Introduction to Java

Week 2 Day 1- Day 5

Git

- Version control systems track file changes over time and aid collaboration among developers. They manage code modifications through specialized databases. Services like GitHub, Bitbucket, and GitLab offer remote repositories for storing and sharing Git projects, with options ranging from free to commercial.

What is Git?

- Git is an open-source distributed version control system ideal for projects of any size. I
- t's built for speed and efficiency, enabling seamless collaboration among developers within the same workspace.
- While it forms the foundation for services like GitHub and GitLab, Git can be used independently, both privately and publicly.
- Initially crafted by Linus Torvalds in 2005 for the Linux Kernel, Git boasts easy learning curves and rapid performance.
- Its capabilities surpass those of other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Benefits of Git.

- Git offers numerous advantages:
- 1. Time Efficiency: Git operates swiftly, executing commands within seconds, saving considerable time compared to navigating through online interfaces.
- 2. Offline Capability: It supports offline work, enabling local operations without reliance on constant internet connectivity, unlike some other CVS like SVN that depend on central repositories.
- 3. Error Reversal: Git provides robust undo options, allowing for quick rectification of mistakes, serving as a safety net during development.
- 4. Change Tracking: With features like Diff, Log, and Status, Git enables easy tracking of modifications, facilitating the review of file or branch status.

Git tools.

- Git comes with various tools to enhance its functionality and user interface.
- Git Bash: This tool, available for Windows, emulates a Git command-line experience, offering a robust shell interface with essential commands like Ssh, scp, cat, find, and the entire set of Git core commands.
- Git GUI: It's a graphical version of Git's command line, providing visual diff tools for easier navigation and interaction with Git functionalities. It can be accessed via right-clicking on a folder or through a command line command like `$ git gui`.

- Gitk:This graphical history viewer acts as a shell over git log and git grep, aiding in visualizing project history and navigating past changes. Invoking it is as simple as typing `$ gitk [git log options]` in the command line within a Git repository.
- These tools within Git offer different interfaces to cater to diverse user preferences, making it more versatile and accessible for developers across various platforms and skill levels.

Git Terminology.

Certainly! Git terminology can be quite extensive, so here's an overview:

- **Branch:** A distinct version of a repository that diverges from the main project.

- **Checkout:** Switching between different versions of a target entity in Git.

- **Cherry-Picking:** Selectively applying commits from one branch to another.

- **Clone:** Creating a local copy of a repository from a remote URL.

- **Fetch:** Updating remote-tracking branches with changes from other repositories.

- **HEAD:** Represents the latest commit in the current branch.

- **Index:** A staging area between the working directory and repository for preparing commits.

- **Master:** The default and often primary branch in Git.

- **Merge:** Integrating forked histories into a single branch.

- Origin: A reference to the original repository from which a project was cloned.

- Pull/Pull Request:Fetching and merging changes from a remote server, and a formal request to merge changes into a branch.

- Push: Uploading local changes to a remote repository.

- Rebase: Rearranging commits onto a new base commit.

- Remote:The shared repository where team members exchange changes.

- Repository: The structure storing file data and change history in Git.

- Stashing:Temporarily saving incomplete work without committing.

- Tag: Marking specific points in Git history as important references.

- Upstream/Downstream: References to the original and integrated repositories.

- Revert: Undoing specific commits.

- Reset:Undoing changes with different modes (soft, mixed, hard).

- Ignore: Specifying untracked files for Git to disregard.

- Diff:Comparing differences between Git data sources.

- Cheat Sheet: A quick reference summary of Git commands.

- Git Flow: A branching model for collaborative development.

- Squash: Combining multiple commits into one.

- Rm: Removing tracked files from the Git index.

- Fork: A duplicate copy of a repository, often used for testing and proposing changes.

Day 3

Basic commands of Git

Git Config

```bash
$ git config --global user.name "Your Name"
$ git config --global user.email "youremail@example.com"
```

Git Init

```bash
$ git init
```

Git Clone

```bash
$ git clone repository_url
```

Git Add

```bash
$ git add filename
$ git add *
```

```

### Git Commit

```bash
$ git commit -m "Commit message"
$ git commit -a
```

Git Status

```bash
$ git status
```

Git Push

```bash
$ git push origin master
$ git push --all
```

Git Pull

```bash
$ git pull origin master
```

Git Branch

```bash
$ git branch
```
```

Git Merge

```bash
$ git merge branch_name
```

Git Log

```bash
$ git log
$ git log -3
```

Staging and commit

Absolutely! Let's break down the major staging and committing operations in Git:

Git Add

- **Purpose**: Adds file contents to the staging area, preparing them for the next commit.
- **Usage**:
  - To add a single file: `$ git add filename`
  - To add all files: `$ git add -A` or `$ git add .`
  - To add only updated and newly created files: `$ git add --ignore-removal`
  - To add only modified and deleted files: `$ git add -u`
  - To add files using wildcard: `$ git add *.java`

Git Commit

- **Purpose**: Records changes in the repository by creating a commit.
- **Usage**:
  - Interactive commit message: `$ git commit`
  - Committing all previously added files: `$ git commit -a`
  - Adding a commit message: `$ git commit -m "Commit message"`

- Amending the last commit's message: `$ git commit --amend`

Git Clone

- **Purpose**: Creates a local copy of a remote repository.

- **Usage**: `$ git clone <repository URL>`

Git Fork

- **Purpose**: Creates a copy of a repository, often used for proposing changes or experimenting without affecting the original project.

- **Usage**: Done through the Git service platform (e.g., GitHub). Forking is not a Git command; it's a feature provided by Git service providers like GitHub.

Day 4

Inspect and Undo changes

Git log is a vital tool in version control systems like Git, offering a comprehensive history of a repository's changes. It includes commit hashes, author details, commit dates, and commit messages. The following options enhance its functionality:

Basic Git Log

```bash
$ git log
```

Displays recent commits with their unique identifiers, dates, authors, and commit details.

Git Log Stat

```bash
$ git log --stat
```

```

Shows modified files, line additions/deletions, and a summary of changes.

### Git Log Patch

```bash
$ git log -p
```

Reveals modified files and the specific line changes made within those files.

Git Checkout

Git checkout switches between different versions of a repository, be it files, commits, or branches. Careful when switching branches as it alters the working directory files to match the chosen branch.

Operations with Git Checkout

- **Checkout Branch**

```bash
$ git checkout <branchname>
```

Switches between branches.

- **Create and Switch Branch**

```bash
$ git checkout -b <branchname>
```

Creates and switches to a new branch simultaneously.

- **Checkout Remote Branch**

```bash
$ git checkout <remotebranch>
```
```

```

Allows accessing and working on a remote branch after fetching its contents.

These commands aid in navigating and inspecting a repository's history, tracking changes, and managing branches efficiently in Git.

**Collaborating**

Sure, here's a simplified summary:

### Git Fetch

- **Purpose:** Downloads commits, objects, and refs from another repository.

- **Usage:**

  - `$ git fetch <repository URL>`: Fetches the complete repository.

  - `$ git fetch <branch URL> <branch name>`: Fetches from a specific branch.

  - `$ git fetch --all`: Fetches all branches simultaneously.

### Git Pull / Pull Request

- **Git Pull:** Receives data from a remote repository to update the local repository.

  - Syntax: `$ git pull <option> [<repository URL> <refspec>...]`

- **Pull Request:** Notifies team members of completed work for review and merging.

### Git Push

- **Purpose:** Uploads local repository content to a remote repository.

- **Usage:**

  - `$ git push <option> [<Remote URL> <branch name> <refspec>...]`

  - `$ git push origin master`: Pushes local content to the master branch of the remote repository.

- **Options:** Includes various options like `--all`, `--prune`, `--mirror`, `--dry-run`, `--tags`, `--delete`, `-u`.

### Git Force Push
```

- **Purpose:** Pushes local repository changes to the remote repository without conflict handling.
- **Usage:**
  - `$ git push <remote> <branch> -f` or `$ git push <remote> <branch> --force`
  - `$ git push <remote> <branch> --force-with-lease` for a safe force push.

Delete a Remote Branch

- **Purpose:** Removes a remote branch from the command line.
- **Usage:** `$ git push origin --delete <branch name>`