Day 11 Task: Advance Git & GitHub for DevOps Engineers: Part-2

Git Stash:

Git stash temporarily shelves (or *stashes*) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on. Stashing is handy if you need to quickly switch context and work on something else, but your mid-way through a code change and aren't quite ready to commit.

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy. This is useful when you need to switch to a different branch to work on something else, but you don't want to commit the changes you've made in your current branch yet.

You can also use git stash drop to delete a stash and git stash clear to delete all the stashes.

How to use git stash

Here's the sequence to follow when using git stash:

1. Save changes to branch A.
2. Run git stash.
3. Check out branch B.
4. Fix the bug in branch B.
5. Commit and (optionally) push to remote.
6. Check out branch A
7. Run git stash pop to get your stashed changes back.

Cherry-pick:

Git cherry-pick command allows you to integrate selected, individual commits from any branch into your current HEAD branch. This is in contrast with other ways such as merge and rebases which normally apply many commits into another branch.

Cherry-picking is just like rebasing, an advanced concept and also a powerful command. It is mainly used if you don’t want to merge the whole branch and you want some of the commits.

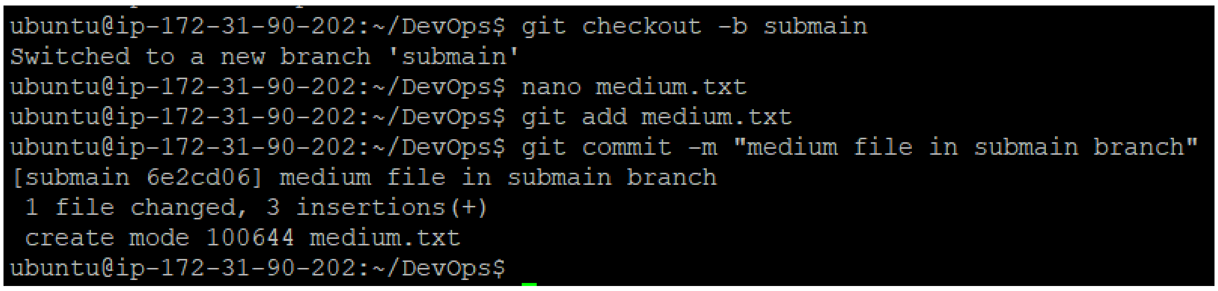
To use git cherry-pick, you first create two new branches and make some commits to them. Then you use git cherry-pick <commit_hash> command to select the specific commits from one branch and apply them to the other.

Resolving Conflicts:

Conflicts can occur when you merge or rebase branches that have diverged, and you need to manually resolve the conflicts before git can proceed with the merge/rebase. git status command shows the files that have conflicts, git diff command shows the difference between the conflicting versions and git add command is used to add the resolved files.

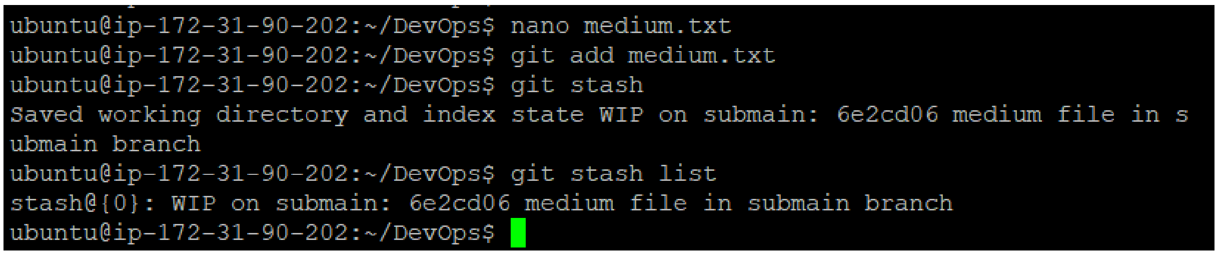
Task-01

- Create a new branch and make some changes to it.



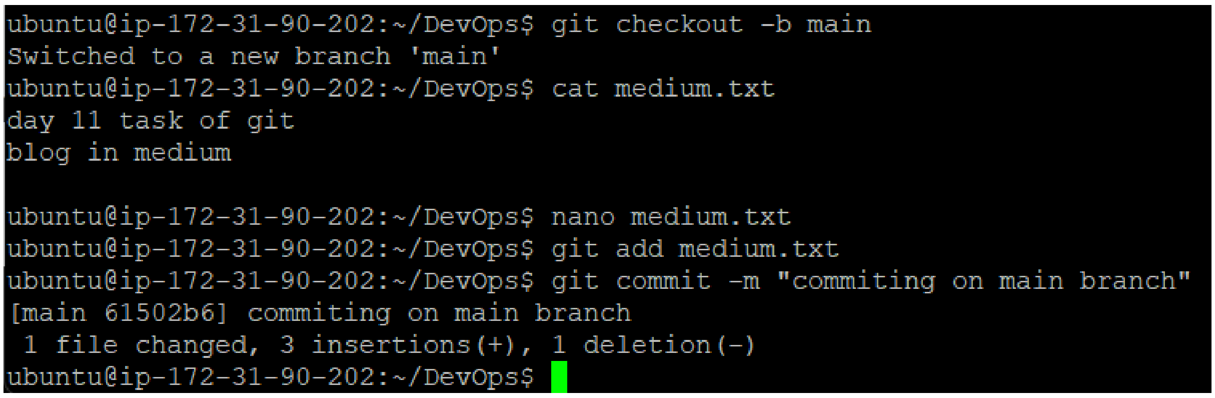
```
ubuntu@ip-172-31-90-202:~/DevOps$ git checkout -b submain
Switched to a new branch 'submain'
ubuntu@ip-172-31-90-202:~/DevOps$ nano medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "medium file in submain branch"
[submain 6e2cd06] medium file in submain branch
 1 file changed, 3 insertions(+)
 create mode 100644 medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$
```

- Use git stash to save the changes without committing them.



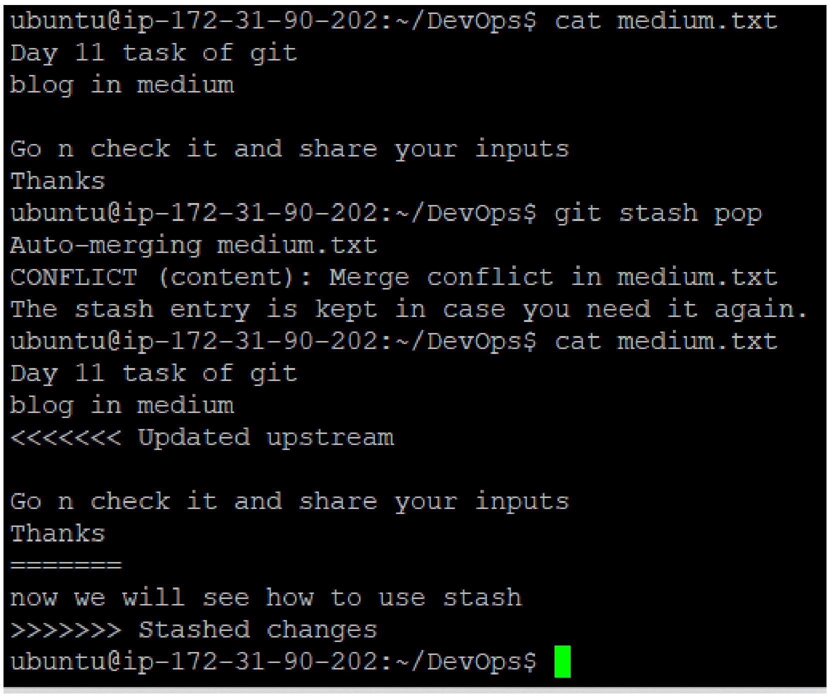
```
ubuntu@ip-172-31-90-202:~/DevOps$ nano medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git stash
Saved working directory and index state WIP on submain: 6e2cd06 medium file in s
ubmain branch
ubuntu@ip-172-31-90-202:~/DevOps$ git stash list
stash@{0}: WIP on submain: 6e2cd06 medium file in submain branch
ubuntu@ip-172-31-90-202:~/DevOps$
```

- Switch to a different branch, make some changes and commit them.



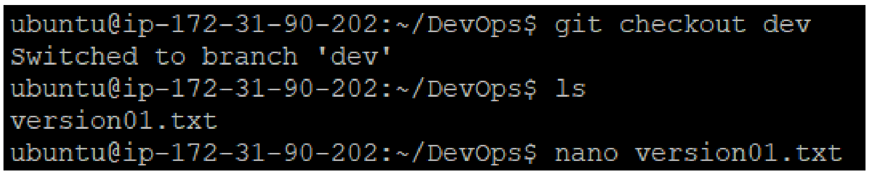
```
ubuntu@ip-172-31-90-202:~/DevOps$ git checkout -b main
Switched to a new branch 'main'
ubuntu@ip-172-31-90-202:~/DevOps$ cat medium.txt
day 11 task of git
blog in medium

ubuntu@ip-172-31-90-202:~/DevOps$ nano medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add medium.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "commiting on main branch"
[main 61502b6] commiting on main branch
 1 file changed, 3 insertions(+), 1 deletion(-)
ubuntu@ip-172-31-90-202:~/DevOps$
```

- Use git stash pop to bring the changes back and apply them on top of the new commits.


```
ubuntu@ip-172-31-90-202:~/DevOps$ cat medium.txt
Day 11 task of git
blog in medium

Go n check it and share your inputs
Thanks
ubuntu@ip-172-31-90-202:~/DevOps$ git stash pop
Auto-merging medium.txt
CONFLICT (content): Merge conflict in medium.txt
The stash entry is kept in case you need it again.
ubuntu@ip-172-31-90-202:~/DevOps$ cat medium.txt
Day 11 task of git
blog in medium
<<<<<<< Updated upstream

Go n check it and share your inputs
Thanks
=======
now we will see how to use stash
>>>>>>> Stashed changes
ubuntu@ip-172-31-90-202:~/DevOps$
```

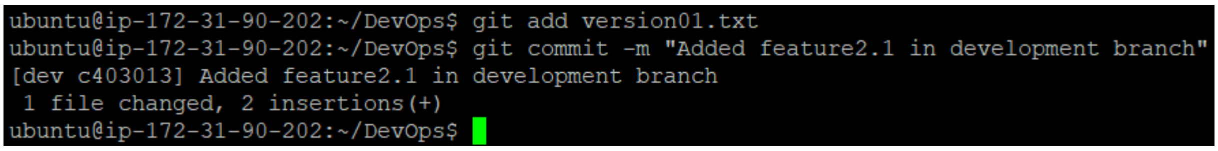
Task-02

- In version01.txt of development branch add below lines after “This is the bug fix in development branch” that you added in Day10 and reverted to this commit.


```
ubuntu@ip-172-31-90-202:~/DevOps$ git checkout dev
Switched to branch 'dev'
ubuntu@ip-172-31-90-202:~/DevOps$ ls
version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
```

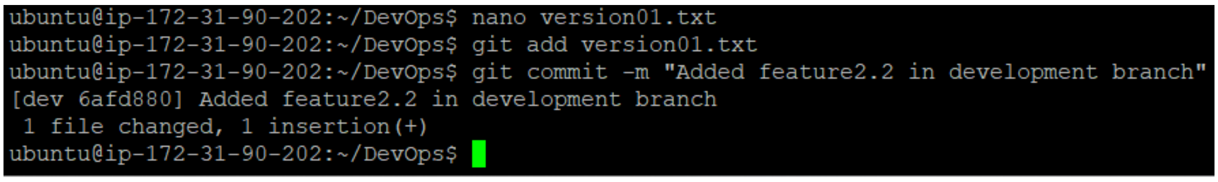
- Line2>> After bug fixing, this is the new feature with minor alteration”

  Commit this with message “ Added feature2.1 in development branch”


```
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature2.1 in development branch"
[dev c403013] Added feature2.1 in development branch
 1 file changed, 2 insertions(+)
ubuntu@ip-172-31-90-202:~/DevOps$
```

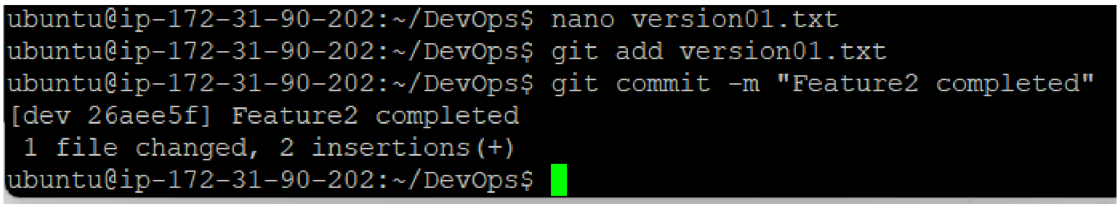
- Line3>> This is the advancement of previous feature

  Commit this with message “ Added feature2.2 in development branch”


```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature2.2 in development branch"
[dev 6afd880] Added feature2.2 in development branch
 1 file changed, 1 insertion(+)
ubuntu@ip-172-31-90-202:~/DevOps$
```

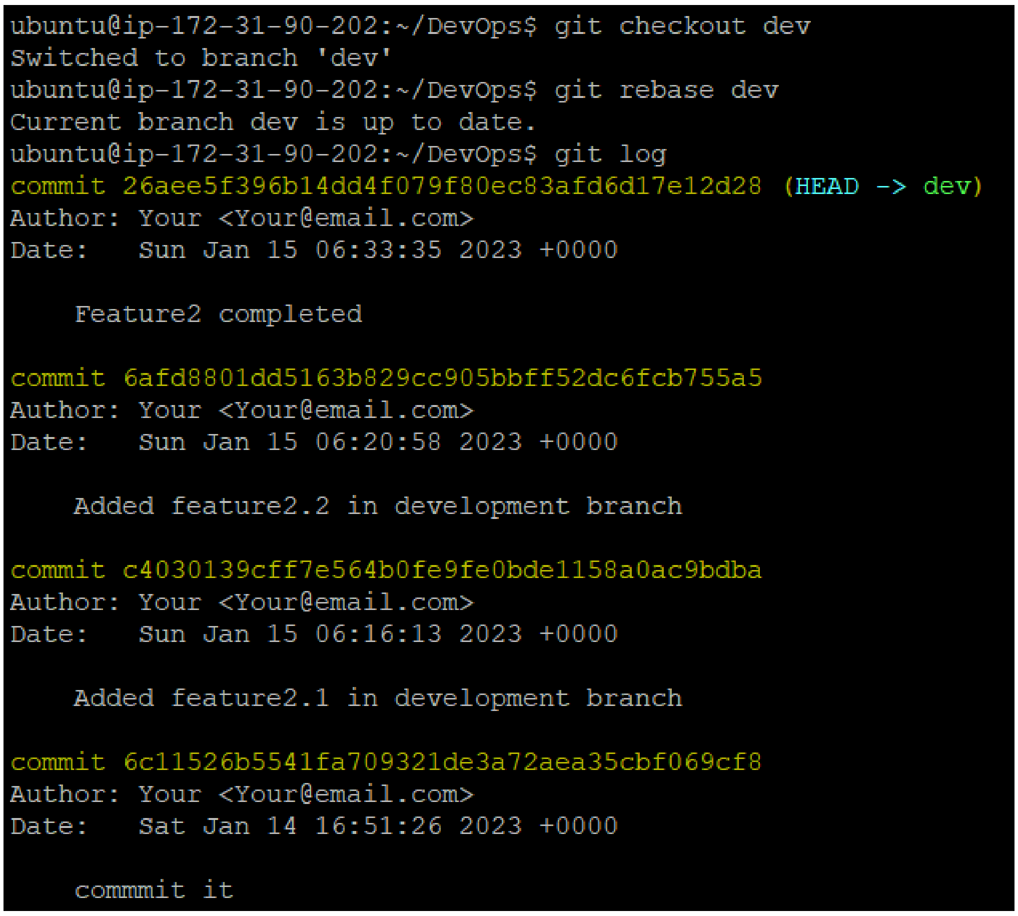
- Line4>> Feature 2 is completed and ready for release

  Commit this with message “ Feature2 completed”



```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Feature2 completed"
[dev 26aee5f] Feature2 completed
 1 file changed, 2 insertions(+)
ubuntu@ip-172-31-90-202:~/DevOps$
```

- All these commits messages should be reflected in Production branch too which will come out from Master branch (Hint: try rebase).

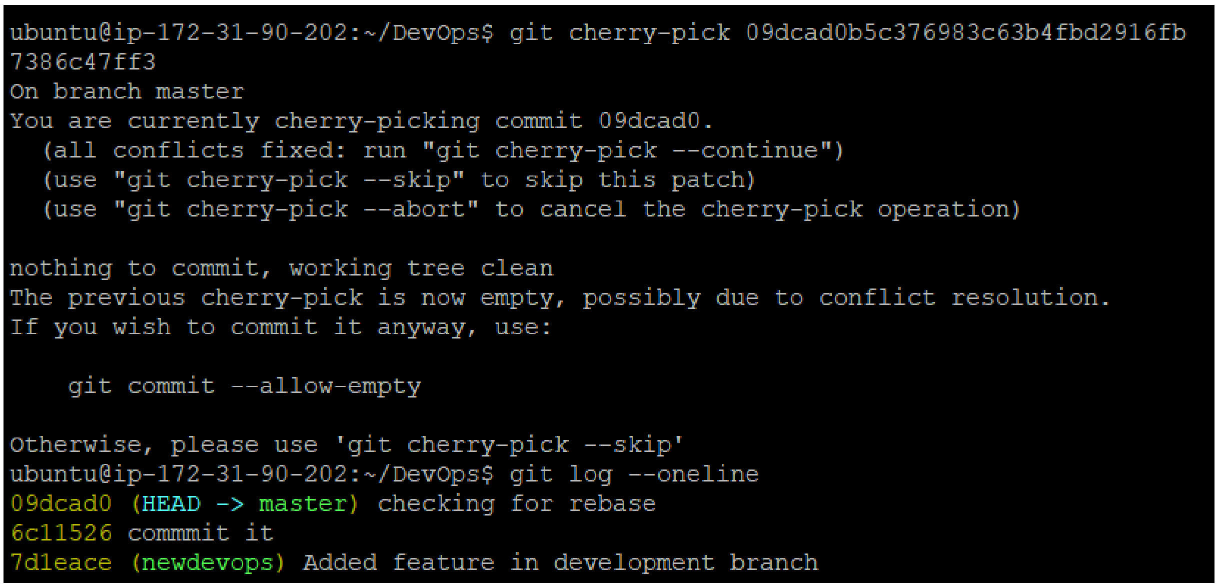


```
ubuntu@ip-172-31-90-202:~/DevOps$ git checkout dev
Switched to branch 'dev'
ubuntu@ip-172-31-90-202:~/DevOps$ git rebase dev
Current branch dev is up to date.
ubuntu@ip-172-31-90-202:~/DevOps$ git log
commit 26aee5f396b14dd4f079f80ec83afd6d17e12d28 (HEAD -> dev)
Author: Your <Your@email.com>
Date:   Sun Jan 15 06:33:35 2023 +0000

    Feature2 completed

commit 6afd8801dd5163b829cc905bbff52dc6fcb755a5
Author: Your <Your@email.com>
Date:   Sun Jan 15 06:20:58 2023 +0000

    Added feature2.2 in development branch

commit c4030139cff7e564b0fe9fe0bde1158a0ac9bdba
Author: Your <Your@email.com>
Date:   Sun Jan 15 06:16:13 2023 +0000

    Added feature2.1 in development branch

commit 6c11526b5541fa709321de3a72aea35cbf069cf8
Author: Your <Your@email.com>
Date:   Sat Jan 14 16:51:26 2023 +0000

    commmit it
```

Task-03

- In Production branch Cherry pick Commit “Added feature2.2 in development branch” and added below lines in it:
- Line to be added after Line3>> This is the advancement of previous feature
- Line4>>Added few more changes to make it more optimized.
- Commit: Optimized the feature

  Here, I have checked out in dev branch and added a new commit “Optimized the feature”



```
ubuntu@ip-172-31-90-202:~/DevOps$ git cherry-pick 09dcad0b5c376983c63b4fbd2916fb
7386c47ff3
On branch master
You are currently cherry-picking commit 09dcad0.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

nothing to commit, working tree clean
The previous cherry-pick is now empty, possibly due to conflict resolution.
If you wish to commit it anyway, use:

    git commit --allow-empty

Otherwise, please use 'git cherry-pick --skip'
ubuntu@ip-172-31-90-202:~/DevOps$ git log --oneline
09dcad0 (HEAD -> master) checking for rebase
6c11526 commmit it
7d1eace (newdevops) Added feature in development branch
```