**Detailed Lab Guide for Git Workflow with Merging, Remote Operations, and Branch Management**

**1. Initial Setup and Creating Files**

**Step 1: Switch to 2ndbranch**

git checkout 2ndbranch

**Step 2: Create and Verify Files**

cat file1

ls

history

**Step 3: Create a New File and Add Content**

touch file3

vi file3 # Add "my name is raman"

**Step 4: Commit Changes to 2ndbranch**

git add .

git commit -m "added file3 in testing branch"

**2. Merging Branches**

**Step 5: Switch to master Branch and Merge**

git checkout master

git merge 2ndbranch

**Step 6: Verify Merged Content**

cat file3 # Expect to see "my name is raman khanna"

**3. Working with Remote Repositories**

**Step 7: Push Changes to Remote Repo**

git checkout 2ndbranch

git push origin main

**Step 8: Clean Up Files**

rm -rf file1 file3 file-remote

git add --all

git commit -m "2ndbranch files deleted"

**Step 9: Create a New File in 2ndbranch**

vi file4-2ndbranch

git add .

git commit -m "added4th file in 2ndbranch"

git push origin 2ndbranch

**Step 10: Add a File Directly on GitHub and Pull Changes**

1. Add file5-2ndbranch manually on GitHub in 2ndbranch.
2. Pull changes:

git pull origin 2ndbranch

cat file5-2ndbranch # Verify the file

**4. Handling HEAD Error during Pull**

**Step 11: Resolving HEAD Errors**

If you've pushed content and face conflicts while pulling:

git fetch --all

git reset --hard origin/master # For master branch

For a specific branch:

git checkout your\_branch\_name\_here

git reset --hard origin/your\_branch\_name\_here

**5. Branch Deletion**

**Step 12: Deleting Local Branch**

Switch to master branch before deleting 2ndbranch:

git checkout master

git branch -d 2ndbranch # Safe delete (merged branches only)

Force delete unmerged branches:

git branch -D 2ndbranch

**Step 13: Deleting Remote Branch**

git push --delete origin 2ndbranch

**6. Important Notes**

* Ensure you perform git fetch before performing any reset or pull to sync remote changes locally.
* For safety, always inspect the output of git status before committing or merging.
* Use git log to track commit history and identify merge conflicts efficiently.

This structured lab guide should help you practice essential Git operations with clarity and precision.

**Hostname and Subdomain isolation** are ways to manage and protect your GitHub instance by customizing how it behaves with the internet domain. Here's what they mean in simpler terms:

**1. Hostname Isolation**

* This refers to the ability to assign a **unique domain name** (hostname) to your GitHub Enterprise Cloud instance.
* For example, instead of accessing your GitHub instance through a generic domain like github.com, you can access it through a custom hostname like github.yourcompany.com.
* **Why it's useful**: This makes your instance feel like it's entirely separate from GitHub's general cloud platform. It adds a layer of security and branding, ensuring that the URLs for your internal resources look distinct and professional.

**2. Subdomain Isolation**

* Subdomain isolation lets you divide your GitHub Enterprise Cloud into multiple **subdomains**, each with its own access and settings.
* For example, you could have dev.github.yourcompany.com for your development team, ops.github.yourcompany.com for the operations team, and so on.
* **Why it's useful**: It makes it easier to manage different parts of your organization or different environments (like dev, staging, and production) under the same GitHub Enterprise Cloud instance. Each subdomain can be tailored for specific use cases or departments while still using the same central GitHub platform.

**How it helps:**

* **Isolation** gives better control over traffic, security, and user access. You can limit access to specific teams or environments based on the subdomain they use.
* **Flexibility** allows you to scale and customize the GitHub instance without worrying about overlapping or conflicting settings.

In short, hostname and subdomain isolation help in managing and securing your organization's GitHub environment by creating separate, customizable spaces for different teams or use cases, all under one GitHub Enterprise Cloud instance.

**. Change the Domain (Hostname)**

* Go to GitHub Enterprise settings.
* In the **"Enterprise settings"**, navigate to **"Domains"**.
* You will see the option to **add a custom domain**.
* Update the domain settings with your new custom domain (e.g., github.yourcompany.com).
* Verify the domain using DNS records as instructed by GitHub.

**2. Set Up Subdomain Isolation**

* Once the custom domain is configured, you can create **subdomains** (e.g., dev.github.yourcompany.com, staging.github.yourcompany.com).
* Go to your DNS provider and set up **CNAME records** for each subdomain to point to GitHub's servers.
* Ensure each subdomain is properly configured in GitHub Enterprise to separate the environments or teams.

**Lab: Demonstrating SSH Access and Command Line Utilities on GitHub Enterprise Cloud**

This lab is designed to provide a hands-on experience in configuring SSH access and using Git command-line utilities on GitHub Enterprise Cloud (GHEC). By the end of the lab, you will understand how to securely access repositories and perform basic Git operations via the terminal.

**Prerequisites:**

* A **GitHub Enterprise Cloud** account.
* Basic knowledge of Git and command-line operations.
* Git and OpenSSH installed on your local machine.

**Lab Objectives:**

1. **Set up SSH access to GitHub Enterprise Cloud**.
2. **Configure Git with user credentials**.
3. **Clone a repository using SSH**.
4. **Perform basic Git operations**:
   * Add changes.
   * Commit changes.
   * Push changes.
   * Pull updates.
5. **Troubleshoot common SSH access issues**.

**Lab Environment Setup**

**Step 1: Installing Git and OpenSSH (if not already installed)**

**Linux** (Ubuntu/Debian-based):

bash

CopyEdit

sudo apt update

sudo apt install git

**Windows**:

* Install Git from [Git for Windows](https://gitforwindows.org/).
* Install OpenSSH from the [Windows Features](https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse) if not already installed.

**Step 2: Create or Use an Existing GitHub Enterprise Cloud Account**

* Ensure you have a GitHub Enterprise Cloud account with access to a repository. If not, create one or request access to a repository.

**Part 1: Setting Up SSH Access**

**Step 3: Generate an SSH Key Pair**

1. Open your terminal.
2. Run the following command to generate an SSH key pair:

bash

CopyEdit

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

Replace "your\_email@example.com" with the email address associated with your GitHub Enterprise Cloud account.

1. When prompted for the file to save the key, press **Enter** to accept the default location (~/.ssh/id\_rsa).
2. Optionally, set a passphrase for additional security. You can leave it blank for no passphrase.

**Step 5: Add SSH Key to GitHub Enterprise Cloud**

1. Copy the contents of your public key (id\_rsa.pub):

bash

CopyEdit

cat ~/.ssh/id\_rsa.pub

Copy the entire output starting with ssh-rsa.

1. Log in to your GitHub Enterprise Cloud account and navigate to **Settings** > **SSH and GPG keys**.
2. Click **New SSH key**, paste the copied key, and give it a descriptive title (e.g., "Work Laptop").
3. Click **Add SSH key**.

**Part 2: Configuring Git**

**Step 6: Set Up Global Git Configuration**

1. Set up your Git name and email (use the same email as in GitHub):

bash

CopyEdit

git config --global user.name "Your Name"

git config --global user.email "your\_email@example.com"

1. You can verify the configuration by running:

bash

CopyEdit

git config --list

**Part 3: Cloning a Repository Using SSH**

**Step 7: Clone a GitHub Repository**

1. Navigate to your GitHub Enterprise Cloud repository page.
2. Click the **Code** button and select **SSH** to get the SSH URL.
3. In the terminal, run:

bash

CopyEdit

git clone git@github.com:username/repository.git

Replace username/repository.git with the actual repository URL.

1. Change into the cloned directory:

bash

CopyEdit

cd repository

GitHub Pages is a feature provided by GitHub that allows you to host static websites directly from a repository. These sites are usually HTML, CSS, and JavaScript files, and they can be used for things like personal blogs, project documentation, portfolios, and more.

**GitHub Pages in GitHub Enterprise**

When it comes to **GitHub Enterprise**, you can use GitHub Pages in a similar way as you would on GitHub.com. The main difference is that GitHub Enterprise is a private, on-premise solution that is often used by organizations, so the hosting environment is private and customizable based on the organization's needs.

Here’s how GitHub Pages works in the context of **GitHub Enterprise**:

1. **Repository-based Hosting**:
   * You create a repository (public or private) on GitHub Enterprise.
   * In that repository, you can add files for a static website (HTML, CSS, JavaScript, images, etc.).
2. **GitHub Pages Settings**:
   * You can enable GitHub Pages for a repository by going to the **Settings** of the repository.
   * Under the **Pages** section, you can select which branch (usually main or gh-pages) to serve the website from. You can also configure which folder (like /docs) to use.
3. **Custom Domain**:
   * If your organization or project has a custom domain, you can configure the website to be accessible via that domain instead of using the default GitHub Pages domain (username.github.io or orgname.github.io).
4. **Access Control**:
   * Since GitHub Enterprise is for private repositories within an organization, the GitHub Pages site can be public or restricted to specific teams within the organization depending on the repository’s visibility (public or private).
   * For private repositories, GitHub Enterprise can also ensure that the pages are only accessible by authenticated users within the organization, though public-facing websites can be made entirely accessible to anyone.
5. **Security & Compliance**:
   * GitHub Enterprise allows organizations to have better control over security and compliance settings for their GitHub Pages sites, such as custom SSL certificates, access control policies, and ensuring that content is being served according to internal security guidelines.

**Key Points:**

* **GitHub Pages** lets you host static websites directly from your GitHub repositories.
* In **GitHub Enterprise**, this feature works the same way, but it's hosted within your organization’s private infrastructure.
* It’s often used for documentation, blogs, or simple websites.
* You can configure custom domains and control access permissions based on repository settings.