|  |  |  |
| --- | --- | --- |
| SCHOOL OF INFORMATION AND TECHNOLOGY | | |
| NAME: Magno, Ronnie L. | DATE PERFORMED: 11/21/24 |  |
| Section: IDC2 | DATE SUBMITTED: 11/21/24 |

# 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 designed to efficiently track changes in source code during software development. Created by Linus Torvalds in 2005, Git allows multiple developers to collaborate on projects without interfering with each other's work. Each developer has a complete local copy of the repository, which includes the entire history of changes, enabling them to work offline and perform operations quickly. Git supports non-linear development through branching and merging, allowing developers to experiment with new features or fixes in isolated environments before integrating them into the main codebase
* The importance of Git in software development lies in its ability to enhance collaboration and streamline workflows. By providing a robust framework for tracking changes and managing different versions of code, Git minimizes the risk of conflicts and errors that can arise when multiple developers work on the same project. Approximately 85% of developers believe that Git simplifies collaboration, making it easier to implement features and fix bugs efficiently. Additionally, Git's integration with continuous integration/continuous deployment (CI/CD) tools facilitates faster release cycles and more responsive development processes, allowing teams to adapt quickly to feedback and changing requirements

1. How does Git track changes in a project?

* Git tracks changes in a project by utilizing a structured system comprising the working directory, the staging area, and the repository. When developers modify files, Git categorizes them as either tracked or untracked. Tracked files are monitored for changes and can be in states such as modified or staged. To begin tracking a file, developers use the `git add` command to move it to the staging area, indicating that it is ready to be committed. Once staged, changes can be committed to the repository using the `git commit` command, which saves a snapshot of the project along with metadata like author and timestamp. This process allows Git to maintain a comprehensive history of all changes, enabling developers to review modifications, revert to previous versions, and collaborate effectively on projects.

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

* A local repository in Git is a version-controlled directory that resides on an individual developer's machine. It allows developers to perform all Git operations, such as adding files, committing changes, and creating branches, without needing an internet connection. This local setup provides the full benefits of version control, enabling users to track changes, revert to previous versions, and experiment with new features in isolation. When a developer initializes a Git repository using the command git init, a hidden .git folder is created within the directory, containing all the necessary metadata for tracking changes. Since all operations are performed locally, developers can work at their own pace and manage their code independently before sharing it with others.

In contrast, a remote repository is hosted on a server or a cloud platform (such as GitHub, Bitbucket, or GitLab) and serves as a centralized location for collaboration among multiple developers. Remote repositories allow team members to share their code and changes with one another through commands like git push to upload local commits and git pull to download updates made by others. Unlike local repositories, remote repositories are accessible over the internet and facilitate collaboration by allowing multiple contributors to work on the same project simultaneously. They also provide features for managing contributions, such as pull requests and code reviews, which help maintain code quality and streamline integration processes among team members working from different locations.

1. What are the basic Git commands?

Basic Git commands:

* **git init** - Initializes a new local Git repository in the current directory.
* **git clone** <repository> - Creates a local copy of a remote repository.
* **git add** <filename> - Stages changes to be included in the next commit. You can stage all changes using git add ..
* **git commit -m** "message" - Records the staged changes in the repository with a descriptive message.
* **git status** - Displays the state of the working directory and staging area, showing which files are staged, modified, or untracked.
* **git push origin** <branch> - Uploads local commits to the specified branch of a remote repository.
* **git pull** - Fetches and merges changes from the remote repository into the current branch.
* **git branch** <branch-name> - Creates a new branch or lists existing branches when used without an argument.
* **git checkout** <branch-name> - Switches to the specified branch, allowing you to work on it.
* **git merge** <branch-name> - Merges changes from the specified branch into the current branch.

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

* To check the status of a Git repository, you use the git status command. This command provides crucial information about the current state of your working directory and staging area. When executed, git status displays which files are tracked, untracked, modified, or staged for the next commit. It also indicates if there are any changes that have not yet been committed or if the working tree is clean, meaning there are no changes to commit. This command is essential for understanding what modifications have been made and what actions you may need to take next, such as staging files or committing changes.

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

* Branches in Git allow developers to work on different features or fixes independently from the main codebase, facilitating parallel development without interference. This isolation helps maintain a clean and stable codebase while enabling collaboration among team members.
* To create a new branch, use `git branch <branch-name>`, and to switch to it, use `git checkout <branch-name>`. Alternatively, you can create and switch to a new branch in one step with `git checkout -b <branch-name>` or `git switch -c <branch-name>`. To switch between existing branches, simply use `git checkout <branch-name>` or `git switch <branch-name>`.

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

* GitLab Desktop and GitHub are both platforms that facilitate version control and collaboration on software projects, but they serve different purposes and have distinct features compared to Git itself.
* Git is a distributed version control system that allows developers to track changes in their codebase, manage versions, and collaborate with others. It operates primarily through command-line interfaces (CLI) and provides the foundational tools for version control without any built-in graphical user interface (GUI). Git can be used independently on a local machine or integrated with various hosting services.
* GitHub is a cloud-based platform that builds on Git's capabilities by providing a user-friendly interface for hosting Git repositories online. It offers additional features such as issue tracking, project management tools, and collaboration functionalities, making it easier for teams to work together on projects. GitHub also supports integrations with third-party applications and services, enhancing its functionality for developers.
* GitLab serves a similar purpose but distinguishes itself by offering a more comprehensive suite of tools that includes built-in Continuous Integration/Continuous Deployment (CI/CD) capabilities. GitLab allows users to manage the entire software development lifecycle within a single platform, from code repository management to deployment. Additionally, GitLab can be self-hosted, giving organizations greater control over their repositories and data.

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

* To connect a local Git repository to a GitLab or GitHub repository, first create a new repository on the platform without initializing it with any files. Then, navigate to your local project directory in the terminal and run git init to initialize it as a Git repository if it isn't already. Next, stage your files with git add . and commit them using git commit -m "Initial commit". After that, add the remote repository URL with the command git remote add origin <repository-url>, replacing <repository-url> with the URL of your GitLab or GitHub repository. Finally, push your local commits to the remote repository using git push -u origin main (or master if applicable), establishing the connection between your local and remote repositories.

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

STEPS:

* **Create a Repository**: The repository owner sets up a new project space on GitHub or GitLab for collaboration.
* **Add Collaborators**: The repository owner invites team members by adding their usernames, granting them access to contribute.
* **Fork the Repository** (if applicable): Collaborators create a personal copy of someone else's repository to make changes independently.
* **Clone the Repository**: Collaborators download a local copy of the repository (or their fork) to their machine for editing.
* **Create a Branch**: A new branch is created by collaborators to work on features or fixes without affecting the main codebase.
* **Make Changes**: Collaborators edit files in their local branch according to the project requirements.
* **Stage and Commit Changes**: Changes are prepared for saving (staged) and then saved (committed) with a descriptive message.
* **Pull Latest Changes**: Collaborators update their local repository with the latest changes from the remote repository to avoid conflicts.
* **Push Changes**: Collaborators upload their committed changes from their local branch to the remote repository.
* **Create a Pull Request** (GitHub) or Merge Request (GitLab): After pushing, collaborators propose their changes for review and merging into the main branch.
* **Review and Merge:** The repository owner and team members evaluate the proposed changes and merge them if they meet project standards.

1. How do you resolve merge conflicts in Git?

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

* Step 1: The easiest way to resolve a conflicted file is to open it and make any necessary changes.
* Step 2: After editing the file, we can use the git add a command to stage the new merged content.
* Step 3: The final step is to create a new commit with the help of the git commit command.
* Step 4: Git will create a new merge commit to finalize the merge.

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

* A pull request (PR) is a proposal to merge changes made in one branch of a repository into another, typically from a feature branch into the main branch. It serves as a formal request for code review and integration, allowing developers to discuss and evaluate the proposed changes before they are incorporated into the main codebase. Pull requests provide a platform for transparent communication among team members, enabling them to review the differences between branches, leave comments, and suggest modifications.

Pull requests are essential in GitHub because they facilitate collaboration and maintain code quality. They help teams manage contributions effectively, especially in larger projects where multiple developers may be working simultaneously. By using pull requests, teams can conduct thorough code reviews, ensure that new features or bug fixes meet project standards, and maintain a well-documented history of changes. This process enhances accountability and fosters learning opportunities within the team, as developers receive feedback on their code and can track discussions related to specific changes .

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

* **Be Clear and Concise-** A commit message should clearly summarize the changes made, avoiding vague descriptions.
* **Include Relevant Context**- Include context such as issue numbers or discussions to explain the motivation behind the commit.
* **Mention Bug Fixes**: Reference and describe any bugs fixed in the commit, providing details of the problem and the solution.
* **Organize Commits logically**- Break changes into smaller, self-contained commits that are easy to review and manage.
* **Use Imperative Verbs-** Start commit messages with action verbs like “Add,” “Fix,” or “Refactor” to describe the changes.
* **Summarize with a Subject Line**- Begin with a brief, meaningful subject line (around 50 characters) that summarizes the commit’s purpose.
* **Test and Proofread-** Test your changes thoroughly and proofread your commit message for errors before committing.
* **Follow a Template-** Use a consistent commit message template to maintain structure and clarity across your team or project.
* **Update Commit Messages**- Use Git's --amend option to update a commit message instead of creating a new commit to avoid clutter.

**REFERENCES:**

Afreen, S. (2024, July 15). *How to resolve merge conflicts in Git?* Simplilearn.com. <https://www.simplilearn.com/tutorials/git-tutorial/merge-conflicts-in-git#what_is_a_git_merge_conflict>

*Basic Git commands | Bitbucket Data Center 9.3 | Atlassian Documentation*. (n.d.). Atlassian Documentation. <https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>

Brian, K. (2023, July 14). *9 Best practices for writing Good Git Commit Messages*. <https://www.linkedin.com/pulse/7-best-practices-writing-good-git-commitmessages-kirinyet-brian>

GeeksforGeeks. (2024, May 22). *Git status*. GeeksforGeeks. https://www.geeksforgeeks.org/git-status/

*Git - Recording changes to the repository*. (n.d.). <https://git-scm.com/book/ms/v2/Git-Basics-Recording-Changes-to-the-Repository>

McKenzie, C. (2023, August 29). How to git push an existing project to GitHub. *How to git push an existing project to GitHub*. <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-push-an-existing-project-to-GitHub>

Radnai, G. (2024, February 29). Git’s Significance in Software Development. *Git’s Significance in Software Development*. <https://blog.rheinwerk-computing.com/gits-significance-in-software-development>

*Repositories | Git tutorial | Nulab*. (n.d.). Nulab. <https://nulab.com/learn/software-development/git-tutorial/git-basics/repositories/>

*What is a pull request & why is it important for code review? | LinearB Blog*. (2024, May 7). <https://linearb.io/blog/what-is-a-pull-request>