Version Control System (VCS)

TEKsystems Global Services

WORKFORCE DEVELOPMENT

LOGISTICS

Class Hours:

- Instructor will set class start and end times.
- There will be regular breaks in class.

Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous:

- Courseware
- Bathroom
- Fire drills

VCS

Day 3 Learning Outcomes:

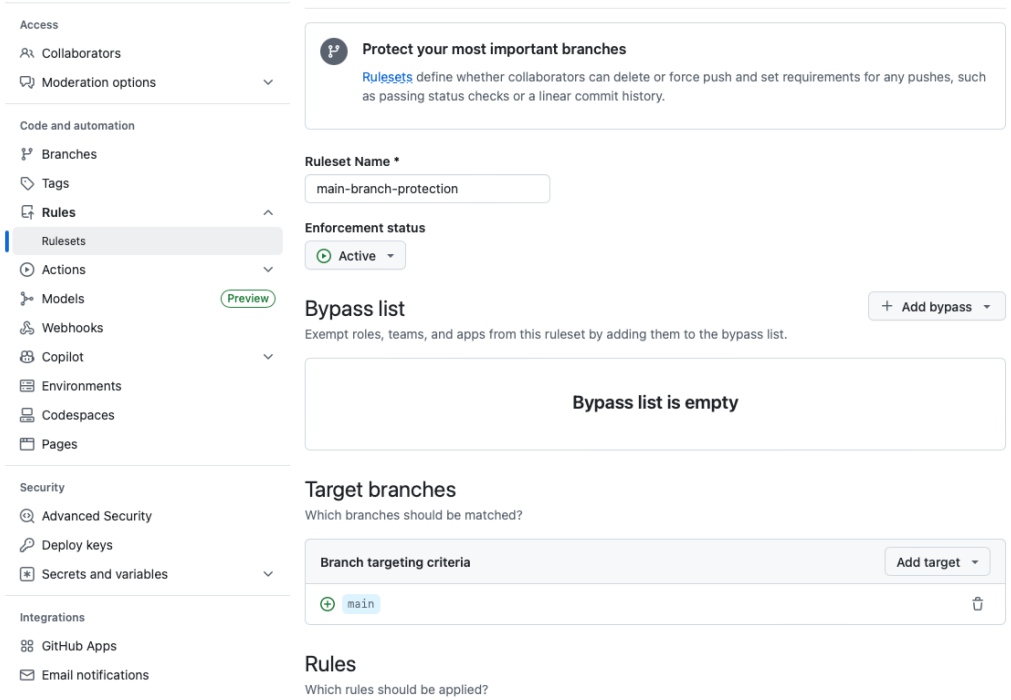
- Implement branch protection rules to safeguard the main branch
- Use forks to collaborate and contribute to repositories you don’t own
- Understand how Teams and Organizations streamline administration and access control
- Run, configure, and inspect workflows to see how CI/CD enhances version control and automates development

BRANCH PROTECTION

Branch protection rules help maintain the stability and integrity of important branches like main by controlling what actions are allowed. They prevent direct pushes, require reviews before merging, and ensure that code changes meet defined quality standards before being added to the main codebase.

By enforcing these rules, teams can reduce errors, maintain cleaner histories, and promote collaboration through Pull Requests and automated checks. Branch protection is an essential safeguard in professional workflows, ensuring that every change is reviewed and tested before deployment.

BRANCH PROTECTION



Branch protection rules define the conditions that must be met before changes can be merged into protected branches like main.

- Prevent direct pushes to the branch
- Require one or more approving reviews before merging
- Block merges with pending reviews or failed checks
- Require branches to be up to date with main before merging
- Restrict who can push or merge changes

BRANCH PROTECTION

Branch protection rules are most effective when you understand how they behave in real scenarios. Try enabling them on your own repositories and experiment with different settings.

Test actions like pushing directly to main, submitting Pull Requests, or merging without reviews to see how each rule responds. Hands-on testing helps you understand not just how to configure these rules, but why they're essential for maintaining a clean and reliable workflow.

LAB 3.1 – BRANCH PROTECTION

LAB INTRODUCTION

In this lab, you'll enable branch protection on your main branch and test how it affects collaboration.

You'll configure rules that require Pull Requests, enforce reviews, and block direct pushes — then experiment by trying actions that violate those rules.

By the end, you'll understand how branch protection ensures code quality, prevents accidental changes, and keeps the main branch stable during teamwork.

POP QUIZ: GIT CLI

What is the primary goal of enabling branch protection rules?

A. To allow faster direct pushes to main
B. To prevent unauthorized or unreviewed changes from being merged
C. To automatically delete inactive branches
D. To disable Pull Requests entirely

POP QUIZ: GIT CLI

What is the primary goal of enabling branch protection rules?

A. To allow faster direct pushes to main
B. To prevent unauthorized or unreviewed changes from being merged
C. To automatically delete inactive branches
D. To disable Pull Requests entirely

POP QUIZ: GIT CLI

Can an administrator push directly to a protected branch that is failing checks?

A. Yes, administrators are always exempt
B. No, branch protection applies to everyone equally
C. Only if they are included in a bypass list configured in the ruleset
D. Only if they created the repository

VCS

Forks

2

FORKS

A **fork** is your own copy of another person's repository that lives under your GitHub account. It lets you freely experiment with changes without affecting the original project.

Developers use forks to propose improvements or new features through pull requests. This enables open collaboration while keeping the original repository stable and protected.

FORKS

Many well-known software projects began as forks of existing ones. For example, **MariaDB** was forked from **MySQL** and **LibreOffice** came from **OpenOffice.**

Forking allows developers to explore new ideas or governance models without disrupting the original project. These forks often evolve into mature, widely adopted tools that shape the open-source ecosystem.

FORKS

When the original repository (often called the **upstream**) gets new commits, your fork won't automatically update. To stay current, you need to **sync your fork** so it includes the latest changes from the upstream project.

Regularly syncing prevents merge conflicts and ensures your contributions are based on the most recent code. This is especially important when working on active, fast-moving projects with multiple collaborators.

FORKS

When you fork a repository, you own that copy — it exists under your account and can't be changed or deleted by the original project's maintainers. You have full control to edit, rename, or make it private if you wish.

You can also delete your fork at any time without affecting the original repository. This independence makes forks ideal for experimentation, learning, and long-term contribution management.

FORKS: CASE STUDY

To propose changes to Python itself, contributors fork the PEPs repository on GitHub. Each Python Enhancement Proposal (PEP) describes a new feature or improvement to the language.

After making updates and syncing with the latest version, contributors submit a Pull Request (PR) for review. The Python community carefully evaluates each proposal to ensure it follows guidelines and aligns with the language's long-term design goals.

https://github.com/python/peps

LAB 3.2 – FORKS

LAB INTRODUCTION

In this lab, you will fork your partner's repository to create your own copy of their project. You'll then make a few small changes, commit them, and submit a Pull Request (PR) back to your partner.

This exercise will help you understand how collaboration works on GitHub. By the end, you'll have hands-on experience with forking, syncing, and reviewing pull requests just like in real-world open-source projects.

POP QUIZ: FORKS

Which of the following best describes a fork on GitHub?

A. A new branch created in the same repository
B. A copy of a repository under your own account
C. A clone of a repository to your local machine
D. A commit that merges two branches

POP QUIZ: FORKS

Which of the following best describes a fork on GitHub?

A. A new branch created in the same repository
B. A copy of a repository under your own account
C. A clone of a repository to your local machine
D. A commit that merges two branches

POP QUIZ: FORKS

When contributing to an open-source project, why might you use a fork instead of a branch?

A. Forks allow full control and prevent changes to the original project
B. Forks automatically sync all branches without conflict
C. Branches can only be created by administrators
D. Forks merge automatically without a pull request

POP QUIZ: FORKS

When contributing to an open-source project, why might you use a fork instead of a branch?

A. Forks allow full control and prevent changes to the original project
B. Forks automatically sync all branches without conflict
C. Branches can only be created by administrators
D. Forks merge automatically without a pull request

VCS

Orgs and Teams

2

ORGS AND TEAMS

Organizations on GitHub provide a shared space where multiple people can collaborate on repositories under a single name. They make it easier to manage access, roles, and repositories for groups, departments, or entire companies.

Within an organization, you can create **teams** to organize members by function or project. Teams simplify permissions and communication, allowing you to assign access levels, mention groups in discussions, and manage contributions efficiently.

ORGS AND TEAMS

Teams help organize members within an organization and streamline collaboration. Each team can have its own access level, communication channel, and responsibilities.

- Teams can be nested to reflect real reporting or project structures
- Each team can be granted specific repository permissions (read, write, admin)
- You can mention entire teams in issues or pull requests using @team-name
- Team discussions provide a private space for internal communication
- Permissions and membership are managed centrally by organization admins

ORGS AND TEAMS

Organizations provide global controls that apply across all repositories and teams. These settings help maintain security, compliance, and consistent collaboration standards.

- Default repository permissions (none, read, or write for new members)
- Member privileges such as allowing or restricting repository creation
- Security policies like enforced 2FA or SSO for all members

- Default branch protection rules that apply to all repositories
- Billing and usage insights managed from a single dashboard
- App and integration management for tools like CI/CD, Webhooks, or issue tracking

PERSONAL ACCESS TOKEN (PAT)

A **Personal Access Token (PAT)** is an alternative to using your GitHub password when accessing repositories through the command line or external tools. It acts like a digital key that authenticates you securely with GitHub's API.

PATs can be scoped to allow specific actions, such as reading repositories, pushing code, or managing workflows. They should be kept secret, stored securely, and revoked immediately if exposed or no longer needed.

To create a PAT, go to settings, then developer settings.

LAB 3.3 – ORGS AND TEAMS

LAB INTRODUCTION

In this lab, you'll explore how organizations and teams are structured and managed on GitHub. You'll visit a public organization such as Docker to see how repositories, teams, and settings are organized in real projects.

You'll also review important organization-level settings, including Personal Access Tokens (PATs) and security options. This exercise will help you understand how larger teams manage access, permissions, and collaboration at scale.

POP QUIZ: ORGS AND TEAMS

What does **PAT** stand for in GitHub?

A. Public Access Tool
B. Personal Access Token
C. Private Authentication Type
D. Project Access Template

POP QUIZ: ORGS AND TEAMS

What does **PAT** stand for in GitHub?

A. Public Access Tool
B. Personal Access Token
C. Personal Authentication Type
D. Project Access Template

POP QUIZ: ORGS AND TEAMS

What is the main advantage of using teams within a GitHub organization?

A. Teams automatically manage billing for repositories
B. Teams group members to assign permissions and communicate easily
C. Teams allow users to create personal forks of private repositories
D. Teams automatically merge pull requests from multiple members

POP QUIZ: ORGS AND TEAMS

What is the main advantage of using teams within a GitHub organization?

A. Teams automatically manage billing for repositories
B. Teams group members to assign permissions and communicate easily
C. Teams allow users to create personal forks of private repositories
D. Teams automatically merge pull requests from multiple members

VCS

Workflows

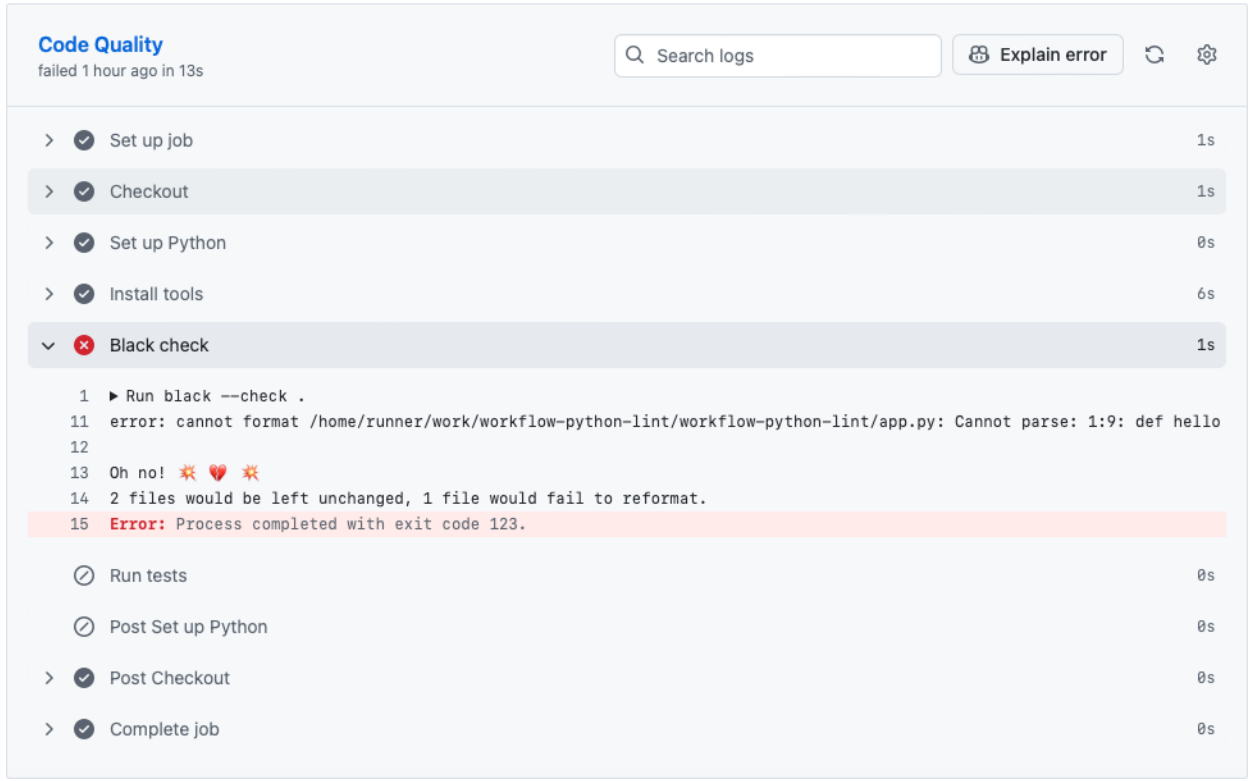
2

WORKFLOWS

A **workflow** in GitHub Actions is an automated process that runs one or more jobs based on specific triggers. It lives in a YAML file under .github/workflows/ and defines when and how your automation should execute. For example, when code is pushed, a pull request is opened, or a manual action is triggered.

Workflows help automate repetitive tasks like testing, building, and deploying code. Each workflow can include multiple steps that run commands, scripts, or reusable actions, allowing teams to enforce quality checks and streamline development from commit to deployment.

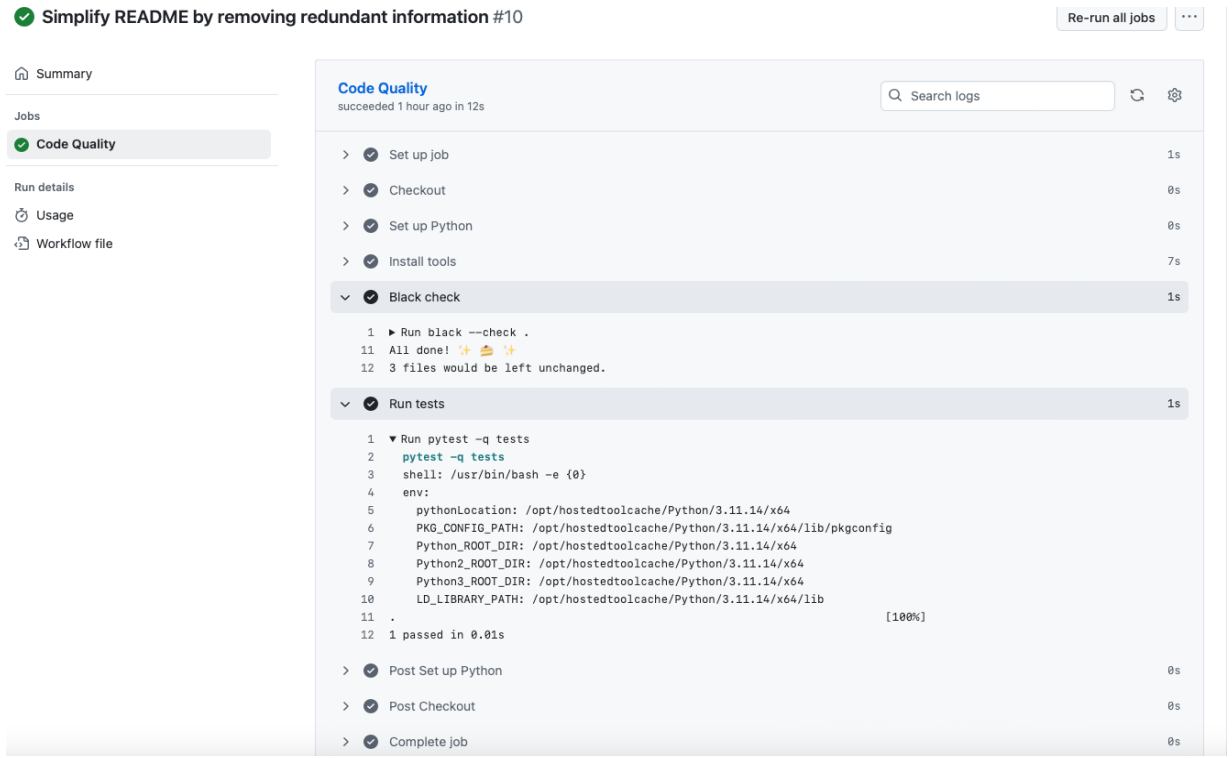
WORKFLOWS



Consider this workflow. It runs a lint check using Black and then executes unit tests on a small Python project. The failed run shows that the code did not meet the formatting rules defined by Black.

This same workflow can be configured in branch protection rules so that it must pass before any pull request can be merged into the main branch.

WORKFLOWS



Here the workflow passes both the Black lint check and the unit tests. Each step completes successfully, and GitHub marks the run as green.

A passing run means the code meets the project's style and testing standards. When this workflow is required by branch protection rules, it confirms the code is ready to merge safely into the main branch.

WORKFLOWS

```yaml
- name: Set up Python
  uses: actions/setup-python@v5
  with:
    python-version: "3.11"

- name: Install tools
  run: |
    python -m pip install --upgrade pip
    pip install black pytest
```

Notice the step named "Set up Python" does not make use of the "run" label. Instead, it has the "uses" label, which imports an action. We simply set python-version label to 3.11, and the action does the work to install Python. You can always review the docs for any action to see how to use it.

The step named "Install tools" Does not make use of an action. It simply runs shell commands on the OS running the entire workflow.

WORKFLOWS

Triggers: Workflows start automatically from events such as pushes, pull requests, or schedules, or can be launched manually.

Jobs and concurrency: Each workflow can have multiple jobs that run in parallel or depend on one another. This allows testing or building across different environments.

Runners: Workflows execute on runners, which are virtual machines provided by GitHub or self-hosted ones that you control.

Artifacts and caching: Workflows can save or reuse files between runs, improving performance for tasks like dependency installs.

Visibility and logs: Every step's output appears in the Actions tab, giving clear feedback for debugging or verification.

LAB 4.4 – WORKFLOW

LAB INTRODUCTION

In this lab, you will fork a repository that contains a simple Python workflow designed to validate code formatting and run tests. The workflow uses tools like Black and pytest to ensure consistency and quality.

You will experiment with branch protection rules, explore how workflows can be enforced before merging, and practice running the workflow both manually and through pull requests.

POP QUIZ: GITHUB

What must you do before a teammate can push changes to your GitHub repository?

A. Make the repository public
B. Add them as a collaborator
C. Share your Git credentials
D. Fork the repository

VCS DAY 3 COMPLETE

You've successfully completed the GitHub Collaboration and Workflows module. You learned how to fork repositories, create pull requests, manage teams and organizations, and enforce quality with automated workflows and branch protection rules.

These are the same tools and practices used in real-world DevOps and software engineering environments. Keep experimenting, keep collaborating, and continue building your confidence as you work with Git and GitHub. Great job!