PW SKILLS | DevOps and Cloud Computing

Introduction to Version Control and Git Fundamentals

Objective

- Understand the purpose and types of version control systems.
- Learn the basics of Git, including commits, branches, and merging.
- Execute basic Git commands for repository management.
- Understand Git workflows and the lifecycle of changes.

PW SKILLS

Importance of version control in collaborative development.

Importance of Version Control

Version control is a critical component in collaborative software development, providing essential tools and processes that enhance teamwork and project management. Here are the key aspects highlighting its importance:

Key Aspects of Version Control in Collaborative Development:

1. Change tracking
2. Concurrent Development
3. Error Recovery
4. Enhanced Collaboration Tools
5. Improved Communication
6. Quality Assurance

PW SKILLS

Different types of version control systems

Local version control (e.g., RCS).

Local version control systems (VCS), such as RCS (Revision Control System), are foundational tools in software development that help manage changes to files on a local machine. Here's an overview of their importance and functionality:

Importance of Local Version Control:

1. Simplicity
2. Change Tracking
3. Offline Access
4. Low Overhead

Local version control (e.g., RCS).

Challenges of Local Version Control:

While local version control systems offer several advantages, they also come with notable limitations:

- Single Point of Failure
- Collaboration Difficulties
- Limited Scalability

Pop Quiz

Q. Which of the following best describes the primary function of RCS?

A

To automate the building of software

B

To manage revisions of text documents and source code

Pop Quiz

Q. Which of the following best describes the primary function of RCS?

A

To automate the building of software

B

To manage revisions of text documents and source code

Centralized version control (e.g., SVN).

Centralized version control systems (CVCS), such as Subversion (SVN), play a crucial role in managing code changes and facilitating collaboration among software development teams. Here's an overview of their key features, benefits, and workflows:

Key Features of Centralized Version Control Systems:

- Client-Server Model
- Single Repository
- File Locking

Centralized version control (e.g., SVN).

Benefits of Using Centralized Version Control:

- Simplified Collaboration
- Version History
- Ease of Use

Typical Workflow in Centralized Version Control:

- Pull Latest Changes
- Make Changes
- Commit Changes
- Resolve Conflicts

Pop Quiz

Q. Which of the following describes the architecture of SVN?

A

Client-server model

B

Distributed model

Pop Quiz

Q. Which of the following describes the architecture of SVN?

A

Client-server model

B

Distributed model

Distributed version control (e.g., Git).

Distributed version control systems (DVCS), such as Git, are essential tools in modern software development, allowing multiple developers to collaborate efficiently. Here's an overview of their key features, advantages, and common practices.

Key Features of Distributed Version Control Systems:

- Local Repository
- Peer-to-Peer Model
- Branching and Merging

Distributed version control (e.g., Git).

Advantages of Distributed Version Control:

- Offline Work
- Speed
- Data Redundancy
- Enhanced Collaboration

Common Practices in Distributed Version Control:

1. Cloning Repositories
2. Committing Changes Locally
3. Pushing Changes
4. Pulling Updates
5. Resolving Conflicts

Pop Quiz

Q. What type of version control system is Git?

A

Distributed version control system

B

Local version control system

Pop Quiz

Q. What type of version control system is Git?

A

Distributed version control system

B

Local version control system

PW SKILLS

Why Git has become the industry standard for version control

Why Git is the Industry Standard

Git has become the industry standard for version control due to several compelling factors that enhance its functionality, flexibility, and integration into modern development practices.

Here are the key reasons for Git's dominance:

Why Git is the Industry Standard

Key Reasons Why Git is the Industry Standard:

- Distributed Architecture
- Speed and Efficiency
- Powerful Branching and Merging
- Collaboration and Community Support
- Open Source and Free
- Flexibility in Workflows

PW SKILLS

Git and its role in tracking changes in projects

Git & its role

Git is an open-source distributed version control system widely used in software development for tracking changes in projects.

- Its primary role is to manage and record modifications to files, allowing teams to collaborate efficiently while maintaining a complete history of their work.

Key Roles of Git in Tracking Changes:

- Version History
- Local Repositories
- Branching and Merging
- Reverting Changes
- Collaboration

PW SKILLS

Explain key concepts: Commits, Branches & Merging

Commits: How Git saves snapshots of changes.

The core functionality of Git revolves around the concept of commits, which serve as snapshots of the project at specific points in time.

How Git Saves Snapshots of Changes:

1. Make Changes to Files
2. Stage Changes:

```
git add <file>
```

Commits: How Git saves snapshots of changes.

3. Create a Commit:

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

4. Unique Identifier: Each commit is assigned a unique SHA-1 hash that identifies it.

5. View Commit History:

```
git log
```

6. Reverting Changes:

```
git checkout <commit_hash>
```

7. Push Changes (if applicable):

```
git push
```

Pop Quiz

Q. What is a commit in Git?

A

A command to push changes to a remote repository

B

A snapshot of the repository at a specific point in time

Pop Quiz

Q. What is a commit in Git?

A

A command to push changes to a remote repository

B

A snapshot of the repository at a specific point in time

Branches: Isolating work streams for parallel development.

In Git, a branch is essentially a pointer to a specific commit in the repository's history. Each branch represents an independent line of development, allowing developers to work on features, bug fixes, or experiments without affecting the main project.

Isolation of Workstreams: By creating separate branches for different tasks (e.g., feature/login, bugfix/header), developers can isolate their changes.

- This isolation ensures that ongoing work does not disrupt the stability of the main branch (often called main or master).

Branches: Isolating work streams for parallel development.

Benefits of Using Branches for Parallel Development:

1. Enhanced Collaboration
2. Experimentation
3. Simplified Code Review
4. Organized Workflow
5. Clear Release Management

Pop Quiz

Q. What is the primary purpose of branching in Git?

A

To work on different features or changes in isolation

B

To create backups of the repository

Pop Quiz

Q. What is the primary purpose of branching in Git?

A

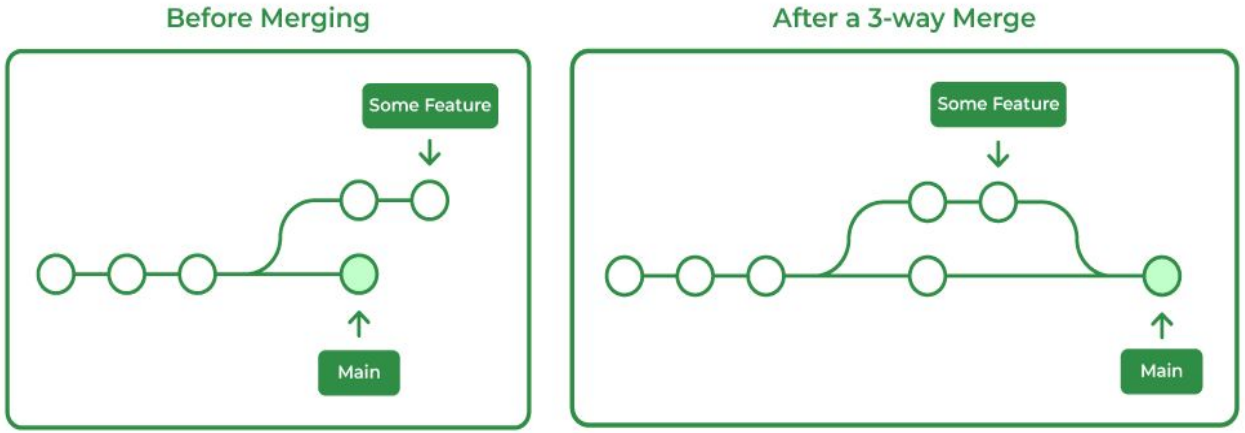
To work on different features or changes in isolation

B

To create backups of the repository

Merging: Combining changes from different branches

Merging is the process of taking the independent lines of development created by branching and integrating them into a single branch. This ensures that all contributions are combined, allowing the project to progress cohesively.



Merging: Combining changes from different branches

Types of Merges:

1. Fast-Forward Merge:
This occurs when the branch being merged has all its commits ahead of the current branch, with no divergent changes. In this case, Git simply moves the pointer of the current branch forward to the latest commit of the feature branch.

2. Three-Way Merge:
This occurs when both branches have diverged, meaning there are commits on both branches since they split from their common ancestor. Git creates a new merge commit that combines changes from both branches.

PW SKILLS

Simple visual diagrams to explain branching and merging workflows.

Visual Diagrams

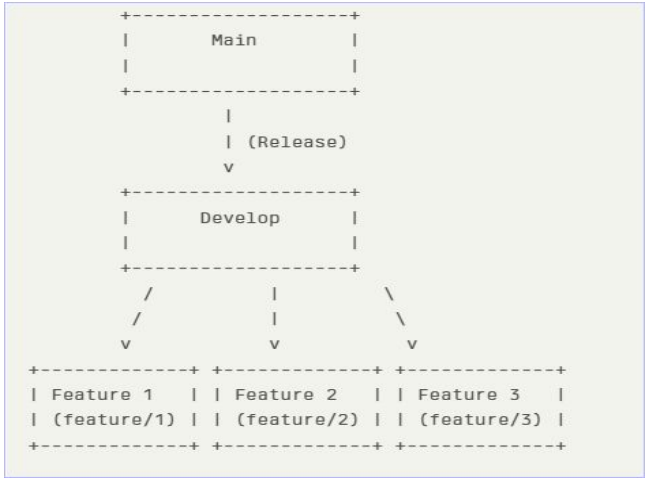
To illustrate branching and merging workflows in Git, let's use simple visual diagrams that represent common practices.

1. **GitFlow Workflow Diagram:**

Main Branch: Holds the production-ready code.

Develop Branch: Serves as an integration branch for features.

Feature Branches: Individual branches created for each new feature being developed.

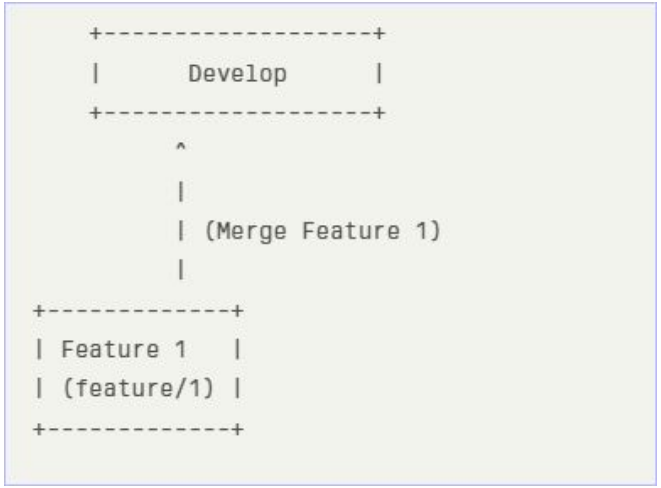


Visual Diagrams

Merging Process in GitFlow:

When a feature is complete, it is merged back into the develop branch:

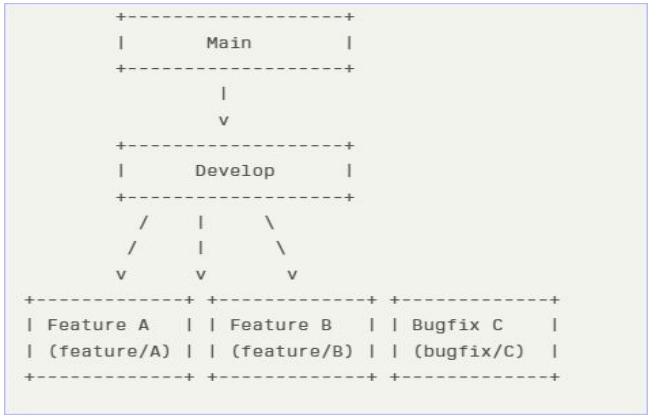
- Once all features are complete and tested, the develop branch can be merged into the main branch for release.



Visual Diagrams

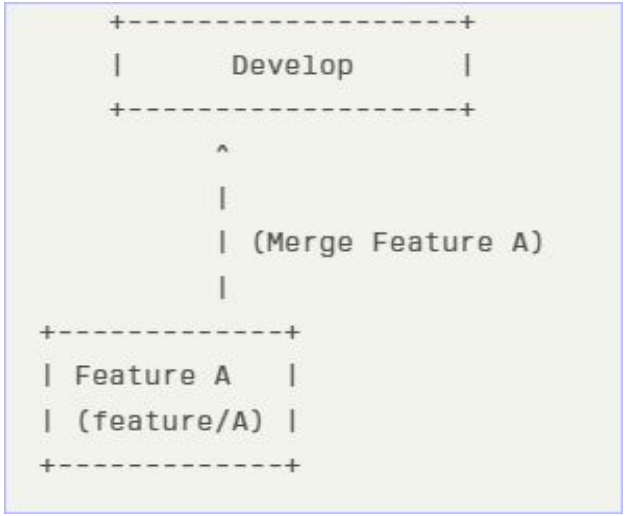
2. Feature Branch Workflow Diagram:

In the Feature Branch Workflow, developers create a new branch for each feature or bug fix. Here's a visual representation:



Visual Diagrams

Merging Process in Feature Branch Workflow:
When a feature or bug fix is complete, it is merged back into the develop branch:



Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

Demonstrating repository initialization and cloning

Let’s do it

1. **Initializing a Repository with ‘git init’:**

To create a new Git repository, follow these steps:

1. Open terminal
2. Navigate to Your Project Directory:

```
cd path/to/your-project
```

3. Initialize the Repository:

```
git init
```

4. Verify Initialization:

```
ls -a
```

You should see the .git folder listed.

Let’s do it

2. Cloning a Repository with git clone:

To create a local copy of an existing remote repository, use the git clone command:

1. Open Terminal

2. Run the Clone Command:

```
git clone <repository-url>
```

3. Navigate into the Cloned Directory:

```
cd repo
```

Pop Quiz

Q. What happens when you run the ‘git init’ command in a directory?

A

It initializes a new Git repository and creates a .git folder.

B

It creates a new remote repository.

Pop Quiz

Q. What happens when you run the ‘git init’ command in a directory?

A

It initializes a new Git repository and creates a .git folder.

B

It creates a new remote repository.

PW SKILLS

How to check the repository status with git status

Git status

To check the status of a Git repository, you can use the ‘git status’ command. This command provides a comprehensive overview of the current state of your working directory and staging area. Here's how to use it effectively:

1. Open your terminal
2. Navigate to Your Repository
3. Run the Command: ‘git status’

Git status

Variants of ‘git status’:

Short Format: To view a more concise output, you can use:

```
git status -s
```

Verbose Mode: For detailed information, including the textual changes in uncommitted files, use:

```
git status -v
```

Pop Quiz

Q. What information does ‘git status’ provide?

A

List of changed files that are staged, unstaged, and untracked

B

List of all branches in the repository

Pop Quiz

Q. What information does ‘git status’ provide?

A

List of changed files that are staged, unstaged, and untracked

B

List of all branches in the repository

PW SKILLS

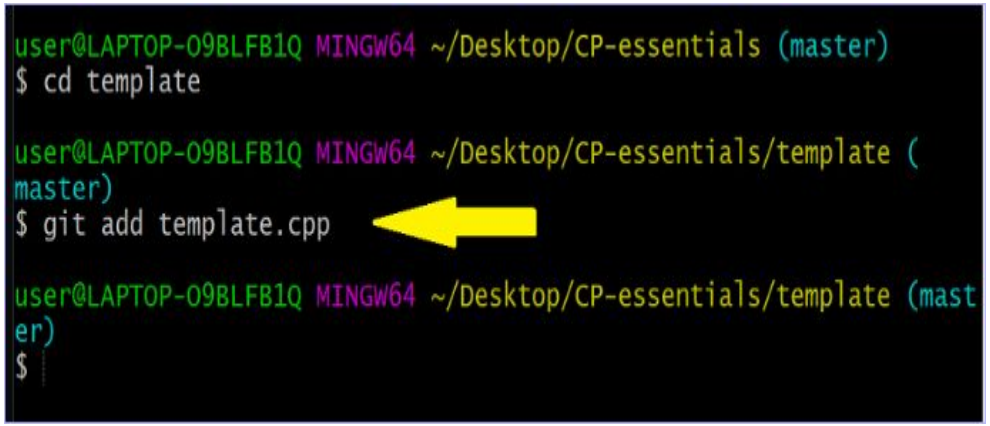
Adding and Committing changes

‘git add’ to stage changes

To stage changes in a Git repository using the git add command, follow these steps:

1. Open your terminal
2. Navigate to Your Repository
3. Run the Command:

```
git add <filename>
```



‘git add’ to stage changes

Staging Multiple Files:

- To stage all changes in the current directory and its subdirectories, use:

```
git add .
```

- Alternatively, you can use:

```
git add -A
```

This stages all changes across the entire repository, including deletions.

‘git commit’ to save changes with meaningful messages.

To save changes in a Git repository, you use the ‘git commit’ command along with a meaningful message. This process captures a snapshot of your staged changes, allowing you to keep track of your project's history.

Steps to Commit Changes:

1. **Stage Your Changes:** Before committing, ensure that you have staged your changes using ‘git add’. For example:

```
git add <filename>
```

or to stage all changes:

```
git add .
```

‘git commit’ to save changes with meaningful messages.

2. **Run the Commit Command: Use the following command to commit your changes with a message:**

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

Example

Here’s an example of how to commit changes:

```
git add index.html
git commit -m "Updated index.html to include new features"
```

Pop Quiz

Q. What happens to a file after you run git add <filename>?

A

It becomes committed.

B

It becomes staged.

Pop Quiz

Q. What happens to a file after you run git add <filename>?

A

It becomes committed.

B

It becomes staged.

Pop Quiz

Q. How do you supply a commit message when committing changes?

A

git commit -m "Commit message".

B

git commit "Commit message"

Pop Quiz

Q. How do you supply a commit message when committing changes?

A

git commit -m "Commit message".

B

git commit "Commit message"

PW SKILLS

Introducing branching commands

Creating new branches (git branch).

To create new branches in Git, you can use the git branch command. This allows you to work on features or fixes in isolation without affecting the main codebase.

Creating New Branches with Git:

- Basic Command: To create a new branch, use:

```
git branch <branchname>
```

Creating new branches (git branch).

2. **Creating from a Specific Commit or Tag:** If you want to create a branch from a specific commit or tag, you can do so with:

```
git branch <branchname> <commit_id>
```

or

```
git branch <branchname> <tagname>
```

Switching branches (git checkout or git switch).

To switch branches in Git, you can use either the ‘git checkout’ command or the newer ‘git switch’ command. Here’s a brief overview of how to use both:

Using git checkout:

1. Switch to an Existing Branch:

```
git checkout <branch_name>
```

2. Create and Switch to a New Branch:

```
git checkout -b <new_branch_name>
```

This command creates a new branch and immediately switches to it.

Switching branches (git checkout or git switch).

PW SKILLS

Using git switch:

1. Switch to an Existing Branch:

```
git switch <branch_name>
```

2. Create and Switch to a New Branch:

```
git switch -c <new_branch_name>
```

Pop Quiz

Q. How do you create a new branch and switch to it in one command?

A

git checkout -b <branch_name>

B

git branch -c <branch_name>

Pop Quiz

Q. How do you create a new branch and switch to it in one command?

A

git checkout -b <branch_name>

B

git branch -c <branch_name>

PW SKILLS

Demonstrating merging branches using ‘git merge’

Let's do it.

1. Checkout the Target Branch:

```
git checkout main
```

2. Pull the Latest Changes:

```
git pull origin main
```

3. Merge the Feature Branch:

```
git merge feature/new-ui
```

Let's do it.

4. Resolve Conflicts (If Any):

```
git add <filename>
```

5. Commit the Merge:

```
git commit -m "Merged feature/new-ui into main"
```

6. Push the Changes:

```
git push origin main
```

Pop Quiz

Q. If you are on the main branch and want to merge a branch named feature-xyz, what command would you use?

A

git merge feature-xyz.

B

git merge main feature-xyz

Pop Quiz

Q. If you are on the main branch and want to merge a branch named feature-xyz, what command would you use?

A

git merge feature-xyz.

B

git merge main feature-xyz

PW SKILLS

The typical Git workflow

Making changes → Staging → Committing → Pushing.

The typical Git workflow consists of a series of steps that developers follow to manage changes in their codebase effectively. Here's a concise overview of the process:

1. **Making Changes:**

Developers modify files in their working directory. This can involve creating new files, editing existing ones, or deleting files as necessary.

2. **Staging:**

After making changes, the next step is to stage these changes using the git add command.

```
git add <filename>
```

Making changes → Staging → Committing → Pushing.

To stage all changes at once, you can use:

```
git add .
```

3. Committing:

Once the changes are staged, developers commit them to the repository with the 'git commit' command.

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

4. Pushing:

Finally, after committing changes locally, developers push their commits to a remote repository using the 'git push' command.

```
git push origin <branch_name>
```

PW SKILLS

Collaboration workflows like feature branches and pull requests

Collaboration workflows

Collaboration workflows in Git are essential for managing team development effectively.

Feature Branches:

- **Definition:** A feature branch is a separate branch created for developing a specific feature or fix. This allows developers to work independently without affecting the main codebase.
- Creation
- Development
- Lifespan

Collaboration workflows

Pull Requests:

Definition: A pull request (PR) is a request to merge changes from a feature branch into the main branch. It serves as a way for developers to propose their changes for review.

- Review Process
- Integration

Collaboration workflows

Benefits of Using Feature Branches and Pull Requests:

1. Isolation of work
2. Code Review
3. Continuous Integration
4. Clear History

PW SKILLS

Best practices, including writing clear commit messages and resolving merge conflicts.

Best Practices

Here are some best practices for using Git effectively, focusing on writing clear commit messages and resolving merge conflicts:

1. Writing Clear Commit Messages:

- Be descriptive
- Use a Standard Format:

```
[Type] Short description (50 characters or less)

Detailed description (optional, wrapped at 72 characters)
```

- Reference Issues:
  Example: "Fix login bug (#123)"

Best Practices

2. Resolving Merge Conflicts:

Understand the conflict
Use a merge tool
Test after resolving
Commit the resolved changes

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistants

Thanks !