHub Actions

This week's focus is on integrating **UV** with **GitHub Actions** to automate the CI/CD pipeline for Python projects. You'll learn how to set up, configure, and optimize UV and Python environments in GitHub Actions, leveraging UV's powerful tools for dependency management, caching, and workflow automation.

### Introduction to GitHub Actions

- *What is GitHub Actions?*
  - GitHub Actions is a CI/CD tool that automates software workflows directly within your GitHub repositories.
  - It allows developers to build, test, and deploy code automatically when specific events occur (e.g., pushes, pull requests).

- *Why Use GitHub Actions?*
  - **Automation**: Automate repetitive tasks like testing, deployment, and integration.
  - **Seamless Integration**: Works natively with GitHub, no need for external CI tools.
  - **Flexibility**: Supports custom workflows with many pre-built actions and integration options.

## Motivation Behind Using GitHub Actions

- **Continuous Integration (CI)**
  - Automatically run tests and checks whenever code is pushed.
  - Prevents the introduction of bugs by testing every change in real-time.
- **Continuous Deployment (CD)**
  - Automatically deploy changes to staging or production environments after tests pass.
  - Speeds up the release cycle, allowing for quicker updates and patches.
- **Collaboration**
  - Simplifies collaboration by ensuring all contributors follow the same process for code quality and deployment.
  - Ensures that pull requests are tested and validated before merging.

## Benefits of GitHub Actions

- **Built-In Integration**:
  - GitHub Actions is deeply integrated into GitHub, providing native support for issues, PRs, repositories, and events.
- **Scalability**:
  - GitHub Actions scales automatically, making it suitable for small projects and enterprise applications alike.
- **Wide Community Support**:
  - Extensive marketplace with thousands of reusable actions.
  - Allows customization of workflows using YAML configuration files.
- **Cost-Effective**:
  - Free for open-source projects and generous limits for private repositories.

## Key Features of GitHub Actions

- **Event-Driven**:
  - Trigger workflows based on events like commits, PRs, or even at scheduled times.
- **Custom Workflows**:
  - Define complex workflows using a simple YAML file.
  - Example: Trigger tests, build, and deploy when a new PR is opened.
- **Matrix Builds**:
  - Test across multiple environments (e.g., different Python versions) using a matrix strategy.

## Basic Workflow Example

```yaml
name: CI Workflow
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.9'
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run tests
        run: pytest
```

- **Explanation**:
  - Triggers on any push or pull request.
  - Runs a Python build job on Ubuntu, installs dependencies, and runs tests using `pytest`.

## Introduction to UV and GitHub Actions

- UV automates Python project management.
- GitHub Actions allows CI/CD automation directly in GitHub repositories.
- Using the **`setup-uv` action** simplifies Python project workflows in GitHub Actions.

### Installing UV in GitHub Actions

- The **`setup-uv` action** installs UV and adds it to the PATH.
- Pinning UV to a specific version is considered best practice.
- Example YAML configuration to install specific version (0.4.17) of UV:

```
name: Example
on: [push, pull_request]
jobs:
  uv-example:
    name: python
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Install uv
        uses: astral-sh/setup-uv@v2
        with:
          version: "0.4.17"
```

## Validating Action

- Navigate to GitHub page -> Actions
- Paste the above YAML code in the workflow file change name of file appropriately.
- Click Commit changes-
- In “Propose changes” page, click on “Create pull request”
- Click on GitHub page -> Actions
- Click on the workflow name to see the details of the workflow.

## Syncing and Running UV Projects

- UV syncs project dependencies using **`uv sync --all-extras --dev`**.

- You can run commands in the UV environment using `uv run`.
- Example workflow to sync and run tests with pytest:

```
steps:
  - name: Install the project
    run: uv sync --all-extras --dev
  - name: Run tests
    run: uv run pytest tests
```

Heres the complete workflow file:

```
name: Example
on: [push, pull_request]

jobs:
  uv-example:
    name: python
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Install uv
        uses: astral-sh/setup-uv@v2
        with:
          version: "0.4.17"
      - name: Install the project
        run: uv sync --all-extras --dev
      - name: Run tests
        run: uv run pytest tests
```

## Setting Up Python with UV

- UV can install Python using the `uv python install` command.
- The **matrix strategy** allows testing across multiple Python versions.
- Example setup for a matrix strategy:

```
strategy:
  matrix:
    python-version:
      - "3.10"
      - "3.11"
      - "3.12"
```

- Alternatively, the official `setup-python` action can be used, which is faster due to GitHub's caching.

Heres the complete workflow file:

```
name: Example
on: [push, pull_request]

jobs:
  uv-example:
    name: python
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version:
          - "3.10"
          - "3.11"
```

```
        - "3.12"
    steps:
      - uses: actions/checkout@v4
      - name: Install uv
        uses: astral-sh/setup-uv@v2
        with:
          version: "0.4.17"
      - name: Setup Python ${{ matrix.python-version }}
        run: uv python install ${{ matrix.python-version }}
      - name: Install the project
        run: uv sync --all-extras --dev
      - name: Run tests
        run: uv run pytest tests
```

## Demo

Running matrix of python versions