Technical Training on Version Control

Introduction to Version Control

What is Version Control?

- Version control is a system that records changes to files over time so that you can recall specific versions later.
- It allows multiple people to work on a project simultaneously without overwriting each other's work.

Why Version Control is Important:

- **Collaboration:** Multiple developers can work on the same project simultaneously.
- **History:** Keeps a detailed history of changes, making it easier to revert to previous versions.
- **Backup:** Provides a backup of your project and changes.
- **Branching and Merging:** Facilitates working on different features or fixes simultaneously without affecting the main codebase.

Types of Version Control Systems

Local Version Control Systems:

- Example: RCS (Revision Control System).
- **How it works:** Maintains versions of files on a local machine. Simple, but not suitable for collaboration.

Centralized Version Control Systems (CVCS):

- Examples: SVN (Subversion), CVS (Concurrent Versions System).
- **How it works:** All versioned files are stored on a single server. Developers check out files, make changes, and then check them back in.

Distributed Version Control Systems (DVCS):

- Examples: Git, BitBucket.
- **How it works:** Every developer has a local copy of the entire project history, allowing for greater flexibility, speed, and resilience against server failures.

Introduction to Git

What is Git?

- Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- Created by Linus Torvalds in 2005 to support the development of the Linux kernel.

Key Features of Git

- **Branching and Merging:** Git allows for flexible branching and merging, making it easy to manage multiple lines of development.
- **Lightweight:** Branches in Git are lightweight, making them cheap and easy to create.
- **Staging Area:** Git has a staging area where changes can be reviewed and edited before committing to the repository.
- **Distributed:** Every clone of a Git repository contains the full history, allowing for offline work and improved redundancy.

Core Concepts in Git

Repository:

- A repository is a directory or storage space where your project files and the entire revision history are stored.
- **Example:** Creating a new repository using `git init` or cloning an existing repository using `git clone`.

Commit:

- A commit is a snapshot of your project at a specific point in time.
- **Example:** After making changes to files, you create a commit using `git commit -m "Your commit message"`.

Branch:

- A branch is a pointer to a particular commit, allowing you to diverge from the main line of development and continue to work without affecting the primary codebase.
- **Example:** Creating a new branch using `git branch feature-branch` and switching to it with `git checkout feature-branch`.

Merge:

- Merging is the process of combining changes from different branches into a single branch.
- **Example:** Merging a feature branch into the main branch using `git merge feature-branch`.

Staging Area (Index):

- The staging area is where you can prepare a snapshot of changes to be committed. It allows you to review and modify what will be included in the next commit.
- **Example:** Adding changes to the staging area using `git add filename`.

Remote Repository:

- A remote repository is a version of your project hosted on the internet or another network. You can push or pull changes to/from a remote repository.
- **Example:** Connecting to a remote repository using `git remote add origin URL`.

Pull/Push:

- **Pull:** Fetching and integrating changes from a remote repository to your local repository.
- **Push:** Sending your committed changes to a remote repository.
- **Example:** Using `git pull origin main` to update your local branch and `git push origin main` to upload your changes.

Clone:

- Cloning is copying a remote repository to your local machine.
- **Example:** `git clone https://github.com/user/repository.git`.

Basic Git Commands

Initializing a Repository:

- `git init`: Initializes a new Git repository.

Checking the Status of Your Files:

- `git status`: Shows the status of your working directory and staging area.

Adding Changes to the Staging Area:

- `git add filename`: Adds a specific file to the staging area.
- `git add .`: Adds all changes in the current directory to the staging area.

Committing Changes:

- `git commit -m "Your commit message"`: Commits the staged changes with a descriptive message.

Viewing Commit History:

- `git log`: Displays the commit history.

Branching and Merging:

- `git branch branch-name`: Creates a new branch.
- `git checkout branch-name`: Switches to the specified branch.
- `git merge branch-name`: Merges the specified branch into the current branch.

Working with Remotes:

- `git remote add origin URL`: Adds a remote repository.
- `git push origin branch-name`: Pushes your changes to the remote repository.
- `git pull origin branch-name`: Pulls changes from the remote repository.

Setting up a new Project

- git init

#Initializes local repository to host github repository

- git remote add origin <repository url>

# Connects the local repository to remote repository

- git checkout -b <branch_name>

# specify which branch of repository to pull from

- git pull origin <branch_name>

# Pull latest code from remote repository to local repository

Day to Day Process

```
git pull origin <branch_name>
```

# Pull latest code from remote repository to local repository

```
git add <filename> (use ‘ . ‘ to add all the files containing changes)
```

# Adds the file to your local repository and stages it for commit. To unstage a file, use 'git reset HEAD YOUR-FILE'.

Commit the file that you've staged in your local repository.

```
git commit -m "testing new file"
```

# Commits the tracked changes and prepares them to be pushed to a remote repository. To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and add the file again.

Push the changes in your local repository to GitHub.

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

# Pushes the changes in your local repository up to the remote rep

Collaborative Git Practices

Code Reviews:

- Always review code changes before merging to ensure quality and consistency.
- **Example:** Use pull requests for team members to review code before it is merged.

Pull Requests:

- A pull request is a way to propose changes to a codebase. It allows for discussion and review before merging.
- **Example:** Creating a pull request on GitHub or Bitbucket for a new feature branch.

Resolving Conflicts:

- Merge conflicts occur when changes in two branches affect the same lines in a file. Learn how to resolve them.
- **Example:** Use `git merge` to merge branches and manually resolve conflicts if they arise.

Best Practices:

- **Commit Often:** Make small, frequent commits with meaningful messages.
- **Use Branches:** Create branches for new features, bug fixes, or experiments.
- **Review and Test:** Always review and test changes before merging them into the main branch.

Conclusion

Understanding and effectively using version control is crucial for any IT professional or developer. Mastery of Git and its workflows not only improves collaboration but also enhances the quality and maintainability of code.

This training will provide the foundation needed to implement version control best practices in any project.