N M VIJAY
LinkedIn

DAY 1

Detailed Notes on Git and Version Control Systems

Introduction to Git

- **Git** is a version control tool used to track changes in files and directories.
- It is essential for **Source Code Management (SCM)**, helping developers efficiently manage different versions of their code.
- Git allows multiple developers to collaborate on a project without overwriting each other's work.

Types of Version Control Systems (VCS)

1. Centralized Version Control System (CVCS)

- **Definition**: A single, central repository stores all versions of the code.
- **Workflow**:
    - Developers commit their changes directly to the central repository.
    - Other developers update their working copies by pulling changes from the central repository.
- **Example**: SVN (Subversion).
- **Disadvantages**:
    - If the central repository fails, all data is lost.
    - No local backup of the code.

2. Distributed Version Control System (DVCS)

- **Definition**: Each developer has a local repository, and changes are pushed to a central repository.
- **Workflow**:
    - Developers commit changes to their local repository.
    - Changes are then pushed to the central repository.

N M VIJAY
LinkedIn

    - Other developers pull changes from the central repository to their local repositories.

- **Example**: Git.
- **Advantages**:
    - If the central repository fails, the code can be recovered from local repositories.
    - Provides better backup, redundancy, and offline access.

Git Workflow

Git operates in different stages:

1. **Working Directory**: The local directory where developers write and edit code.
2. **Staging Area**: An intermediate area where changes are reviewed before committing.
3. **Local Repository**: Stores committed changes before pushing them to a central repository.

Key Git Commands

- **git** init → Initializes a new Git repository.
- **git** add <file> → Moves changes from the working directory to the staging area.
- **git** commit -m "message" → Moves changes from the staging area to the local repository.
- **git** push origin <branch> → Pushes changes from the local repository to the central repository (e.g., GitHub).
- **git** pull origin <branch> → Updates the working copy with changes from the central repository.

Practical Example

Scenario: Two developers (one from India and another from the US) are working on the same project.

1. The **Indian developer** commits changes to their local repository and pushes them to the central repository.

N M VIJAY
LinkedIn

2. The **US developer** pulls the latest changes from the central repository into their local repository.
3. Both developers continue working independently and merge their changes seamlessly.

Interview Questions and Answers

1. What is Git, and why is it used?

- **Answer**: Git is a **distributed version control system** used to track changes in files and directories. It helps developers manage code versions, collaborate efficiently, and maintain a history of all changes.

2. What is the difference between CVCS and DVCS?

- **Answer**:
    - **CVCS**: A single central repository where all changes are committed. If it fails, all data is lost.
    - **DVCS**: Each developer has a local repository. Even if the central repository fails, local copies act as backups, ensuring better redundancy.

3. Explain the Git workflow.

- **Answer**: Git has three main stages:
    1. **Working Directory** → Where developers write and edit code.
    2. **Staging Area** → A temporary area to review changes before committing.
    3. **Local Repository** → Stores committed changes, which can be pushed to a central repository.

4. What is the purpose of the staging area in Git?

- **Answer**: The staging area allows developers to review changes before committing them. It acts as an intermediate space where files can be modified or removed before finalizing a commit.

5. How do you resolve conflicts in Git?

- **Answer**:
    - Conflicts occur when two developers modify the same part of a file.
    - Git marks the conflicting areas. Developers must **manually edit** the file to resolve the conflict.

N M VIJAY
LinkedIn

- After resolving, the changes are **staged (git add)** and **committed (git commit)**.

6. What is a commit ID in Git?

- **Answer**: A **commit ID** is a unique **40-character alphanumeric identifier** assigned to each commit. It represents a specific version of the code in the repository.

7. What is the difference between git pull and git fetch?

- **Answer**:
  - git pull: **Fetches** changes from the remote repository and **merges** them into the working directory.
  - git fetch: Only **downloads** the changes but does not merge them. Developers must manually merge them.

8. How do you revert a commit in Git?

- **Answer**: Use the command git revert <commit-id>. This creates a **new commit** that undoes the changes from the specified commit, without deleting history.

9. What is branching in Git, and why is it important?

- **Answer**:
  - **Branching** allows developers to create **independent versions** of the code for different features or bug fixes.
  - Once the changes are tested, the branch can be **merged** into the main codebase.

10. What is the difference between git merge and git rebase?

- **Answer**:
  - git merge: **Combines** changes from two branches, keeping the history of both branches.
  - git rebase: **Moves** a branch to a new base commit, creating a **linear history**. Used to clean up the commit history before merging.

Conclusion

- Git is a powerful tool for **version control**, enabling **efficient collaboration** and **code management**.

N M VIJAY
LinkedIn

- Understanding the **differences between CVCS and DVCS**, as well as **mastering the Git workflow**, is crucial for **DevOps** and **software development**.
- **Practice Git commands** regularly to become proficient in version control and prepare for technical interviews.

Key Takeaways

✓ **Git** is a distributed version control system used for **tracking file changes**.
✓ **CVCS** vs. **DVCS** → DVCS provides **better backup and redundancy**.
✓ **Git Workflow** → **Working Directory** → **Staging Area** → **Local Repository** → **Remote Repository**.
✓ **Key Git Commands**: init, add, commit, push, pull, merge, rebase, revert, fetch.
✓ **Branching** allows developers to work on separate features **without affecting the main branch**.

Suggestions for Further Learning

📌 Practice Git commands using GitHub or GitLab.
📌 Explore **Git branching** and **merging strategies**.
📌 Learn about **Git workflows** like **Feature Branch Workflow**, **Gitflow**, and **Forking Workflow**.

This version ensures **clarity, correctness, and easy note-taking**, making it perfect for learning and reference. 🚀 Let me know if you need further improvements! 😊

N M VIJAY
LinkedIn

Day 2

Detailed Notes on Git and DevOps Workflow

Introduction to Git and DevOps Tools

- **Git**: A **distributed version control system (DVCS)** used for tracking changes in source code during software development.
- **DevOps Tools**: A collection of tools used for automation and continuous integration/continuous deployment (CI/CD).
    - Common tools: **Git, Jenkins, Docker, Chef, Puppet, Kubernetes**.
    - Among these, **Git and Jenkins** are widely used across most companies.

Centralized vs. Distributed Version Control Systems (CVCS vs. DVCS)

Centralized Version Control System (CVCS)

- Stores code in a **single central repository**.
- Developers **commit and update** code directly to the central repository.
- **Disadvantages**:
    - If the central repository fails, **all data is lost**.
    - Higher risk of failure, as there is no local backup.

Distributed Version Control System (DVCS)

- Developers **commit code to a local repository first**.
- The local repository then **pushes** changes to a central repository.
- **Advantages**:
    - If the central repository fails, the code **remains safe** in local repositories.
    - Provides **better backup, redundancy, and offline access**.

N M VIJAY
LinkedIn

Git Workflow

1. Working Directory (Working Copy)

- This is where developers **write, edit, and modify** code.
- The modified files remain in the **working directory** until they are staged.

2. Staging Area

- A **temporary storage area** where changes are reviewed before committing them.
- Files in the **staging area** are **ready for commit** but not yet saved in the repository.

3. Local Repository

- Once a file is **committed**, it is stored in the local repository.
- Each commit generates a **unique commit ID (SHA-1 hash)**.
- After committing, files cannot be removed from the repository but can be reverted.

Git Commands and Their Usage

Basic Git Commands

Command	Description
git init	Initializes a new Git repository.
git status	Shows the status of files in the working directory, staging area, or local repository.
git add <file>	Moves files from the working directory to the staging area.
git commit -m "message"	Commits files from the staging area to the local repository.
git log	Lists all commits and their commit IDs.
git show <commit-id>	Displays the details of a specific commit.

N M VIJAY
LinkedIn

Working with Remote Repositories

Command	Description
git remote add origin <URL>	Links the local repository to a remote repository (e.g., GitHub).
git push origin master	Pushes local commits to the remote repository.
git pull origin master	Pulls the latest changes from the remote repository to the local machine.

Git Status Color Codes

Color	Meaning
Red	File is in the working directory but not staged.
Green	File is in the staging area, ready for commit.
No changes	Code is committed and stored in the local repository.

GitHub and Personal Access Tokens

What is GitHub?

- **GitHub** is a cloud-based repository hosting service where developers store and manage code collaboratively.

What is a Personal Access Token?

- A **Personal Access Token (PAT)** is an authentication method used instead of a password to push code to GitHub.

Steps to Generate a Personal Access Token:

1. Log in to GitHub and navigate to **Settings > Developer Settings**.
2. Click on **Personal Access Tokens** and select **Generate new token**.
3. Select necessary permissions and generate the token.
4. Copy the token and **use it as a password** when pushing code to GitHub.

N M VIJAY
LinkedIn

Practical Steps to Work with Git and GitHub

1. Launch an EC2 Instance and Connect

- **Launch an EC2 instance** (e.g., Ubuntu 22.04) and connect via SSH.

2. Install Git

- **For Ubuntu/Debian**:
- sudo apt install git
- **For Amazon Linux/CentOS**:
- sudo yum install git

3. Create a Directory and Initialize Git

mkdir my_project && cd my_project

git init

4. Configure Git User Details

git config --global user.name "your_username"

git config --global user.email "your_email"

5. Create a File and Add Content

echo "Hello, World!" > myfile.txt

6. Check the Git Status

git status

7. Add the File to the Staging Area

git add myfile.txt

8. Commit the File to the Local Repository

git commit -m "Initial commit"

9. Push Code to GitHub

git remote add origin <GitHub-URL>

git push origin master

- Use your GitHub **username** and **personal access token** for authentication.

10. Pull Code from GitHub

N M VIJAY
LinkedIn

git pull origin master

Interview Questions and Answers

1. What is the difference between CVCS and DVCS?

- **CVCS** stores code in a **central repository**, and developers commit directly to it. If the central repo fails, all data is lost.
- **DVCS** allows developers to commit to a **local repository first**, ensuring redundancy even if the central repo fails.

2. Explain the Git workflow.

- **Working Directory** → Where developers write and edit code.
- **Staging Area** → Where files are prepared for commit.
- **Local Repository** → Where committed files are stored.

3. What is a commit ID, and why is it important?

- A **commit ID** is a **unique identifier** for each commit in Git.
- It allows developers to track changes, revert commits, and manage different versions of the code.

4. How do you push code to GitHub?

- First, connect Git to GitHub:
- git remote add origin <GitHub-URL>
- Then, push changes using:
- git push origin master

5. What is a personal access token, and why is it needed?

- A **Personal Access Token (PAT)** is used instead of a password for authentication when pushing code to GitHub.

6. How do you check the commit history in Git?

- Use git log to **list all commits**.
- Use git show <commit-id> to **view details** of a specific commit.

7. How do you revert a commit in Git?

- Use:

N M VIJAY
LinkedIn

- git revert <commit-id>
- This creates a **new commit** that undoes the changes from the specified commit.

8. How do you resolve a Git conflict?

- Git marks conflicting changes. Developers must **manually edit** the file, stage it (git add), and commit the resolved version.

Conclusion

- Git is an essential tool for **version control**, enabling **efficient collaboration** and **code management**.
- Understanding **Git commands, GitHub authentication, and Git workflows** is crucial for **DevOps and software development**.
- Regular **practice with Git commands** will improve proficiency and help in **technical interviews**.

N M VIJAY
LinkedIn

Day 3

Detailed Notes on Git Concepts

---

1. Branching in Git

🔹 **Definition**: Branching enables **parallel development**, allowing multiple teams or individuals to work on different parts of a project **without affecting the main codebase**. These branches can later be merged into the main branch (usually master or main).

🔹 **Purpose**:
✅ Enables multiple developers to **work simultaneously** on different features.
✅ **Isolates** new features, bug fixes, or experimental changes from the main codebase.

🔹 **Example**:

- **Master Branch** → Contains the **main** version of the code (e.g., Login Page).
- **Feature Branch (Branch 1)** → A new feature is developed (e.g., Logout Page).
- After development, **Branch 1 is merged into the Master Branch**.

🔹 **Common Git Commands for Branching**:

Command	Description
git branch	Lists all branches.
git branch <branch_name>	Creates a new branch.
git checkout <branch_name>	Switches to a different branch.
git checkout -b <branch_name>	Creates and switches to a new branch in a single step.
git merge <branch_name>	Merges a specified branch into the current branch.

N M VIJAY
LinkedIn

2. Git Workflow

🔹 Git follows a **three-stage process** to manage changes effectively:

1. **Working Directory (Working Copy)** → Where you **edit** files.
2. **Staging Area** → An **intermediate area** where changes are reviewed before committing.
3. **Local Repository** → Where committed changes are stored.

🔹 **Git Workflow Process**:

✓ Edit files in the **Working Directory**.
✓ Move changes to the **Staging Area** using git add.
✓ Commit changes to the **Local Repository** using git commit -m "message".
✓ Push changes to a **Remote Repository** (GitHub) using git push.

---

3. Merge Conflict in Git

🔹 **Definition**: A **merge conflict** occurs when **two branches modify the same file differently**, and Git cannot automatically merge them.

🔹 **Scenario**:

- **Master Branch**: file007 contains "Hello Guys".
- **Branch 2**: file007 contains "How Are You".
- When merging, Git **cannot decide** which content to keep, resulting in a **conflict**.

🔹 **Steps to Resolve a Merge Conflict**:

1. Open the conflicting file in an editor (e.g., vi file007).
2. Manually edit the file to resolve the conflict (**choose one version or combine both**).
3. Save the file and mark it as resolved using git add file007.
4. Commit the resolved changes:

git commit -m "Merge conflict resolved"

---

4. Stashing in Git

🔹 **Definition**: **Stashing** is used to **temporarily save uncommitted changes** when you need to switch branches or work on something else **without committing unfinished work**.

🔹 **Scenario**:

N M VIJAY
LinkedIn

- You are working on **Feature A (Flipkart code)** but suddenly need to switch to **Feature B (Amazon code)**.
- Instead of committing unfinished work, **stash it** and retrieve it later.

🔹 **Common Git Stash Commands**:

Command	Description
git stash	Stashes (saves) the current changes.
git stash list	Lists all stashed changes.
git stash apply	Applies the most recent stash.
git stash clear	Clears all stashed changes.

---

5. Git Log and Commit IDs

🔹 **Git Log**: Shows the commit history with unique **commit IDs**.
🔹 **Commit ID**: A **unique identifier** assigned to each commit, helping track changes.

🔹 **Useful Commands**:

Command	Description
git log	Displays the commit history.
git checkout <commit_id>	Moves to a specific commit version.

N M VIJAY
LinkedIn

💡 Interview Questions and Answers

1. What is branching in Git, and why is it important?

✅ **Answer**:
Branching allows developers to work on **different features or fixes in parallel** without affecting the main codebase. It **enables team collaboration**, isolates new work, and simplifies merging changes.

---

2. How do you resolve a merge conflict in Git?

✅ **Answer**:

1. Open the **conflicting file** in an editor.
2. **Manually edit** the file to resolve the conflict.
3. Save the file and mark it as resolved using git add.
4. **Commit** the changes using:
5. git commit -m "Conflict resolved"

---

3. What is the purpose of the staging area in Git?

✅ **Answer**:
The **staging area** acts as a **buffer** between the working directory and the repository. It allows developers to **review and selectively commit** changes, ensuring better version control.

---

4. What is stashing in Git, and when would you use it?

✅ **Answer**:
Stashing temporarily saves uncommitted changes so that you can switch tasks **without committing unfinished work**. Use git stash to save changes and git stash apply to restore them later.

N M VIJAY
LinkedIn

---

5. How do you create and switch between branches in Git?

✅ **Answer**:

- **Create a new branch**:
- git branch <branch_name>
- **Switch to a branch**:
- git checkout <branch_name>
- **Create and switch in one command**:
- git checkout -b <branch_name>

---

6. What is a commit ID, and how is it used?

✅ **Answer**:
A **commit ID** is a unique **40-character identifier** for each commit. It allows developers to track changes, revert commits, and restore previous versions.

---

7. What is the difference between git merge and git rebase?

✅ **Answer**:

Command	Description
git merge	Combines two branches, preserving their history.
git rebase	Moves commits to a new base commit, creating a **linear** history.

---

8. How do you delete a branch in Git?

✅ **Answer**:

- **Delete a branch**:
- git branch -d <branch_name>
- **Force delete an unmerged branch**:
- git branch -D <branch_name>

N M VIJAY
LinkedIn

---

9. What is the purpose of git log, and how do you use it?

✅ **Answer**:
git log shows **commit history, commit IDs, authors, and dates**. It helps track changes and review project history.

🔟 **What is the difference between git stash and git commit?**

✅ **Answer**:

Command	Purpose
git stash	Temporarily **saves** changes without committing.
git commit	Permanently **records** changes in the repository.

📝 **Summary**

✅ **Branching**: Allows developers to work on features **in parallel**.
✅ **Merge Conflict**: Occurs when different branches modify the same file **differently**.
✅ **Stashing**: Temporarily saves changes to **switch tasks without committing**.
✅ **Git Log**: Helps track changes using **commit IDs**.
✅ **Essential Commands**: git branch, git checkout, git merge, git stash, git log.

🎯 **Practice Tips**

✅ Create **feature branches**, merge them, and resolve conflicts.
✅ Use git stash to manage **unfinished work**.
✅ Run git log to review **commit history**.

By mastering these **concepts and commands**, you will be well-prepared for **Git workflow management and technical interviews**. 🚀

N M VIJAY
LinkedIn

Day 4
Git Concepts and Commands

1. Git Reset

- **Definition:**
  - Git Reset is a command used to undo changes in the local repository by moving the **HEAD** pointer to a specific commit.
  - It can remove changes from the **staging area** or **working directory**.
- **Usage Scenarios:**
  - Undo staged changes (git add but not committed yet).
  - Discard changes in the working directory.
- **Types of Git Reset:**
  - **Soft Reset (git reset --soft)**: Moves **HEAD** to a commit but keeps changes in the **staging area**.
  - **Mixed Reset (git reset --mixed)**: Moves **HEAD** and removes changes from staging, but keeps them in the **working directory**.
  - **Hard Reset (git reset --hard)**: Moves **HEAD** and discards all changes in both staging and working directories.
- **Example Commands:**
- git reset file007 # Removes from staging area, keeps in working directory
- git reset --hard # Discards all changes completely

---

2. Git Revert

- **Definition:**
  - Git Revert is used to **undo a commit** by creating a new commit that reverses the previous commit's changes.
  - It does not delete commits, keeping commit history intact.
- **Usage Scenarios:**
  - Undo a commit after pushing to a shared repository.
  - Maintain a clean commit history.
- **Example Command:**

N M VIJAY
LinkedIn

- git revert abc123 # Reverts commit with ID abc123

---

3. Git Clone

- **Definition:**
    - Creates a local copy of a remote repository, downloading all files, branches, and commit history.
- **Usage Scenarios:**
    - Start working on a remote project.
    - Contribute to an open-source project.
- **Example Command:**
- git clone https://github.com/username/repository.git

---

4. Git Fork

- **Definition:**
    - Forking creates a personal copy of someone else's repository on platforms like GitHub.
    - Allows independent changes without affecting the original repository.
- **Usage Scenarios:**
    - Contributing to open-source projects.
    - Experimenting with a project without modifying the original.
- **Steps to Fork and Clone:**

1. Click **Fork** on GitHub.
2. Clone the forked repository:
3. git clone https://github.com/your-username/Python123.git

---

5. Git Merge vs. Git Rebase

- **Git Merge:**
    - Combines changes from one branch into another.
    - Preserves commit history.

N M VIJAY
LinkedIn

- **Git Rebase:**
  - Moves a branch to the latest commit of another branch, rewriting history.
  - Creates a linear commit history.
- **Usage Guidelines:**
  - Use **Merge** to preserve history.
  - Use **Rebase** for a clean, linear history.

---

6. Git Cherry-Pick

- **Definition:**
  - Applies a specific commit from one branch to another.
- **Example Command:**
- git checkout main
- git cherry-pick xyz456

---

7. Git Pull vs. Git Fetch

- **Git Fetch:**
  - Retrieves changes from the remote repository but does not apply them.
- **Git Pull:**
  - Retrieves and applies remote changes to the local branch.
- **Example Commands:**
- git fetch origin main
- git pull origin main

N M VIJAY
LinkedIn

Extras
Comprehensive Notes on Git & GitHub

1. Git Reset

- **Definition:** Undo changes in the local repository by moving the HEAD pointer to a specific commit.
- **Use Cases:**
  - Remove changes from staging area or working directory.
  - Discard unnecessary changes.
- **Types:**
  - **Soft Reset (git reset --soft)**: Moves HEAD but keeps changes staged.
  - **Mixed Reset (git reset --mixed)**: Moves HEAD and unstages changes.
  - **Hard Reset (git reset --hard)**: Moves HEAD and discards all changes.
- **Example:**
- git reset file007 # Removes from staging, keeps in working directory
- git reset --hard # Discards all changes

---

2. Git Revert

- **Definition:** Creates a new commit that reverses changes from a previous commit without deleting history.
- **Use Case:** Undo a commit safely in a shared repository.
- **Example:**
- git revert abc123 # Creates a new commit that undoes abc123

---

3. Git Clone

- **Definition:** Creates a local copy of a remote repository.
- **Example:**
- git clone https://github.com/user/repository.git

---

4. Git Fork

N M VIJAY
LinkedIn

- **Definition:** Creates a personal copy of a repository on GitHub.
- **Steps to Fork & Clone:**
    1. Click **Fork** on GitHub.
    2. Clone it to local machine:
    3. git clone https://github.com/your-username/repository.git

---

5. Git Merge vs. Git Rebase

- **Merge:** Combines branches while preserving history.
- **Rebase:** Moves a branch onto another, rewriting commit history.
- **Use Merge when** preserving history is important.
- **Use Rebase when** a cleaner, linear history is needed.

---

6. Git Cherry-Pick

- **Definition:** Apply specific commits from one branch to another.
- **Example:**
- git checkout main
- git cherry-pick xyz456

---

7. Git Pull vs. Git Fetch

- **Git Fetch:** Retrieves latest changes but does not merge.
- **Git Pull:** Fetches and merges changes from remote.
- **Example:**
- git fetch origin
- git pull origin main

---

8. Git Stash

- **Definition:** Temporarily saves uncommitted changes.
- **Commands:**

N M VIJAY
LinkedIn

- git stash # Save changes
- git stash pop # Apply and remove stash
- git stash list # Show stashed changes

---

9. Git Branching

- **Create a new branch:**
- git branch feature-branch
- **Switch branches:**
- git checkout feature-branch
- **Delete branch:**
- git branch -d feature-branch

---

10. Git Tagging

- **Definition:** Mark a specific commit as a release version.
- **Example:**
- git tag v1.0.0
- git push origin v1.0.0

---

11. GitHub Pull Requests (PR)

- **Steps to Create a PR:**
    1. Push changes to a new branch.
    2. Go to GitHub and select **New Pull Request**.
    3. Add a description and submit for review.

---

12. GitHub Issues

- **Definition:** Track bugs and feature requests.
- **How to Create an Issue:**
    1. Go to **Issues** tab on GitHub.

N M VIJAY
LinkedIn

2. Click **New Issue** and describe the problem.

---

13. GitHub Actions (CI/CD)

- **Definition:** Automate workflows for testing and deployment.
- **Example Workflow:**
- name: CI/CD Pipeline
- on: [push]
- jobs:
- build:
- runs-on: ubuntu-latest
- steps:
- - uses: actions/checkout@v2
- - name: Run Tests
- run: npm test

---

14. GitHub Webhooks

- **Definition:** Trigger events (e.g., deploy when code is pushed).
- **Steps to Add Webhook:**
  1. Go to repository **Settings** → **Webhooks**.
  2. Click **Add Webhook** and enter the URL.

---

15. Git Ignore (.gitignore)

- **Definition:** Prevent specific files from being tracked.
- **Example .gitignore file:**
- node_modules/
- .env
- *.log

GIT/GITHUB NOTES

N M VIJAY
LinkedIn

This document provides a complete reference for **Git & GitHub**, ensuring you have everything needed for practical use. 🚀