|  |  |  |
| --- | --- | --- |
| SCHOOL OF INFORMATION AND TECHNOLOGY | | |
| NAME: Caccam, Jesus Allyson | 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 DevOps tool used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development.
* Git is an essential tool in software development, functioning as a distributed version control system that allows developers to monitor changes, collaborate effectively, and manage code efficiently. Its distributed architecture provides each developer with a full local copy of the repository, enabling offline work and significantly improving speed and efficiency.

1. How does Git track changes in a project?

* As you edit files, Git sees them as modified, because you’ve changed them since your last commit. As you work, you selectively stage these modified files and then commit all those staged changes, and the cycle repeats

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

* A local repository is stored on an individual developer's machine and contains all project files along with their complete version history, allowing developers to perform Git operations such as committing changes and creating branches without needing an internet connection. It can be created using the command git init, enabling developers to experiment freely without affecting others' work, as changes made locally remain private until pushed to a remote repository.
* Remote repositories are typically created on hosting services and linked to local repositories using commands like git remote add <name> <url>, facilitating collaboration and providing a backup of the project's history.

1. What are the basic Git commands?

* **Git Clone -** Git clone is a command for downloading existing source code from a remote repository (like Github, for example). In other words, Git clone basically makes an identical copy of the latest version of a project in a repository and saves it to your computer.
* **Git Branch -** Branches are highly important in the git world. By using branches, several developers are able to work in parallel on the same project simultaneously. We can use the git branch command for creating, listing and deleting branches
* **Git Checkout -** This is also one of the most used Git commands. To work in a branch, first you need to switch to it. We use git checkout mostly for switching from one branch to another. We can also use it for checking out files and commits.
* **Git Status -** The Git status command gives us all the necessary information about the current branch. We can gather information like:Whether the current branch is up to date, whether there is anything to commit, push or pull, whether there are files staged, unstaged or untracked or whether there are files created, modified or deleted.
* **Git Add -** When we create, modify or delete a file, these changes will happen in our local and won't be included in the next commit (unless we change the configurations).
* **Git Commit -** This is maybe the most-used command of Git. Once we reach a certain point in development, we want to save our changes (maybe after a specific task or issue). Git commit is like setting a checkpoint in the development process which you can go back to later if needed.
* **Git push -** After committing your changes, the next thing you want to do is send your changes to the remote server. Git push uploads your commits to the remote repository.
* **Git pull -** The git pull command is used to get updates from the remote repo. This command is a combination of git fetch and git merge which means that, when we use git pull, it gets the updates from remote repository (git fetch) and immediately applies the latest changes in your local (git merge).
* **Git revert -** Sometimes we need to undo the changes that we've made. There are various ways to undo our changes locally or remotely (depends on what we need), but we must carefully use these commands to avoid unwanted deletions. A safer way that we can undo our commits is by using git revert.
* **Git merge -** When you've completed development in your branch and everything works fine, the final step is merging the branch with the parent branch (dev or master). This is done with the git merge command. Git merge basically integrates your feature branch with all of its commits back to the dev (or master) branch. It's important to remember that you first need to be on the specific branch that you want to merge with your feature branch.

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

* **Git Status -** The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven’t, and which files aren’t being tracked by Git. Status output does not show you any information regarding the committed project history. For this, you need to use git log.

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

* A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.
* The git branch command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the git checkout and git merge commands.

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

* **Git -** is a free and open-source distributed version control system designed to handle projects of any size with speed and efficiency. Unlike centralized systems, Git allows developers to work independently with a full copy of the codebase on their local machines.
* **GitHub -** GitHub is a cloud-based hosting service that provides a user-friendly web interface for managing Git repositories. It allows developers to store, share, and collaborate on their codebase with teams or the open-source community. In 2018, GitHub was acquired by Microsoft, further solidifying its position as a leading platform for software development.
* **GitLab -** GitLab is a web-based platform that streamlines development workflows. It does this by merging Git repository management with continuous integration (CI), deployment, and collaboration tools. GitLab facilitates code versioning and team cooperation and automates the pipeline from development to deployment, simplifying the entire software lifecycle within its unified platform.
* Git is a distributed version control system that does not provide hosting capabilities for repositories. In contrast, GitHub is a web-based platform that hosts Git repositories and focuses on collaboration and community engagement, while GitLab also hosts repositories but emphasizes integrated DevOps tools, including built-in continuous integration and delivery (CI/CD) features. Both platforms offer user-friendly interfaces, but they differ in their primary focus and functionality enhancements over Git.

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 a README or other files. Next, open your terminal and navigate to your local project directory; if it's not already a Git repository, run git init to initialize it. Stage all project files for the first commit using git add ., then commit the changes with a message using git commit -m "Initial commit". After that, link your local repository to the remote one by adding its URL with the command git remote add origin <REMOTE-URL>, replacing <REMOTE-URL> with the actual URL of your GitHub or GitLab repository. Finally, push your local commits to the remote repository using git push -u origin main, or replace "main" with "master" if that is your default branch. Following these steps will successfully connect your local Git repository to a remote GitLab or GitHub repository, enabling effective collaboration and code management.

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

* Review collaboration policy in the project
* Fork the repo
* Clone forked repo
* Update master branch
* Create a branch - Make our improvements
* Create a pull request
* Sync a fork
* Rebasing a pull request

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 is a proposal to merge a set of changes from one branch into another. In a pull request, collaborators can review and discuss the proposed set of changes before they integrate the changes into the main codebase.

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

* **Be Clear and Concise**: Write commit messages that summarize changes clearly, avoiding vague descriptions like "Fixed a bug." Instead, provide specific details about the purpose or impact of the changes.
* **Use Imperative Mood**: Start your commit messages with imperative verbs, such as "Add," "Fix," or "Update," to convey the action taken. This approach makes the messages more actionable and consistent.
* **Structure Your Messages**: A well-structured commit message typically includes a brief subject line (ideally under 50 characters) followed by a more detailed description. Separate the subject from the body with a blank line.
* **Follow the 50/72 Rule**: Keep the subject line to a maximum of 50 characters and format the body of the message to wrap at 72 characters for better readability.
* **Include Relevant Context**: Mention any related issue numbers or discussions in your commit messages to provide context and facilitate collaboration among team members.
* **Organize Commits Logically**: Break larger changes into smaller, logical commits that represent self-contained units of work, making it easier to review and manage changes.
* **Proofread Your Messages**: Before committing, double-check your changes for errors and ensure that your commit message is free from typos or grammatical mistakes.
* **Use Templates if Possible**: Establishing a commit message template for your team can help maintain consistency and ensure that all necessary information is included.

Reference:

*How to collaborate with a GitHub project*. (n.d.). Gist. <https://gist.github.com/neklaf/9002d3acccf6b6e448db5c4c4e8764c0>

Pina, N. (2022, January 4). *How to Write Better Git Commit Messages – A Step-By-Step Guide*. freeCodeCamp.org. <https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/>

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#:~:text=Step%201%3A%20The%20easiest%20way,of%20the%20git%20commit%20command>.

McKenzie, C. (2024, March 1). How to git push an existing project to GitLab. *thereserverside*. <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-add-and-push-an-existing-project-to-GitLab#:~:text=Upload%20your%20existing%20project%20to%20GitLab&text=To%20do%20this%2C%20issue%20a,along%20with%20the%20%2Du%20switch.&text=Legacy%20Git%20repositories%20create%20a,matches%20your%20local%20Git%20repository>.

Chinonso, E. (2024, April 30). *Git vs. GitHub vs. GitLab*. DevOps Blog. <https://kodekloud.com/blog/git-vs-github-vs-gitlab/#:~:text=your%20project's%20needs.-,Key%20Takeaways,CD%20pipelines%20and%20security%20features>..

*Git Review - remotes/locals and forking/cloning - Front-End Engineering Curriculum - Turing School of Software and Design*. (n.d.). <https://frontend.turing.edu/lessons/module-2/git-forking-vs-cloning.html>

Haffiyan, R. D. (2020, March 9). *Why Git is important in software development?* <https://www.linkedin.com/pulse/why-git-important-software-development-razaqa-dhafin-haffiyan>

<https://git-scm.com/book/ms/v2/Git-Basics-Recording-Changes-to-the-Repository#:~:text=As%20you%20edit%20files%2C%20Git,changes%2C%20and%20the%20cycle%20repeats>.

freeCodeCamp. (2020, January 19). *10 Git commands every developer should know*. freeCodeCamp.org. <https://www.freecodecamp.org/news/10-important-git-commands-that-every-developer-should-know/>