Week 2: Git + Python + Project Setup

CSI403 Full Stack Development | Lab 1 (8%)

Semester 1/2569

Agenda

What is Git?

Distributed Version Control System

- Track changes to your code over time
- Collaborate with team members
- Revert to previous versions if needed
- Branch for new features without affecting main code

Why Git?

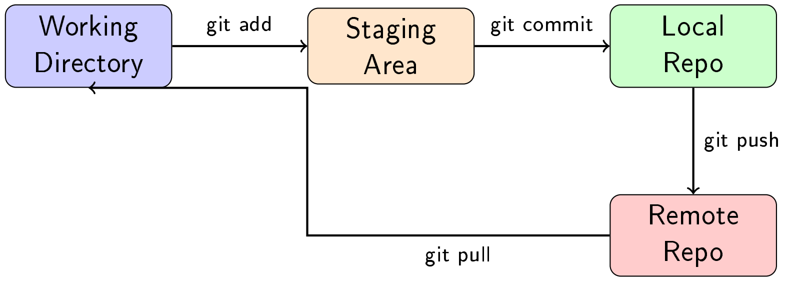
Industry Standard:

- Used by 90%+ companies
- Required for most dev jobs
- Essential for team work

Platforms:

- GitHub
- GitLab
- Bitbucket
- Azure DevOps

Git Workflow Overview



Git Workflow Steps

1. **Working Directory** – Your local files
2. **git add** – Stage changes you want to save
3. **Staging Area** – Changes ready to commit
4. **git commit** – Save staged changes locally
5. **Local Repository** – Your local Git history
6. **git push** – Upload to remote server
7. **git pull** – Download latest changes

Git Setup Commands

```
# Set your name
git config --global user.name "Your Name"

# Set your email
git config --global user.email "your@email.com"

# Verify settings
git config --list
```

Important: Use the same email as your GitHub account!

Clone Repository

```
# Clone a repository from GitHub
git clone https://github.com/username/taskflow.git

# Navigate to the project folder
cd taskflow

# Check the status
git status
```

Basic Git Commands

```
# Check status
git status

# Add all files to staging
git add .

# Add specific file
git add filename.py

# Commit with message
git commit -m "Add task API endpoints"

# Push to remote
git push origin main
```

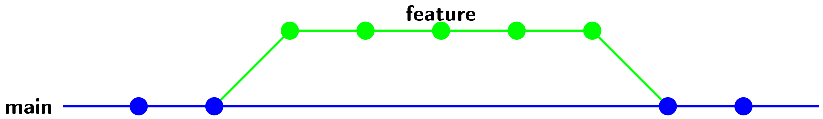
Pull Latest Changes

```
# Pull latest changes from remote
git pull origin main

# View commit history
git log --oneline

# View changes
git diff
```

Why Branching?



- Develop features without affecting main code
- Easy to test and review changes
- Merge when feature is complete

Branch Commands

```
# Create and switch to new branch
git checkout -b feature/task-api

# List all branches
git branch

# Switch to existing branch
git checkout main

# Delete branch (after merge)
git branch -d feature/task-api
```

Branch Workflow Example

```
# 1. Create feature branch
git checkout -b feature/task-api

# 2. Make changes and commit
git add .
git commit -m "Implement task CRUD"

# 3. Push branch to remote
git push -u origin feature/task-api

# 4. Create Pull Request on GitHub
# 5. After merge, switch back to main
git checkout main
git pull origin main
```

Python in This Course

We will use Python 3.11+

Key features:

- Type hints for better code quality
- Async/await for web servers
- Dataclasses for data models
- Modern f-string formatting

Variables and Type Hints

```python
# Without type hints
name = "TaskFlow"
version = 1.0
is_active = True

# With type hints (recommended)
name: str = "TaskFlow"
version: float = 1.0
is_active: bool = True
count: int = 42

# Optional type
description: str | None = None
```

Functions with Type Hints

```python
# Function with type hints
def greet(name: str) -> str:
    return f"Hello, {name}!"

# Function with optional parameter
def create_task(title: str, priority: str = "medium") -> dict:
    return {
        "title": title,
        "priority": priority
    }

# Call functions
message = greet("Student")
task = create_task("Complete Lab", "high")
```

Lists and Dictionaries

```python
# List
tasks: list[str] = ["Task 1", "Task 2", "Task 3"]

# Dictionary
task: dict = {
    "id": 1,
    "title": "Complete Lab",
    "status": "pending",
    "priority": "high"
}

# Access values
print(task["title"])  # "Complete Lab"
```

Python Classes

```
class Task:
    def __init__(self, id: int, title: str):
        self.id = id
        self.title = title
        self.status = "pending"

    def mark_done(self):
        self.status = "done"

# Usage
task = Task(1, "Complete Lab")
task.mark_done()
print(task.status)  # "done"
```

Dataclasses

```
from dataclasses import dataclass

@dataclass
class Task:
    id: int
    title: str
    status: str = "pending"
    priority: str = "medium"

# Usage - no need for __init__
task = Task(id=1, title="Complete Lab")
print(task.title)   # "Complete Lab"
print(task.status)  # "pending"
```

What is Virtual Environment?

Project A
FastAPI 0.109
Python 3.11

Project B
Django 4.2
Python 3.10

Project C
Flask 2.0
Python 3.9

- Isolated Python environment for each project
- Different dependencies won't conflict

Create Virtual Environment

```
# Create virtual environment named "venv"
python -m venv venv

# Activate on Windows
venv\Scripts\activate

# Activate on Mac/Linux
source venv/bin/activate

# You'll see (venv) in your terminal
```

Install Packages

```
# Install a package
pip install fastapi

# Install from requirements.txt
pip install -r requirements.txt

# Save current packages
pip freeze > requirements.txt

# Deactivate
deactivate
```

TaskFlow Project Structure

Main folders:

- `app/` – Main application code
- `app/models/` – SQLAlchemy database models
- `app/schemas/` – Pydantic validation schemas
- `app/routes/` – API endpoint handlers
- `app/templates/` – Jinja2 HTML templates
- `app/static/` – CSS, JavaScript, images
- `tests/` – Test files

requirements.txt

```
# Web Framework
fastapi==0.109.0
uvicorn==0.27.0

# Database
sqlalchemy==2.0.25
pyodbc==5.0.1

# Templates
jinja2==3.1.3

# Testing
pytest==7.4.4
```

app/config.py

```python
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    APP_NAME: str = "TaskFlow"
    APP_VERSION: str = "1.0.0"
    DEBUG: bool = True
    DATABASE_URL: str = "sqlite:///./taskflow.db"

    class Config:
        env_file = ".env"

settings = Settings()
```

app/main.py

```python
from fastapi import FastAPI
from app.config import settings

app = FastAPI(
    title=settings.APP_NAME,
    version=settings.APP_VERSION,
)

@app.get("/")
def root():
    return {"message": f"Welcome to {settings.APP_NAME}!"}

@app.get("/health")
def health():
    return {"status": "healthy"}
```

Run the Application

```
# Activate virtual environment
venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Run the server
uvicorn app.main:app --reload

# Open browser: http://localhost:8000
# API docs: http://localhost:8000/docs
```

Lab 1: Git + Python + Setup (8%)

Deliverables:

1. Create GitHub repository “taskflow”
2. Setup complete folder structure
3. Create requirements.txt with dependencies
4. Create app/config.py with settings
5. Create app/main.py with health endpoint
6. Use Git branching workflow
7. Write README.md documentation

Lab 1 Grading Rubric

Criteria	**Points**
GitHub repo + folder structure	2%
requirements.txt + virtual env	1%
config.py + main.py working	2%
Git workflow (branch + PR)	2%
README.md documentation	1%
Total	**8%**

Submission Instructions

How to Submit:

1. Push all code to GitHub
2. Make sure README.md is complete
3. Submit repository URL via Google Form

Deadline: Sunday 23:59

Late Policy: -10% per day (max 3 days)

Questions?

Let's start Lab 1!

Deadline: Sunday 23:59