DevOps Shack Git Assignment | Task:5

Task 5: Cherry-Picking Commits Across Branches

5.1 Introduction to Cherry-Picking in Git

Cherry-picking in Git is a **precise and powerful tool** that allows you to **select a specific commit** (or multiple commits) from one branch and **apply them to another branch**, without merging the entire branch history.

5.2 Why Cherry-Picking Matters in Real-World Projects

Imagine you're in a corporate DevOps environment, and:

- A **bug fix** was made in the feature-backend branch.
- This bug fix is urgently needed in main, but **the rest of the backend work isn't ready** for production.

In this case, you don't want to merge **all of feature-backend**. You just want that **one bug fix commit**.

Cherry-pick allows you to:

- **Selectively move that commit** from feature-backend to main.
- Keep **main stable** without introducing **incomplete or unrelated changes**.

Real-World Scenarios for Cherry-Picking:

1. **Hotfixes**:
    - A critical fix on a **feature branch** needs to be applied to **production** (main).
2. **Porting Changes Across Versions**:
    - Apply a fix made in release-v1.0 to release-v2.0 **without merging the entire branch**.
3. **Selective Integration**:
    - Apply **only certain commits** from a **long feature branch** into another.

Analogy:
Think of **cherry-picking** like going to a **buffet** and choosing **only the dishes you want**, without taking the entire spread.

5.3 How Cherry-Picking Works Internally

When you **cherry-pick a commit**, Git:

1. **Extracts the changes** from that specific commit.
2. **Applies those changes** as a **new commit** on the target branch.
3. The **commit hash changes** (because commits are tied to:
    - **Parent commits**.
    - **Timestamp**.
    - **Commit content**.

This is **not a merge** or **rebase**. It is **copying changes**, not history.

Visualizing Cherry-Picking:

Before Cherry-Picking:

main: A---B---C

feature: A---B---C---D---E

You want to cherry-pick **commit D** into main.

After Cherry-Picking:

main: A---B---C---D'

feature: A---B---C---D---E

- D' is a **new commit** on main (same changes as D, but **different hash**).
- **No merging** or **rebase happens**—only **D's changes** are applied.

5.4 When Not to Cherry-Pick

- **Large feature sets**:
    - Avoid cherry-picking **complex, dependent commits**—use **merge or rebase**.

- **Multiple related commits**:
    - If commits **depend on each other**, cherry-picking may cause **conflicts**.
- **Collaborative branches**:
    - Communicate with your team when cherry-picking **shared commits**.

5.5 Step-by-Step Implementation of Cherry-Picking

Scenario Setup:

- You're working on:
    - **feature-backend** (with multiple commits).
    - **main** (stable production).
- A **bug fix commit** (Fix API endpoint issue) exists in **feature-backend** and needs to be **applied to main**.

Step 1: Identify the Commit to Cherry-Pick

Switch to **feature-backend** and **list commit history**:

git log --oneline

Example output:

e9d4c2b (HEAD -> feature-backend) Fix API endpoint issue
c8b1f9a Add backend validation logic
b6d7a3e Setup backend scaffolding
The **commit hash e9d4c2b** (or part of it) identifies the **bug fix commit**.

Step 2: Switch to the Target Branch (main)

git checkout main

git pull origin main # Ensure up-to-date

Step 3: Apply the Commit (Cherry-Pick)

git cherry-pick e9d4c2b

- Git:
    - **Extracts changes** from e9d4c2b.

- **Applies them** as a **new commit** in main.

Step 4: Resolve Conflicts (if any)

If **conflicts occur** during cherry-picking:

- Git **pauses** the cherry-pick process.
- You manually **edit the conflicted files**.

After resolving:

git add <resolved-files>

git cherry-pick --continue

Step 5: Verify Commit History

git log --oneline

You'll see **D'** (the cherry-picked commit):

f3e1b7d (HEAD -> main) Fix API endpoint issue (cherry-picked from e9d4c2b)

Step 6: Push Changes

git push origin main

5.6 Advanced Cherry-Picking Techniques

Cherry-Pick Multiple Commits:

git cherry-pick <commit1> <commit2> <commit3>

- Useful for **picking several unrelated commits**.

Cherry-Pick a Range of Commits:

git cherry-pick <start-commit>^..<end-commit>

- Picks **multiple consecutive commits**.

Abort a Cherry-Pick:

If things go wrong:

git cherry-pick --abort

- Restores **working directory** to the state **before cherry-pick**.

Skip a Commit During Cherry-Pick:

git cherry-pick --skip

- Skips the **current commit** during a **conflict resolution** process.

5.7 Visualizing Cherry-Pick Workflows

Multiple Cherry-Picked Commits:

main: A---B---C---D'---E'

feature: A---B---C---D---E

- **D'** and **E'** are cherry-picked versions of **D** and **E**.

5.8 Cherry-Picking vs Other Git Operations

Operation	**Purpose**	**History Preserved**	**Use Case**
Merge	Combine branches, preserve both histories	Yes	Integrating full branches
Rebase	Reapply commits from one branch onto another	No (rewrites)	Keeping history linear for feature branches
Cherry-Pick	Apply **specific commits** to another branch	No (copy commits)	Selective commit application

---

5.9 Best Practices for Cherry-Picking

1. **Use only for isolated commits**:
    - Ideal for **hotfixes**, **simple patches**.
2. **Avoid cherry-picking dependent commits**:

    - Risk of **missing dependencies** or **introducing conflicts**.
3. **Always communicate cherry-picks**:
    - In team environments, notify about **cherry-picked changes**.
4. **Document cherry-picks in commit messages**:
    - Helps track **where the commit came from**.

5.10 Cherry-Picking in CI/CD Pipelines

- **Cherry-picking is manual**:
    - Typically not used **inside pipelines**.
    - Use **merge** or **rebase** in **automated CI/CD**.
- **When applicable**:
    - Apply **hotfixes selectively** before triggering a **release pipeline**.

5.11 Common Mistakes & Pitfalls

Mistake	**How to Avoid**
Cherry-picking **complex dependent commits**	Use **merge or rebase** instead.
Forgetting **conflict resolutions**	Use git status and git cherry-pick --continue.
Losing context of where commits came from	Use **commit messages** like (cherry-picked from <hash>).
