|  |  |  |
| --- | --- | --- |
| SCHOOL OF INFORMATION AND TECHNOLOGY | | |
| NAME: LAURIANO, RYAN G. | 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?

**What is Git?**

Git is a distributed version control system (dVCS). As the name suggests, version control is all about controlling and tracking different versions of a given project. A VCS tracks and records changes to any file (or a group of files) allowing you to recall specific iterations later on or as needed. VCSs are sometimes called source code management (SCM) or revision control systems (RCS).

**Why is it important in Software Development?**

Imagine working on a software project with your team, where both you and your teammates are editing the same file. By the time you’ve finished your changes, your teammate may also have completed theirs, leaving you with the challenge of combining both sets of updates. This can quickly become confusing, as the only way to merge the changes might be to manually copy and paste the code until everything fits together correctly.

To simplify this process and avoid such confusion, you can use an amazing tool called Git. Git is a free, open-source, distributed version control system designed to handle projects of any size with speed and efficiency. It’s commonly used for tracking changes in files and helps teams work together seamlessly without worrying about file conflicts. Additionally, Git keeps a history of all changes, making collaboration and project management much easier.

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?

**Local Repository vs. Remote Repository**

A **local repository** is hosted on a local machine for an individual user. A local repository contains copies of objects in the central repository. There are two ways to update your local repository with contents of the central repository:

* You can copy objects from the central repository to your local repository.
* You can check out objects from the central repository to your local repository.

A **remote repository** is hosted on a remote (this could be on the internet or an off-site server; it could even be the same machine in a different path) and is shared among multiple team members.

1. What are the basic Git commands?

|  |  |
| --- | --- |
| **Basic Git Commands** | |
| **Setup Commands** | |
| git config --global user.name | Sets your Git username. |
| git config --global user.email | Sets your Git email address. |
| git config --list | Displays the Git configuration settings. |
| git init | Initializes a new Git repository. |
| git clone | Copies an existing repository from a URL. |
| **Basic Workflow Commands** | |
| git status | Shows the current state of the working directory and staging area. |
| git add | Stages a specific file for the next commit. |
| git add . | Stages all changes in the current directory. |
| git commit -m | Saves the staged changes with a descriptive commit message. |
| git push | Uploads local repository changes to a remote repository. |
| git pull | Fetches and merges changes from a remote repository into the current branch. |
| git fetch | Retrieves changes from the remote repository but doesn’t merge them. |
| git merge | Combines a specified branch into the current branch. |
| **Viewing History Commands** | |
| git log | Shows the commit history. |
| git log --oneline | Displays a simplified version of the commit history. |
| git show | Shows detailed information about a specific commit. |

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

You can inspect a Git repository by using the git status command. This command allows you to see which changes have been staged, which haven’t, and which files aren’t being tracked by Git.

You should try and remember that status output does not show you any information regarding the committed project history. For this, you need to use the git log command.

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

Allow multiple people or tasks to work on different parts of a project without affecting the main codebase. With branches, you can develop new features, fix bugs, or try out experimental changes independently, and then merge them back into the main branch when they’re ready.

To create a new branch, use: **git branch**

To switch to an existing branch, use: **git checkout**

You can also create a new branch and immediately switch to it with: **git checkout -b**

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

**What is GitLab Desktop?**

GitLab Desktop is the client application that integrates with GitLab, a web-based Git repository manager. It enhances the Git workflow by providing a graphical interface for managing repositories, branches, commits, and merges. This desktop application is designed to simplify the interaction with Git repositories, making version control more accessible to developers of all skill levels.

**What is GitHub?**

GitHub is a for-profit company that offers a cloud-based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

**How are they different from Git?**

**GitLab Desktop** and **GitHub** are platforms that build on **Git’s** core functionality. They provide a user-friendly interface and cloud-based services for managing Git repositories, allowing for easier collaboration, code sharing, and project management.

**GitLab Desktop** is a client for interacting with GitLab repositories, whereas **GitHub** is a web service for hosting and collaborating on Git repositories.

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

Step 1: Create a new directory: *mkdir my-first-git-project*

Step 2: Navigate to the project directory: *cd my-git-project*

Step 3: Initialize a Git repository: *git init*

Step 4: Create a new file: *nano magic.txt*

Step 5: Check the status of your repository: *git status*

Step 6: Add the file to the staging area: *git add magic.txt*

Step 7: Make a local commit: *git commit -m "Initial commit"*

Step 8: Add a remote repository (GitHub or GitLab): *git remote add origin*

Step 9: Push your changes to the remote repository: *git push -u origin master*

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

To collaborate with others, Git and GitHub/GitLab provide easy ways to fork repositories, submit pull/merge requests, and review code.

* **Fork the Repository:**  
  Forking allows you to create a personal copy of a repository. This lets you make changes without affecting the original project. On GitHub, click the "Fork" button on the repository's page. On GitLab, use the "Fork" option as well. This creates a copy of the repository under your account.
* **Clone the Forked Repository:**  
  After forking, clone the repository to your local machine using the following command:

***git clone <repository-url>***

* **Create a New Branch:**  
  Before making changes, it's best practice to create a new branch. This allows you to work on features or fixes separately from the main codebase. You can create a new branch with:

***git checkout -b <branch-name>***

* **Make Changes Locally:**  
  Edit the files in your repository as needed. After making changes, check the status with git status, add the changes to the staging area using git add, and commit them with:

***git commit -m "Your commit message"***

* **Push Changes to GitHub/GitLab:**  
  After committing your changes, push them to your forked repository on GitHub or GitLab with:

***git push origin <branch-name>***

* **Submit a Pull Request (GitHub) / Merge Request (GitLab):**
* On GitHub, go to your repository page and click on the "New Pull Request" button. Select the branch with your changes and compare it with the main repository’s branch (usually main or master).
* On GitLab, create a "Merge Request" to propose your changes, comparing your feature branch with the target branch.
* **Merge the Changes:**  
  Once the pull or merge request is approved, it can be merged into the main repository. The project maintainers or collaborators will merge the changes into the main branch.

1. How do you resolve merge conflicts in Git?

Sometimes, you’ll encounter merge conflicts when changes from different branches are incompatible. Git will notify you of the conflict and mark the affected files. You’ll need to manually resolve the conflict by editing the file, then add and commit it again.

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?

When you’re ready to contribute to a project, you can open a **Pull Request** (GitHub) or **Merge Request** (GitLab). This lets the project maintainer review your changes before merging them.

A pull request in GitHub is a way to propose changes from your branch to another, usually the main branch. It lets project maintainers review, discuss, and approve updates before merging them, ensuring code quality and encouraging collaboration.

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

* **Commit Often**: Small, freent commits make it easier to track changes.
* **Write Meaningful Commit Messages**: Describe what changed and why. Use the imperative mood, if applied, this commit will your subject line here.
* **Use Descriptive Subject Lines**: Keep the first line of your commit message short (50-72 characters) and informative.
* **Provide Additional Details**: If needed, add a body to your message explaining the context or reasoning for the changes.
* **Branch for Each Feature**: Keep your main branch clean by creating new branches for each task.
* **Review Before Merging**: Use pull/merge requests to ensure code quality.
* **Test Before Committing**: Ensure the changes work as intended to avoid committing broken or incomplete code.
* **Follow a Consistent Format**: Use a standard format for all messages, such as "Subject: Description" or include issue references (e.g., "Fix #123: Resolve login bug").
* **Focus on One Change per Commit**: Avoid bundling unrelated changes into a single commit to make debugging and reverting easier.
* **Avoid Temporary Messages**: Don’t use vague messages like "Fix stuff" or "Update code"; these don’t help track specific changes.

References:

Atlassian. (n.d.-a). *Basic Git Commands | Atlassian Git Tutorial*. <https://www.atlassian.com/git/glossary#commands>

Atlassian. (n.d.-b). *Git Branch | Atlassian Git Tutorial*. <https://www.atlassian.com/git/tutorials/using-branches>

*Git - Recording changes to the repository*. (n.d.). <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>.

Holcombe, J. (2022, January 28). *How to inspect a GIT repository on GitHub*. GreenGeeks. <https://www.greengeeks.com/tutorials/inspect-a-git-repository/#:~:text=You%20can%20inspect%20a%20Git,t%20being%20tracked%20by%20Git>.

Kinsta. (2023, November 17). *What is GitHub? A beginner’s introduction to GitHub*. Kinsta®. <https://kinsta.com/knowledgebase/what-is-github/>

Novković, A. (2024, November 16). Git, GitHub, and GitLab: a beginner’s guide to version control and collaboration. *Medium*. <https://medium.com/@anaiscoding/git-github-and-gitlab-a-beginners-guide-to-version-control-and-collaboration-bfd6a7899d7e>

*Repositories | Git tutorial | Nulab*. (n.d.). Nulab. <https://nulab.com/learn/software-development/git-tutorial/git-basics/repositories/#:~:text=A%20remote%20repository%20is%20hosted,machine%20for%20an%20individual%20user>.

Rs, G. (2024, November 6). GitLab Desktop: Everything you need to know in 2024. *Devzery Latest*. <https://www.devzery.com/post/your-guide-to-gitlab-desktop-everything-you-need-to-know-in-2024#:~:text=GitLab%20Desktop%20is%20the%20client,branches%2C%20commits%2C%20and%20merges>.

Saxena, A. (2024, February 4). *Local Repo connect to a remote repository:* <https://www.linkedin.com/pulse/local-repo-connect-remote-repository-aakriti-saxena-mggpf>

Worsley, S. (2023, October 18). What is Git? - The Complete Guide to Git. *Datacomp*. Retrieved November 20, 2024, from <https://www.datacamp.com/blog/all-about-git>