|  |  |  |
| --- | --- | --- |
| SCHOOL OF INFORMATION AND TECHNOLOGY | | |
| NAME: BUSONGAN, DEMIRAYE-ANNE | DATE PERFORMED: NOV 20, 2024 |  |
| Section: IDC1 | DATE SUBMITTED: NOV 20, 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 designed to handle everything from small to very large projects with speed and efficiency. It allows multiple developers to work on a project simultaneously without interfering with each other's changes. Git is crucial in software development because it enables collaboration within developers as they work together on the same codebase, merging their changes seamlessly whether in they’re in their offices, or different locations. It allows Version Tracking where every changes that is made on the code is recorded, allowing developers to revert to previous versions if needed. It also enables Branching & Merging where developers can create branches to experiment with new features without affecting the main codebase, which can later be merged back into the main project.
2. **How does Git track changes in a project?**- Git works by taking snapshots of the project’s file system to track changes over time. The staging area is where you prepare changes by selecting files with `git add`. This step allows you to decide which modifications to include in the next commit. Afterward, using `git commit`, Git records a snapshot of the files in their current state, storing it along with a unique commit hash, author information, and timestamp, creating a permanent record of that version in the project history.
3. **What is the difference between a local repository and a remote repository in Git?**- A local repository is the version of a project stored on a developer's own computer. It allows developers to make changes, commit updates, and create branches independently, without affecting others. This gives each developer the freedom to work on their tasks without interference. A remote repository, on the other hand, is hosted on a server, such as GitHub or GitLab, and acts as a central hub for the entire team. It’s where developers can push their changes to share with others and pull updates made by their teammates. The remote repository ensures that everyone stays in sync and can collaborate efficiently on the same project.
4. **What are the basic Git commands?**- git init: Initializes a new Git repository.  
   - git clone <url>: Copies an existing remote repository to your local machine.  
   - git status: Shows the current state of your working directory and staging area.  
   - git add <file>: Stages changes to be included in the next commit.  
   - git commit -m "<message>": Commits the staged changes with a message.  
   - git push <remote> <branch>: Uploads your local commits to the remote repository.  
   - git pull <remote> <branch>: Fetches and merges changes from the remote repository.  
   - git branch: Lists branches or creates a new branch.  
   - git checkout <branch>: Switches to a different branch or restores files.  
   - git merge <branch>: Merges changes from one branch into another.  
   - git log: Shows the commit history.  
   - git remote: Manages remote repositories (view, add, remove).  
   - git reset: Unstages changes or removes commits.  
   - git rm <file>: Removes files from the working directory and stages for deletion.  
   - git diff: Shows the differences between changes in the working directory and the staged changes.
5. **How do you check the status of a Git repository?**- To check the status of the a Git repository, we use “git status”, this command display information about staged files, unstaged changes, untracked files and which branch you are currently on.
6. **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 simultaneously without affecting the main codebase usually called main or master.

Creating a Branch: Use the command: git branch [branch-name]  
Switching Between Branches: git checkout [branch-name]

Alternatively, you can create and switch to a new branch in one command: git checkout -b [branch-name]

1. **What are GitLab Desktop and GitHub, and how are they different from Git?**- GitLab Desktop and GitHub are platforms that provide hosting for Git repositories. They offer additional features beyond version control, for GitHub, it is prrimarily known for its social coding features, it allows users to collaborate on projects through pull requests, issues tracking, and project management tools. On the contrary GitLab offers similar functionalities but emphasizes continuous integration/continuous deployment (CI/CD) capabilities directly integrated into its platform. Although both platforms facilitate collaboration they differ in their specific tools and community features
2. **How do you connect a local Git repository to a GitLab or GitHub repository?**- To connect your local repository to a remote repository on GitHub or GitLab:  
   Create an empty repository on your chosen platform.  
   In your local repository, run: git remote add origin [url]  
   Push your local commits to the remote repository: git push -u origin master  
   This sets up your local repository to track changes against the remote one.
3. **What are the steps to collaborate with others using GitLab or GitHub?**
   1. **Set Up Your Repository**
      1. Create an Account: If you don't have one, sign up for GitHub or GitLab.
      2. Create a New Repository: Click on "New Repository" and provide a name and description. Choose whether it should be public or private.
      3. Initialize with a README: This helps provide context about the project.
   2. **Invite Collaborators**
      1. GitHub:
      2. Navigate to your repository's main page.
      3. Click on the Settings tab.
      4. Under the Access section, click on Collaborators.
      5. Click on Add People, enter their GitHub username or email, and send the invitation.
      6. GitLab:
      7. Go to your project page.
      8. Click on Members in the sidebar.
      9. Enter the username or email of the person you want to invite, select their role (e.g., Developer), and click Invite.
   3. **Clone the Repository**
      1. Collaborators should clone the repository to their local machine using:   
         git clone https://github.com/yourAccountName/your-repo-url.gi
   4. **Create a Branch**
      1. To avoid conflicts, collaborators should create a new branch for their work:   
         git checkout -b feature-branch-name
   5. **Make Changes and Commit**
      1. After making changes to files, add them to the staging area and commit:   
         git add .  
         git commit -m "Description of changes"
   6. **Push Changes to Remote Repository**
      1. Push your changes to the remote repository:  
         git push origin feature-branch-name
   7. **Open a Pull Request (GitHub) / Merge Request (GitLab)**
      1. After pushing changes, open a pull request (PR) on GitHub or a merge request (MR) on GitLab:
      2. Go to the repository page and click on Pull Requests (GitHub) or Merge Requests (GitLab).
      3. Click on New Pull Request/Merge Request, select your branch, and provide details about your changes.
   8. **Review and Merge Changes**
      1. Collaborators can review the PR/MR and discuss any necessary changes. Once approved, merge it into the main branch.
   9. **Sync Changes Regularly**
      1. Regularly pull changes from the main branch to stay updated:  
         git pull origin main

1. **How do you resolve merge conflicts in Git?**- Merge conflicts occur when two branches have competing changes that cannot be automatically merged. To resolve them:  
   Attempt to merge branches using: git merge [branch-name]  
     
   If conflicts occur, open the conflicting files and look for markers (<<<<<<<, =======, >>>>>>>) indicating conflicting areas.  
   Edit these sections manually to resolve conflicts.  
   After resolving conflicts, stage the changes: git add [file]  
   Complete the merge with: git commit  
     
   This process ensures that all conflicting changes are addressed before finalizing the merge.
2. **What is a pull request, and why is it used in GitHub?**- A pull request (PR) is a request by a contributor to merge their branch into another branch (usually main or master). It serves several purposes which include facilitates code review by allowing team members to discuss proposed changes before they are merged. It helps maintain code quality by enabling checks like automated tests before integration. Lastly, it provides documentation of what was changed and why through comments associated with each PR
3. **What are some best practices for writing commit messages?**- Best practices for writing effective commit messages include:  
   Use Imperative Mood: Write messages as commands (e.g., "Add feature" instead of "Added feature").  
   Be Concise but Descriptive: Summarize what was changed in 50 characters or less; use more detail in subsequent lines if necessary.  
   Reference Issues: If applicable, reference issue numbers related to your commits for context.