|  |  |  |
| --- | --- | --- |
| SCHOOL OF INFORMATION AND TECHNOLOGY | | |
| NAME: Charles Hanzel Gerardo | DATE PERFORMED: 11/21/2024 |  |
| Section: IDC2 | DATE SUBMITTED: 11/21/2024 |

# SYSADM1 – Git Basics

Answer the following research questions about Git, GitLab desktop and GitHub.

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

Git is a distributed version control system (VCS) used to track changes in source code during software development. It allows developers to collaborate on projects, maintain a history of changes, and manage different versions of a project without overwriting or losing any work. Git is important because it enables collaborative development, provides version history for tracking and managing changes, and helps to maintain code integrity across teams and over time.

1. How does Git track changes in a project?

Git tracks changes using a system of snapshots and commits:

* Snapshots: Git records a snapshot of the entire project directory (or parts of it) every time you commit, saving the state of all files.
* Commits: Each snapshot is linked to a commit, which is like a historical record that includes metadata such as the author, timestamp, and a message describing what was changed. The snapshots are stored in a `.git` directory within the project.
* Hashes: Git assigns a unique hash (SHA-1) to each commit, ensuring that changes are tracked and identified distinctly.

1. What is the difference between a local repository and a remote repository in Git?

* Local Repository: This is the version of the project that is stored on your local machine. It includes your working directory (the actual project files), staging area (index), and a `.git` directory that contains the Git history and configuration.
* Remote Repository: This is a repository hosted on a server or cloud platform (like GitHub or GitLab). It serves as a centralized place where multiple contributors can push their changes and pull others' changes. Examples include repositories on GitHub or GitLab.

1. What are the basic Git commands?

Here are some of the most commonly used Git commands:

* `git init`: Initializes a new Git repository.
* `git clone <repo\_url>`: Clones a remote repository to your local machine.
* `git add <file>`: Adds a file to the staging area (preparing it for commit).
* `git commit -m "message"`: Commits changes in the staging area to the local repository with a message describing the changes.
* `git status`: Displays the status of the working directory and staging area.
* `git push`: Pushes commits from the local repository to a remote repository.
* `git pull`: Fetches and merges changes from a remote repository to the local repository.
* `git branch`: Lists, creates, or deletes branches.
* `git checkout <branch>`: Switches to another branch.
* `git merge <branch>`: Merges the changes from one branch into the current branch.

1. How do you check the status of a Git repository?

To check the status of your Git repository, use the command ‘git status’.

This command shows:

* Which files have been modified but not yet staged.
* Which files are staged and ready for commit.
* Whether there are any untracked files in the working directory.
* Information about the current branch.

1. What is the purpose of branches in Git, and how do you create and switch between them?

Purpose of Branches: Branches allow developers to work on separate features or bug fixes without affecting the main codebase (typically the `main` or `master` branch). It enables parallel development, experimentation, and isolation of changes.

* Create a Branchn = git branch <branch\_name>
* Switch to a Branch = git checkout <branch\_name>
* Create and switch to a branch in one step using = git checkout -b <branch\_name>

1. What are GitLab Desktop and GitHub, and how are they different from Git?

* GitLab Desktop: GitLab Desktop is a graphical user interface (GUI) application that helps users interact with Git repositories hosted on GitLab. It simplifies tasks such as committing, branching, and pushing changes to GitLab repositories, but it still relies on Git as the underlying version control system.
* GitHub: GitHub is a platform for hosting Git repositories and collaborating with other developers. It provides web-based interfaces to manage repositories, track issues, pull requests, and collaborate on code.
* Difference from Git: Both GitLab and GitHub are web-based platforms that integrate with Git, providing remote repository hosting, issue tracking, and collaboration tools. Git is the version control system that tracks changes locally, while GitHub and GitLab provide additional features such as web interfaces and project management tools for Git repositories.

1. How do you connect a local Git repository to a GitLab or GitHub repository?

To connect a local Git repository to a remote GitLab or GitHub repository:

1. Create a new repository on GitLab or GitHub.

2. In your local Git repository, run the ‘git remote add origin <repo\_url>’ to add the remote repository. Replace `<repo\_url>` with the URL of your GitLab or GitHub repository.

3. To push your local repository to the remote run ‘git push -u origin <branch\_name>’

1. What are the steps to collaborate with others using GitLab or GitHub?

1. Fork or clone a repository: If you want to contribute to an existing project, fork the repository (on GitHub/GitLab) or clone it to your local machine.

2. Create a branch: Always create a new branch for your changes to avoid working directly on the main branch, git checkout -b <feature\_branch>.

3. Make changes: Edit the code and add your changes.

4. Commit and push changes: After editing, commit your changes and push them to the remote repository.

git add .

git commit -m "Description of changes"

git push origin <feature\_branch>

5. Open a pull request (GitHub) / Merge Request (GitLab): After pushing your changes, open a pull request (GitHub) or merge request (GitLab) to request the integration of your changes into the main codebase.

6. Review and merge: The project maintainers will review the changes and merge them if everything looks good.

1. How do you resolve merge conflicts in Git?

Merge conflicts occur when two branches make changes to the same line of a file or the same file. To resolve them:

1. Run `git merge <branch\_name>` to attempt to merge the changes.

2. Git will notify you of the conflict and mark the conflicted files.

3. Open the conflicted files, and look for the conflict markers (`<<<<<<`, `======`, `>>>>>>`).

4. Manually resolve the conflicts by editing the file to keep the correct changes.

5. After resolving, add the files to the staging area:

git add <file\_name>

6. Commit the resolved changes:

git commit -m "Resolved merge conflict in <file\_name>"

1. What is a pull request, and why is it used in GitHub?

A pull request (PR) is a request to merge changes from one branch into another, typically from a feature branch into the main branch. It is used in GitHub to:

* Review changes: Team members or project maintainers can review the proposed changes before they are merged into the main codebase.
* Discuss changes: The pull request provides a space for discussions, feedback, and suggestions.
* Track changes: Pull requests make it easy to track which changes have been proposed, reviewed, and merged.

1. What are some best practices for writing commit messages?

Here are some best practices for writing clear and effective commit messages:

1. Use the imperative mood: Write commit messages as if giving a command (e.g., "Fix bug" or "Add feature").

2. Be concise: Keep the subject line (the first line of the commit message) under 50 characters.

3. Provide context: If necessary, include a detailed description of the changes in the body of the message (after a blank line).

4. Reference issue numbers: If the commit addresses a specific issue, reference it (e.g., "Fixes #42").

5. Explain "why": If the change is complex, explain why the change was made, not just what was done.