**1. Git Fetch:**

* **What it does:** Downloads the latest commits, files, and updates from the remote repository to your local repository's tracking branches (e.g., origin/main), but **does not merge** these changes into your current working branch.
* **Usage:** It allows you to see what changes have been made remotely without affecting your working directory. This is useful if you want to review changes before integrating them.
* **Command:**

**git fetch**

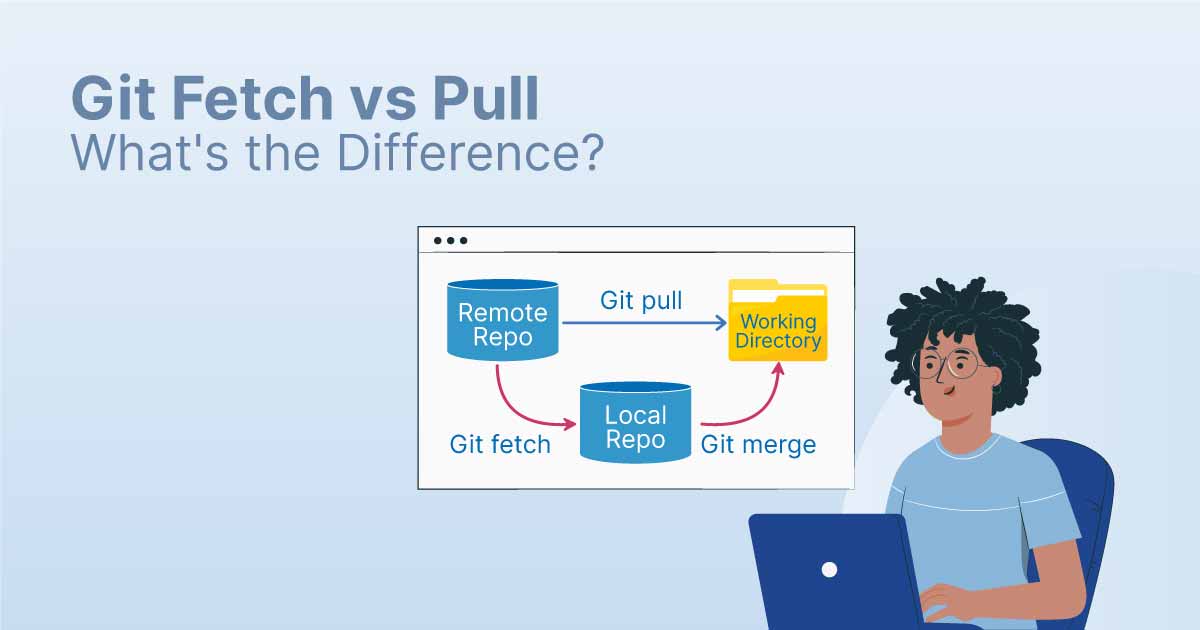
* **Scenario:** You fetch updates from the remote repository but still need to manually merge or rebase the changes into your local branch.

**2. Git Pull:**

* **What it does:** Combines fetch and a subsequent **merge** (or **rebase**) in one step. It downloads the changes from the remote repository and immediately attempts to integrate them into your current branch.
* **Usage:** It automatically updates your working directory with the changes from the remote repository.
* **Command:**

**git pull**

* **Scenario:** You want to get the latest changes from the remote repository and apply them to your current branch immediately.



**Key Difference:**

* fetch only downloads the data but does not affect your working directory or branches.
* pull downloads and automatically merges the changes into your current working branch.

**Recommended Workflow:**

* **Safe Approach:** Use fetch first to review changes and then manually merge them.
* **Quick Approach:** Use pull when you are confident and want to update your branch in one step.

**Git Stash Overview:**

git stash is a feature in Git that allows you to temporarily save (stash) changes that are not ready to be committed. It is particularly useful when you are in the middle of working on something, and suddenly need to switch to a different task or branch, but do not want to commit incomplete changes.

Imagine you are working on a **Dockerfile** and then receive a high-priority production support task. You don’t want to lose your current work, nor do you want to commit incomplete changes. This is where git stash comes in handy.

**Steps for Using Git Stash:**

**1. Stash Your Current Changes:**

To stash your uncommitted changes, use:



This saves your current changes (both tracked and untracked files if configured) into a temporary storage area, freeing up your working directory so you can switch tasks.

**2. View Stashed Changes:**

To see a list of all stashed changes, run:



The output will show **stashes like stash@{0}, stash@{1},** etc. Each stash is identified by an index and typically has a message like **"WIP" (Work In Progress)** to indicate the state when the stash was created.

**3. Apply the Most Recent Stash:**

To reapply the most recent stash (the one on top of the stash stack), use:

This restores the stashed changes to your working directory without removing them from the stash list.

**4. Apply a Specific Stash:**

If you have multiple stashes and want to apply a specific one, specify the stash index:

**5. Apply and Remove the Most Recent Stash:**

To apply the most recent stash and simultaneously remove it from the stash list, use:



This is equivalent to running git stash apply followed by git stash drop.

**6. Remove a Specific Stash:**

If you want to delete a specific stash from the list without applying it, use:

**7. Clear All Stashed Changes:**

If you want to remove all stashed changes from the stack, use:



**Why Use Git Stash?**

* **Interruptions in Work:** When you're working on a feature but need to urgently switch to another task (e.g., fixing a bug in production).
* **Clean Working Directory:** Stash allows you to keep your working directory clean when you are not ready to commit.
* **Context Switching:** It helps in quickly switching between different branches or tasks without having to commit incomplete work.
* **Experimentation:** You can stash changes when trying out something experimental and later decide to discard or apply them.

By using Git stash, you can effectively manage your work, prevent incomplete commits, and ensure a smooth development workflow.

**Git Revert:**

git revert is used to undo changes from a previous commit by creating a new commit that reverses those changes. Unlike git reset, it doesn't modify the commit history or lose any data, making it a safer option for collaborative work.

**Example:**

git revert 98bd832

This creates a new commit that undoes the changes introduced by commit 98bd832. The original commit remains in the history, and a new commit will apply the reverse changes.

**Git Reset:**

git reset is used to move the HEAD pointer to a previous commit, effectively "resetting" your branch to that point. There are three types of resets:

**1. Soft Reset:**

* **Effect:** Moves the HEAD pointer to a specific commit while keeping changes staged.
* **Example:**

**git reset --soft HEAD~1**

**git reset --soft 8dee86f**

This command undoes the last commit, but the changes remain staged, allowing you to recommit or modify them.

**2. Mixed Reset (Default Behavior):**

* **Effect:** Moves the HEAD pointer to a specific commit, unstages the changes, but keeps them in the working directory.
* **Example:**

**git reset HEAD~1**

This unstages changes from the last commit, but you can still modify or recommit them.

**3. Hard Reset:**

* **Effect:** Moves the HEAD pointer to a specific commit and **discards all changes** after that point in both the staging area and the working directory.
* **Example:**

**git reset --hard HEAD~1**

This command deletes all changes made after the last commit. Use it with caution as uncommitted changes will be lost.

**Visual Representation:**

You can refer to the image above, which illustrates how the revert and reset commands work in different scenarios:

* **Git Revert:** Shows the creation of a new commit that undoes changes without affecting history.
* **Git Reset:** Depicts soft, mixed, and hard reset scenarios, showing how they move the HEAD and what happens to the changes in the working directory.

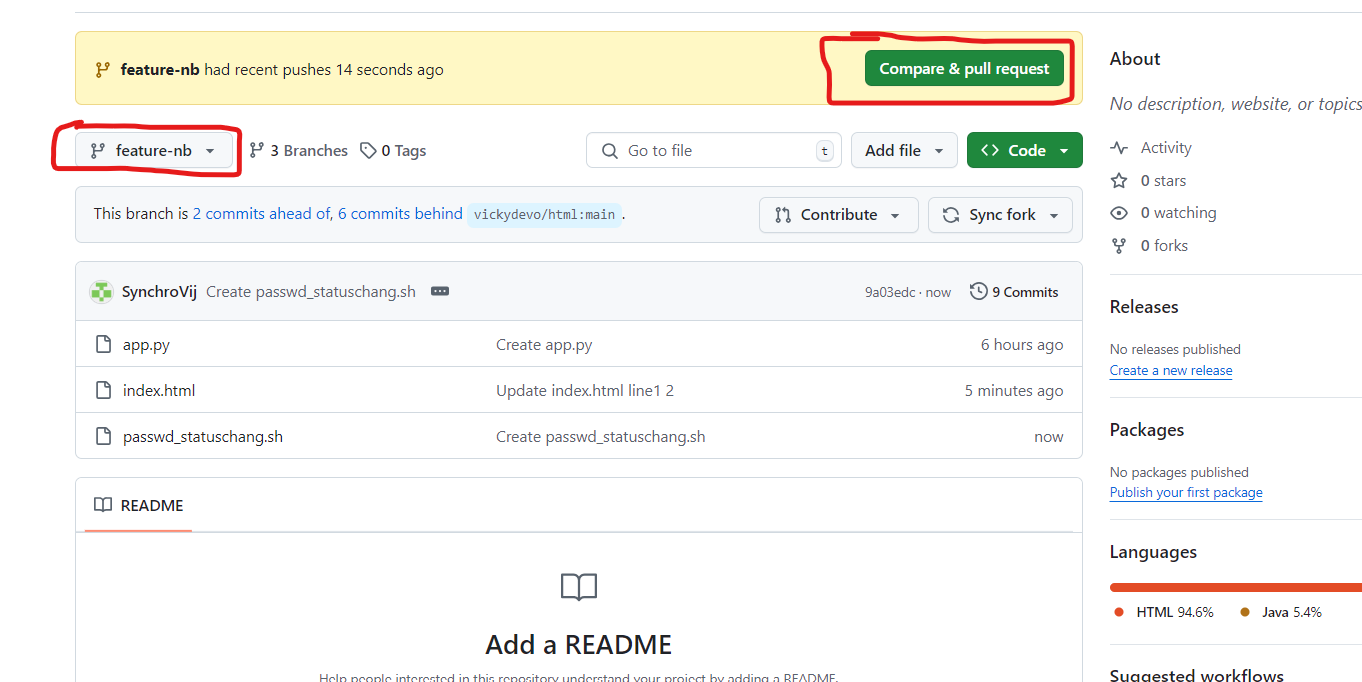
**Creating and Merging Pull Requests on GitHub**

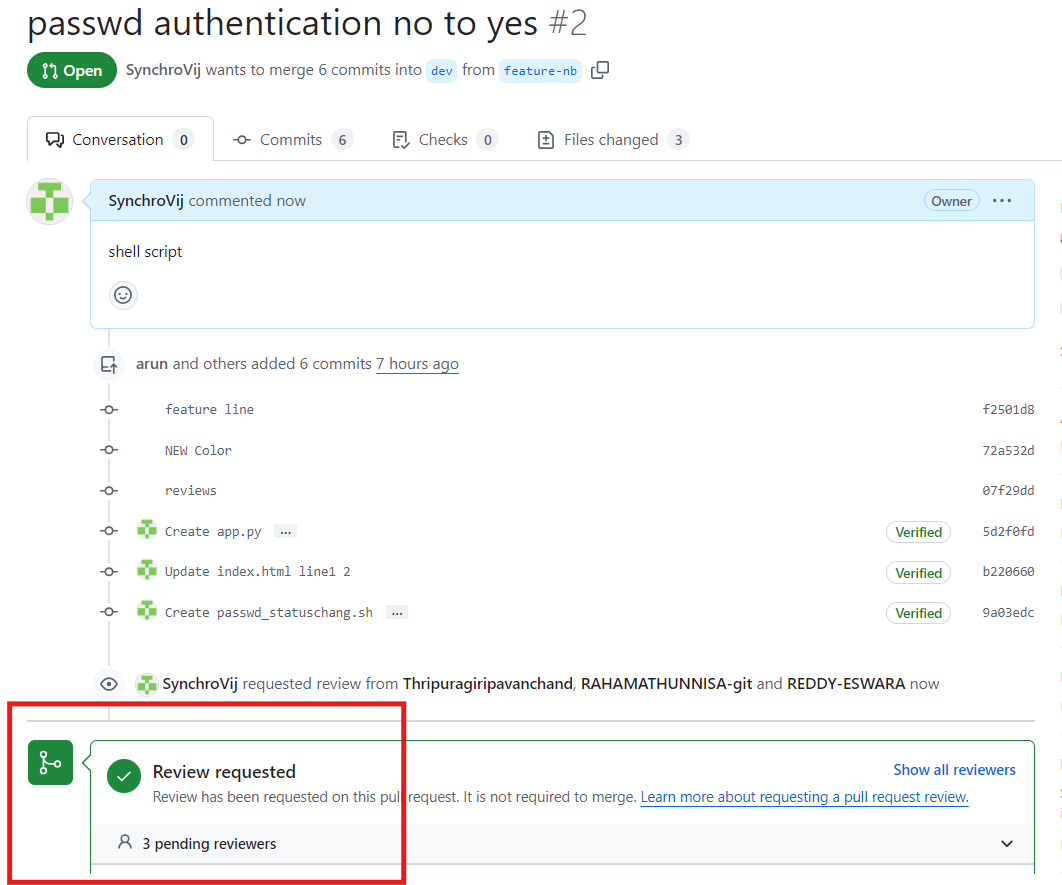
**Understanding Pull Requests**

A pull request (PR) is a feature on GitHub that allows you to propose changes to a repository. It's a way to suggest new code, fix bugs, or make improvements to the existing codebase.

**Steps to Create a Pull Request**

1. **Fork the Repository:**
   * Go to the repository you want to contribute to on GitHub.
   * Click the "Fork" button in the top right corner. This creates a copy of the repository under your account.
2. **Create a New Branch:**
   * Navigate to your forked repository.
   * Click the "Code" button and select "Branch".
   * Enter a descriptive name for your new branch (e.g., "feature-new-feature").
   * Click "Create branch".
3. **Make Your Changes:**
   * Clone your forked repository to your local machine.
   * Make the necessary changes to the code.
   * Commit your changes using git commit -m "Your commit message".
4. **Push Your Changes to Your Fork:**
   * Push your changes to your remote fork: git push origin your-branch-name.
5. **Create a Pull Request:**
   * Go back to your forked repository on GitHub.
   * You should see a notification about the new branch you pushed.
   * Click the "Compare & pull request" button.





* + Review the changes and add a descriptive title and description for your pull request.
  + Click "Create pull request".

**Merging a Pull Request**

Once you've created a pull request, the maintainer of the original repository will review your changes. If they approve, they can merge the pull request into the main branch.

1. **Review and Approve:**
   * The maintainer will review your code, provide feedback, and potentially request changes.
   * Once the changes are satisfactory, the maintainer will approve the pull request.
2. **Merge:**
   * The maintainer can choose to merge the pull request directly or use a merge strategy like "Squash and Merge" or "Rebase and Merge".

**Additional Tips**

* Use clear and concise commit messages to describe your changes.
* Follow the project's coding style and conventions.
* If you encounter any issues, don't hesitate to ask for help from the community or the repository's maintainers.

**Screenshots**

[Insert screenshots of the GitHub interface showing the steps to create a pull request and merge it]

**Command-Line Usage**

If you prefer to work from the command line, here are the basic commands you'll use:

