DevOps CI/CD

Johnny Sylvain 11/3/2025

Things you should know about:

3rd biweekly due

SDD draft due

Reading: (Shore) Chapter 9: Developing

Activity

Project: Stand UP.

Stand up: 30 seconds each.

What have you done since your last meeting? (Could be coding software, writing requirements, working on design, delivering expensive furniture to your little old Korean aunt in Boston, attending the first annual Maine Symposium on Biomedicine)

What are your plans for the project?

- What are your immediate plans (for the next 2 days)?
- What are your medium-term plans (for the next 1-2 weeks)?

Core Idea:

Why is DevOps important?

Core Idea:

Why is DevOps important?

- Automation
- Version Control
- Find Bugs Early
- Scalability
- Collaboration

Continuous Delivery

Each push to main on github should be treated as functioning code. Won't always be the case, but use individual branches to your benefit.

NOT JUST FOR

ToolMaster/Version Control Master/Support Manager

TO KNOW!

Github Basics

Main

Working code only!
CI/CD executed here
Coding standards
Technically a branch

Push/Pull

Push - Send commit to repository

Pull - Fetch latest changes from repository

Merge

Combine branches

Conflicts must be resolved

Branch / Fork

Fork - create a copy of a repo to work on individually

Branch - Isolated environment in a repo from main

Actions

Create workflows for CI/CD

Workflow templates

Using Git with external server (SSH) 1

1) SSH into your server & generate a new SSH key for GitHub

```
ssh-keygen -t ed25519 -C "Example Remote Server" -f ~/.ssh/github_key
```

Generates two keys: ~/.ssh/github_key (private) and ~/.ssh/github_key.pub (public)

2) Since we're using a custom key name (github_key instead of the default), we need to tell SSH which key to use when connecting to GitHub. Create an SSH config file:

```
# Create or edit SSH config file

nano ~/.ssh/config
```

Add the following configuration:

```
Host github.com
   HostName github.com
   User git
   IdentityFile ~/.ssh/github_key
   IdentitiesOnly yes
```

Using Git with external server (SSH) 2

3) Display your public key & copy it

```
cat ~/.ssh/github_key.pub
```

4) Add your SSH key to github

Go to settings -> SSH and GPG keys (left nav bar)

Add new SSH key (name it and paste into key field)

5) Set your name and email for Git commits

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

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

# Verify your configuration

git config --global --list
```

Using Git with external server (SSH) 3

- Clone a repository using SSH
  - git clone git@github.com:username/repository-name.git
- Navigate into the cloned repository
  - cd repository-name
- Add file/folder contents to commit to git
  - git add . (add all contents of folder)
  - git commit -m "Initial commit"
- Set remote origin - Tell git to add a new remote connection
  - git remote add origin git@github.com:username/repo-name.git
- Push to main/branch
  - git push -u origin main
  - git push -u origin nameOfBranch

https://docs.github.com/

Continuous Integration

Testing as you push.

- Small commits (like the ones in your individual branches) - so big ones, like merge to main does not have major issues.
- Test types
- Hooks Git - Git Hooks
- Actions/Workflows

Tests

Software is in one of two states:

1. Performs the desired task + contains bugs.

2. Does not perform the desired task + contains bugs.

- Unit – single component.
- Integration – interaction between two or more components.
- System / end-to-end – evaluation of whole system, sometimes requires human in the loop (GUI testing).

Testing with containers / In the cloud

Docker / Singularity / Kubernetes

Configured in YAML

No magic, just physical machines in some other location.

• Choosing service, influenced by:

– Arbitrary company constraints.

– Where are the machines physically located.

– Limits on parallel execution.

– What resources does a single VM/container get (RAM, CPU,

GPU, compute time).

– Connect your machines to the cloud based service (self hosted

runner in GitHub speak). -> Self-hosted runners - GitHub Docs

Reality Check

- Lack of test failures does not indicate lack of faults.
- Probability of failures in code paths which are not exercised by any test is higher than for those which are.
- Early fault detection facilitates faster integration into the code base.
- Best effort subject to resource constraints (memory size, computation time, schedule etc.).
- Existence of failures does not always preclude deployment (cost-benefit analysis).

Git Hooks (briefly)

Triggers for custom scripts when important things happen

Client-side:
Pre-commit - Used to inspect the snapshot that's about to be committed. This is what your homework will use.
Prepare-commit-msg, commit-msg, Post-commit, post-merge

Server-side:
Pre-receive, update, Post-receive

Docs: Git - Git Hooks

LINTING

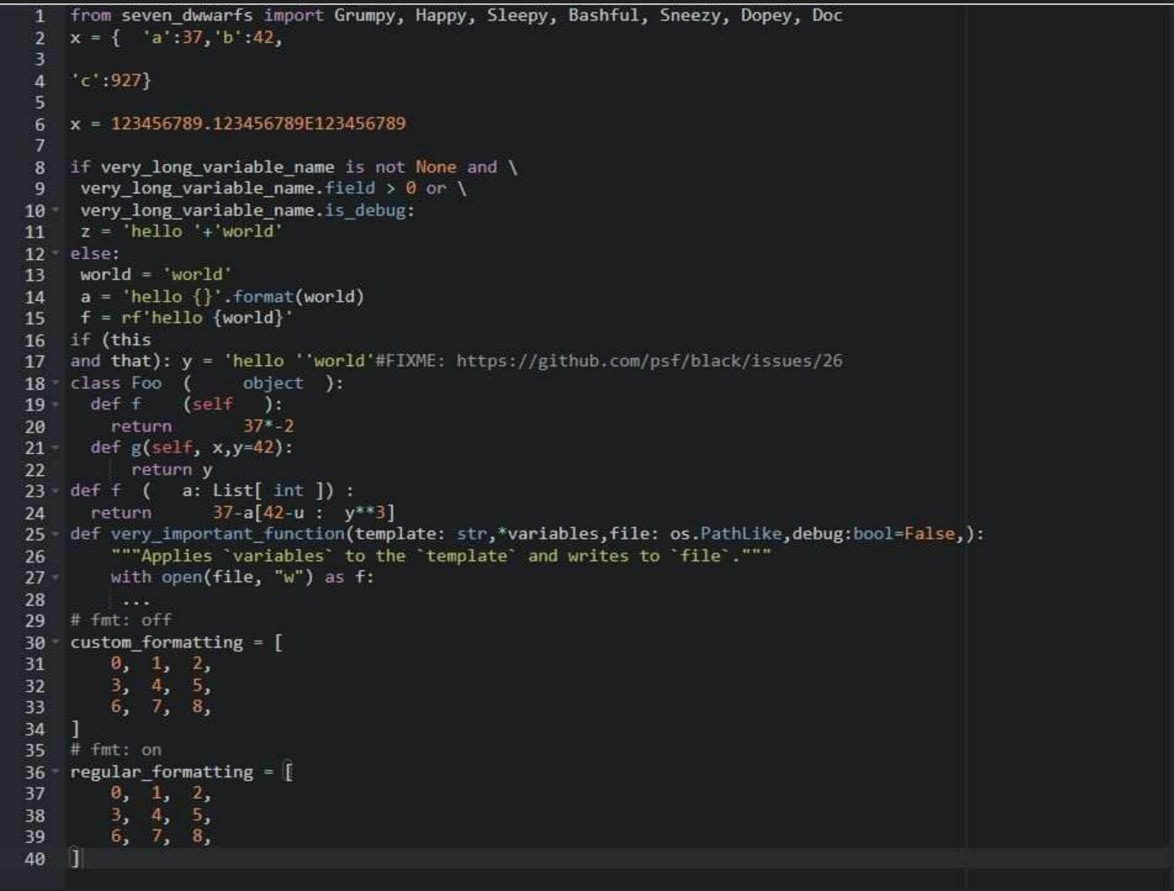
Ensures codebase is consistent and follows guidelines throughout projects.

Flake8 - Style Checker

Black - Code Formatter

VScode extensions

Black Code formatter - What are the rules?

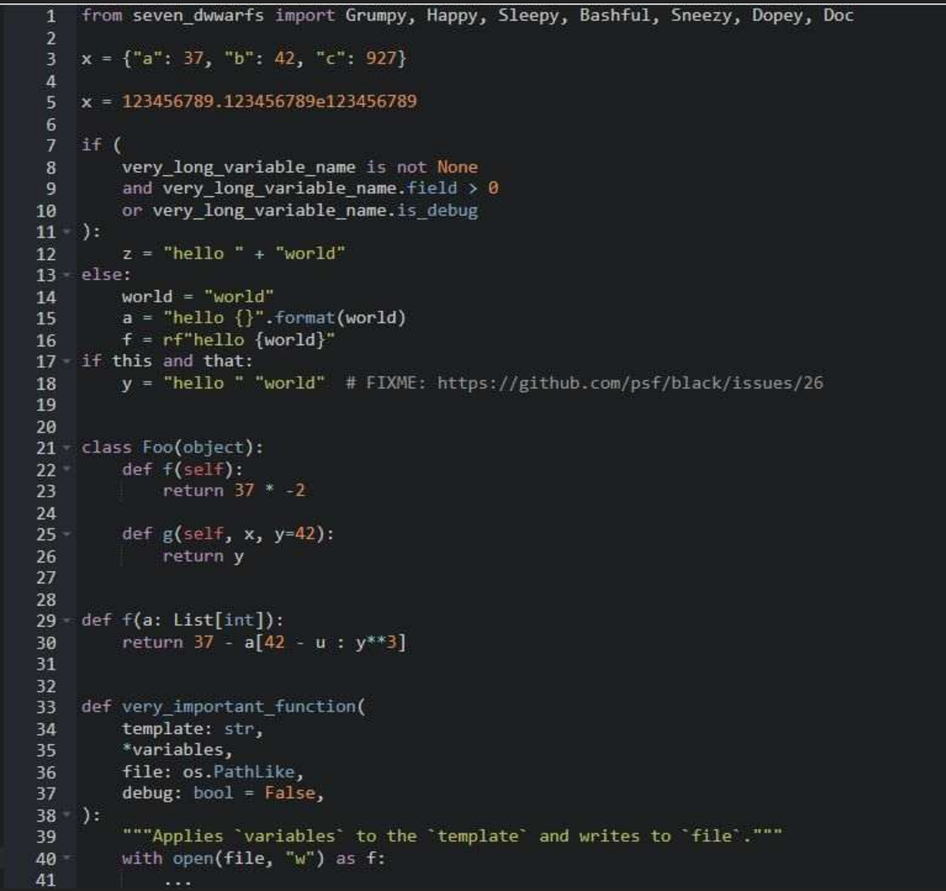


```python
from seven_dwwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezy, Dopey, Doc
x = {  'a':37,'b':42,

'c':927}

x = 123456789.123456789E123456789

if very_long_variable_name is not None and \
 very_long_variable_name.field > 0 or \
 very_long_variable_name.is_debug:
 z = 'hello '+'world'
else:
 world = 'world'
 a = 'hello {}'.format(world)
 f = rf'hello {world}'
if (this
and that): y = 'hello ''world'#FIXME: https://github.com/psf/black/issues/26
class Foo  (     object  ):
  def f    (self   ):
    return       37*-2
  def g(self, x,y=42):
      return y
def f  (   a: List[ int ]) :
  return      37-a[42-u :  y**3]
def very_important_function(template: str,*variables,file: os.PathLike,debug:bool=False,):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, "w") as f:
     ...
# fmt: off
custom_formatting = [
    0,  1,  2,
    3,  4,  5,
    6,  7,  8,
]
# fmt: on
regular_formatting = [
    0,  1,  2,
    3,  4,  5,
    6,  7,  8,
]
```



```python
from seven_dwwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezy, Dopey, Doc

x = {"a": 37, "b": 42, "c": 927}

x = 123456789.123456789e123456789

if (
    very_long_variable_name is not None
    and very_long_variable_name.field > 0
    or very_long_variable_name.is_debug
):
    z = "hello " + "world"
else:
    world = "world"
    a = "hello {}".format(world)
    f = rf"hello {world}"
if this and that:
    y = "hello " "world"  # FIXME: https://github.com/psf/black/issues/26


class Foo(object):
    def f(self):
        return 37 * -2

    def g(self, x, y=42):
        return y


def f(a: List[int]):
    return 37 - a[42 - u : y**3]


def very_important_function(
    template: str,
    *variables,
    file: os.PathLike,
    debug: bool = False,
):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, "w") as f:
        ...
```

BONUS ACTIVITY

5 minutes

What could have continuous integration in your project? What kind of tests would be helpful to ensure everything is working as intended?

Report out a test!

How does github actions work?

A workflow is a configurable automated process that will run one or more jobs.

- Manual
- Git hooks
- Scheduled Times
- Authentication with third party

Hosted on runners - Github owned servers (free [enough for our purposes] if your repo is public)

- You can host your own runners and connect them to github

Basic structure of YAML workflow

name: – Human-readable name for the workflow.

on: – Specifies the events that trigger the workflow (e.g., push, pull_request, schedule).

jobs: – A workflow can have multiple jobs that run in parallel or sequentially.

runs-on: – Defines the virtual environment (e.g., ubuntu-latest, windows-latest).

steps: – Each job contains steps that run commands or use actions.

Github Actions Demo

https://github.com/EndBug/actions-workshop

Additional Readings:

About Custom Actions - GitHub docs

Automatic Token Authentication - GitHub docs

Automate Your Workflow - On Microsoft Learn

Homework 5

Due on Nov 24, 2025 11:59 PM

Individual Grades - Team Submission
As a TEAM:
Create Repo (don’t forget to invite me john.sylvain@maine.edu)
Implement workflow with github actions, lint, and test 3 sorting algorithms.

As an Individual:
Submit a write up including a screen capture of your working workflow, your individual contribution and impressions of CI/CD, as well as a minimum 300 word design for a test case for your project:
What framework do you need, what OS/Systems will it be tested on, what cases are being considered, input/output, what is the speed needed to pass? Type of test? BE CREATIVE!
Coordinate with your team to have different test cases.

Decomposition of HW5 - 1

Repo Manager - Create Repo, share with john.sylvain@maine.edu, setup test_basic_sort.py to interface with int_sort.py, create sample test data & compare outputs of Dev 1-3’s algorithms as pytests

Lint & Style Checker - complete .pre-commit-config.yml (detect aws keys, limit maximal file size & lint using black and flake8) and ensure team members know the formatting rules and abide by them.

Dev 1 - Bubble sort: Measure CPU usage

Dev 2 - Quick sort: Measure runtime

Dev 3 - Insertion sort: Measure memory usage

Decomposition of HW5 - 2

All -

- Modify README to demonstrate the capabilities of your repo (overview, Installation instructions, usage guide)
- Table of results from tests on CPU and memory use and runtime differences between Operating Systems.
- Your group name and the contributions of each member.

Part 6 - Optional!! This is not for extra credit! If this is implemented and working, your entire assignment should also work.

Decomposition of HW5 - 3 - GRADING

60% Repo:

- Commit history (Did you do what you said you did?) - 15%
- Working workflows - 30%
- Table of test measurements - 10%
- README - 5%

40% Writeup:

- Individual contribution and impressions of CI/CD - 10%
- Design for a test case - 30%

Due on Nov 24, 2025 11:59 PM