DevOps Shack Git Assignment | Task:4

Task 4: Stashing and Switching Contexts

4.1 Introduction to Git Stash

Git stash is like a **temporary shelf** where you can **save your unfinished changes** without committing them. It allows you to **pause your current work**, switch to another branch (or task), and **come back later** without losing any progress.

Think of it as **putting your work aside safely** to deal with **urgent interruptions**.

4.2 Why Git Stash is Critical in Real-World Projects

Scenario:

Imagine you're working on a **long-term feature** on the feature-backend branch, halfway through some changes, but:

- A **critical production bug** is reported.
- You need to **switch immediately** to the main branch to patch and deploy the fix.
- Your **current work is incomplete** and **shouldn't be committed yet**.

If you try to switch branches without committing, Git **blocks you**:

error: Your local changes to the following files would be overwritten by checkout

Git stash lets you **save your work temporarily**, **switch branches**, handle the urgent task, then **retrieve your work** later.

Real-World Use Cases:

- **Context switching**: Jumping between tasks without losing work.
- **Avoiding messy commits**: You don’t want to commit **half-done work** just to switch branches.
- **Multitasking across features**: Developers handle **multiple branches** (bugfixes, features, releases).

4.3 Deep Dive: How Git Stash Works Internally

When you run git stash:

1. Git **captures the current state** of:
    - **Tracked files** with changes.
    - **Staged files**.
2. It **saves this state** as a **stash object**:
    - This object contains:
        - A **commit snapshot** of the changes.
        - Metadata (branch name, timestamp, etc.).
3. The **working directory** and **staging area** are **cleaned**:
    - You can now **switch branches freely**.
4. Later, when you **apply or pop the stash**:
    - Git **restores the changes** from the stash object back to your working directory.

How Stashes Are Stored:

- Git keeps **multiple stashes** in a stack-like structure:
    - stash@{0}: Most recent stash.
    - stash@{1}, stash@{2}, etc.
- Stashes are stored in:
    - .git/refs/stash (for latest stash).
    - .git/logs/refs/stash (for stash history).

4.4 Step-by-Step Implementation with Ultra-Deep Explanation

Scenario Setup:

You're working on feature-backend, making **backend changes**. Suddenly, you're asked to **switch to main** to fix an **urgent bug**.

Step 1: Start Making Changes (Uncommitted Work)

- On feature-backend:
    - Edit backend.txt with **new logic**.
    - Don’t stage or commit yet.

This represents **uncommitted work** that **cannot be switched away from** without risking loss.

Step 2: Attempt to Switch Branch (Blocked)

If you try to switch to main:

git checkout main

Git stops you:

error: Your local changes to the following files would be overwritten by checkout:

backend.txt

Step 3: Save Your Work with Git Stash

git stash

- Git:
    - **Captures the diff** of the uncommitted changes.
    - **Cleans your working directory**.
- **Default stash message**:
    - "WIP on feature-backend: <commit-hash> <commit-message>"

Step 4: Verify Stashed Changes

git stash list

- Example output:

stash@{0}: WIP on feature-backend: 3a4f5cd Added backend logic

---

Step 5: Switch to Another Branch

git checkout main

- **Now allowed** because your **working directory is clean**.
- Apply **hotfix** or perform **other tasks**.

Step 6: Retrieve Stashed Changes

Option 1: Apply (Keep Stash for Later)

git stash apply

- **Reapplies the stashed changes** but **keeps the stash in the list**.

Option 2: Pop (Apply and Remove Stash)

git stash pop

- **Reapplies** and **removes** the stash from the list.

Step 7: Verify Stash is Gone (if Popped)

git stash list

- Stash is **removed** if popped, remains if **applied**.

4.5 Visualizing the Stash Workflow

Working Directory (with changes)
|
git stash
↓
Working Directory (clean)
|
Switch Branches
|
git stash pop/apply
↓
Working Directory (restored)

4.6 Advanced Stash Techniques

Stashing Only Staged Files:

git stash --keep-index

- **Stashes only staged files**, leaves working directory changes intact.

Stashing Untracked or Ignored Files:

git stash -u # Include untracked files

git stash -a # Include all (untracked + ignored)

Why?
By default, **untracked/ignored files** are **not stashed**.

Naming Your Stashes (Descriptive Messages):

git stash save "Refactor backend logic"

Or:

git stash push -m "Backend refactor changes"

- Helps **identify** stashes later.

Viewing Stash Contents:

git stash show stash@{0}

- Shows a **summary of changes**.

git stash show -p stash@{0}

- Shows **full diff**.

Applying Specific Stashes:

git stash apply stash@{1}

- Applies a **specific stash** without disturbing others.

Deleting Stashes:

git stash drop stash@{0}

- Deletes a **specific stash**.

git stash clear

- Deletes **all stashes**.

4.7 Real-World Best Practices for Stashing

1. **Always label stashes** with meaningful messages.
2. **Pop stashes** as soon as possible to avoid **forgetting them**.

3. Use **apply** instead of **pop** when unsure—keeps stash safe.
4. Be cautious when **stashing large changes**—reapply carefully.
5. In **CI/CD pipelines**, avoid leaving **stashes untracked**—could lead to **unexpected behavior**.

4.8 Common Mistakes & Pitfalls

Mistake	**How to Avoid**
Forgetting about stashed changes	**Label stashes** clearly and review them regularly.
Overwriting work during stash application	Use git stash show and **review changes first**.
Assuming untracked files are stashed	Use git stash -u or git stash -a to include them.
Accidentally clearing stashes	Avoid using git stash clear unless absolutely sure.

4.9 Stashing vs Committing Unfinished Work

Aspect	**Stash**	**Commit**
Purpose	Temporarily save changes without committing	Permanently save changes in history
Visibility	Hidden from history (in stash list only)	Visible in **commit log**
Risk	Forgetting stashed work	Cluttering commit history with WIP commits
Use Case	**Temporary context switching**	**Work preservation, even if incomplete**