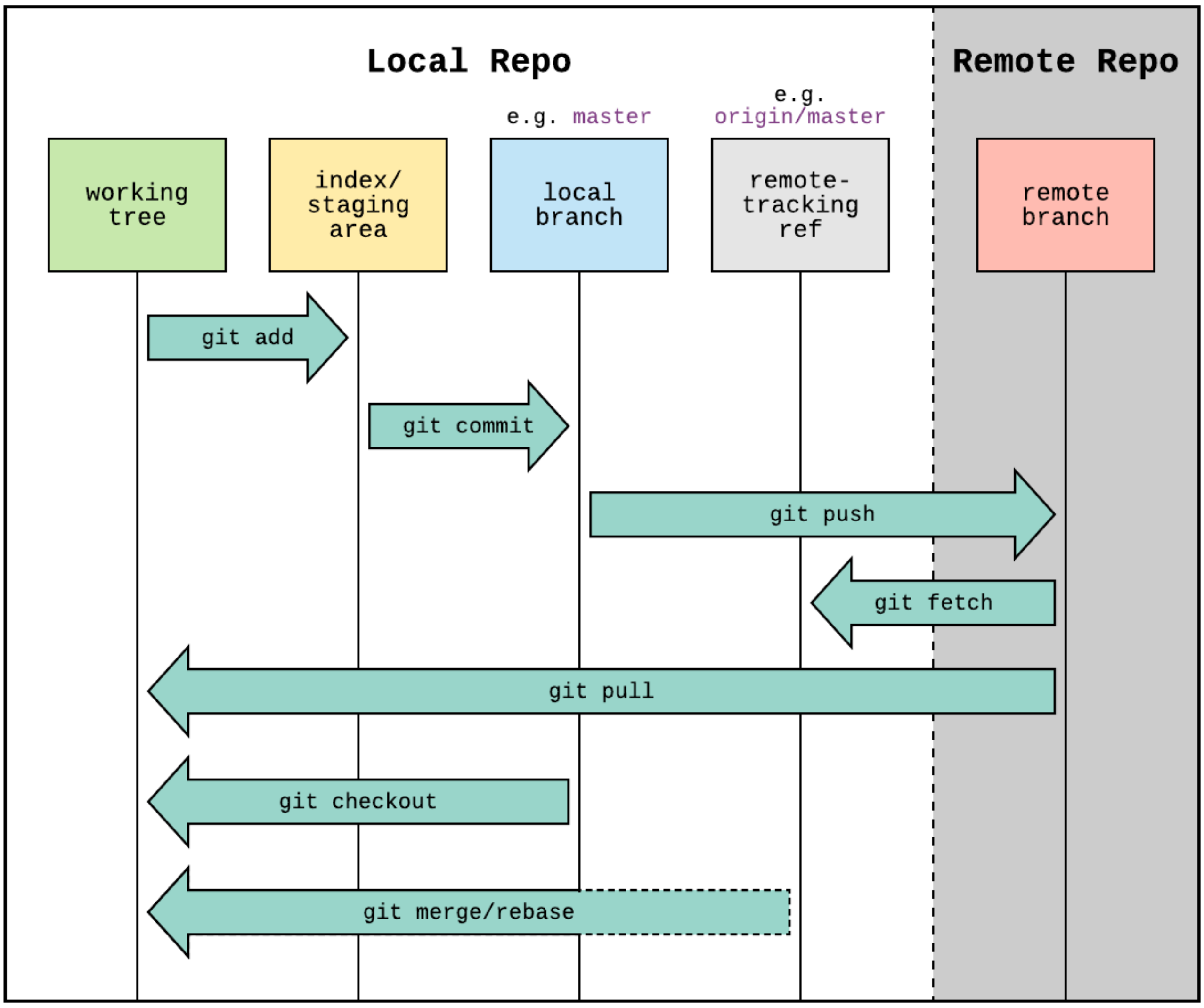
#90DaysOfDevOps Challenge - Day 11 - Advanced Git & GitHub for DevOps Engineers Part 2

Welcome to Day 11 of the #90DaysOfDevOps challenge. In today's continuation of our exploration of Git, we will dive deeper into advanced techniques such as stashing, cherry-picking and conflict resolution. These techniques play a vital role in ensuring efficient collaboration and effective version control in software development.



Git Stash: Saving Changes for Later

`Git stash` is a handy command that allows you to temporarily save changes you've made in your working directory without committing them. This comes in handy when you need to switch to a different branch to work on a separate task but want to keep your current changes.

To use `Git stash`, create a new branch and make some changes. Then, by running `git stash`, you can save those changes. This action removes the changes from your working directory and stores them in a new stash. You can view your stashed changes using `git stash list` and delete specific stashes with `git stash drop`. If you want to clear all stashes, use `git stash clear`.

Cherry-Pick: Selective Commits

`Git cherry-pick` is a powerful command that allows you to choose specific commits from one branch and apply them to another. This technique is useful when you only want to incorporate specific changes from one branch into another, without merging the entire branch.

To cherry-pick commits, create two branches and make commits to each of them. Then, using `git cherry-pick <commit>`, you can select the specific commits you want to apply from one branch to the other. This provides fine-grained control over which changes are included in your target branch.

Resolving Conflicts: Bridging the Gap

Conflicts can arise when merging or rebasing branches that have diverged. Resolving conflicts is a crucial skill in Git, and Git provides helpful commands to streamline the process.

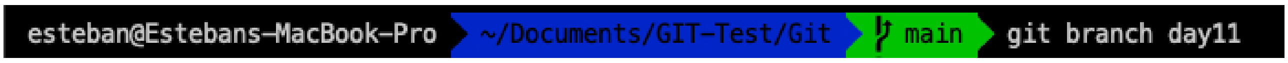
When conflicts occur, use `git status` to identify the files with conflicts. By running `git diff`, you can examine the differences between the conflicting versions, helping you understand the conflicting lines. To resolve conflicts, manually edit the conflicted files, keeping the desired changes and removing the conflict markers. After resolving conflicts, use `git add` to stage the resolved files. This informs Git that the conflicts have been resolved and allows the merge or rebase process to proceed smoothly.

Task 1: Stashing and Applying Changes

In this task, we will work with branches and use the `git stash` command to save changes without committing them. Follow the steps below:

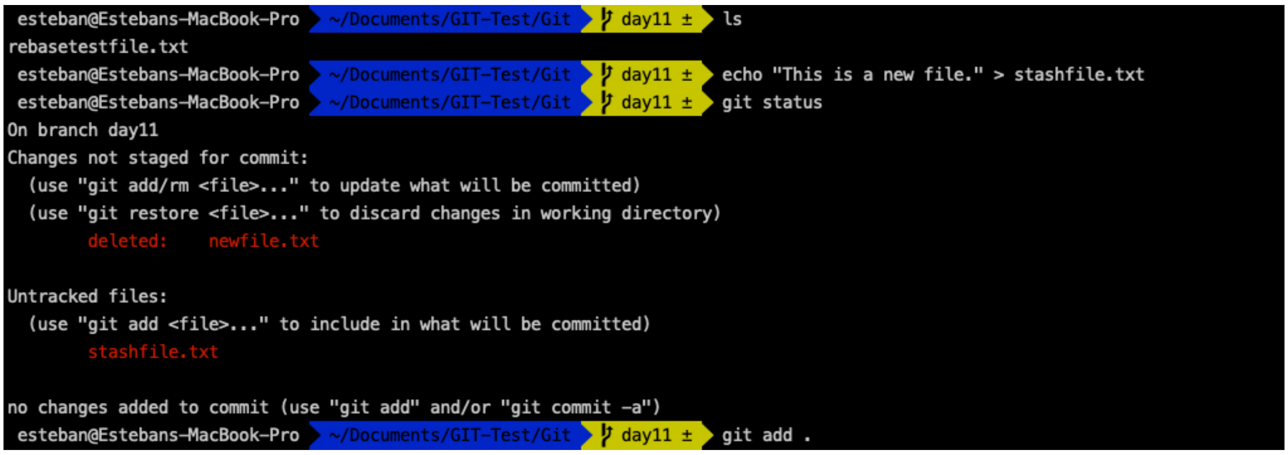
1. Create a new branch using the command:

   ```
   git checkout -b day11
   ```



```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  git branch day11
```

2. Make some changes to the files in this branch.



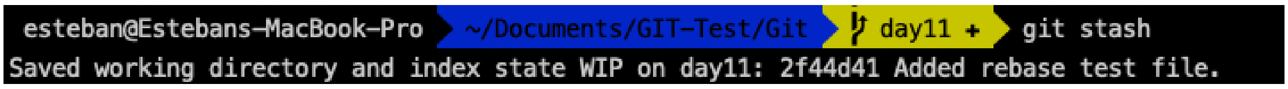
```
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11 ±  ls
rebasetestfile.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11 ±  echo "This is a new file." > stashfile.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11 ±  git status
On branch day11
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    newfile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        stashfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11 ±  git add .
```

3. Use `git stash` to save the changes without committing them:

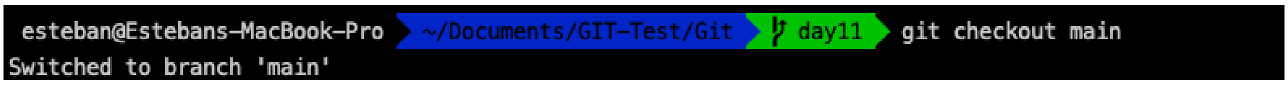
```
git stash
```



```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11 +  git stash
Saved working directory and index state WIP on day11: 2f44d41 Added rebase test file.
```

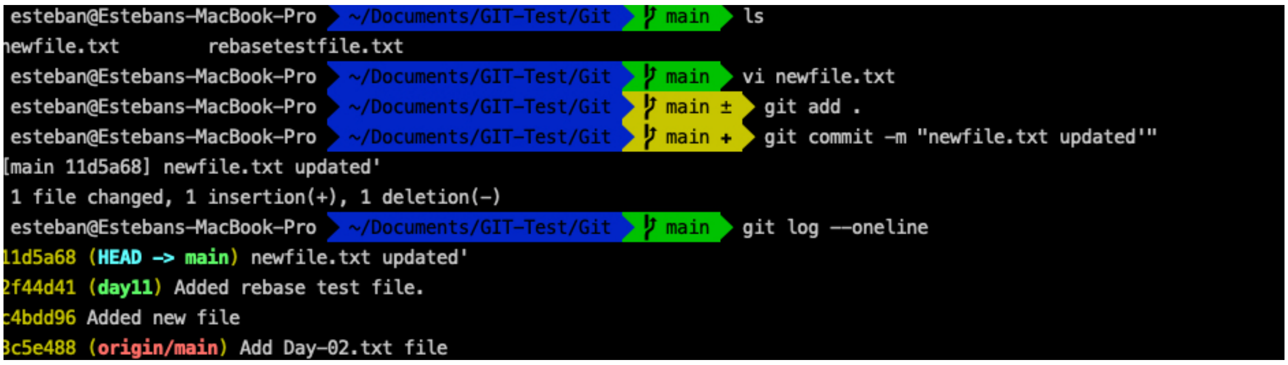
4. Switch to the `main` branch using the command:

```
git checkout main
```



```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  day11  git checkout main
Switched to branch 'main'
```

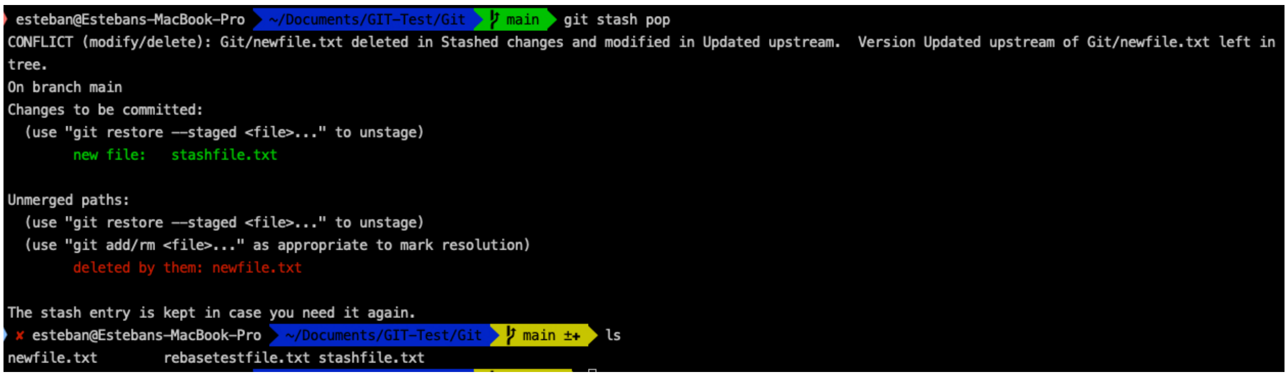
5. Make some changes to the files in the new branch and commit them.



```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  ls
ewfile.txt         rebasetestfile.txt
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  vi newfile.txt
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main ±  git add .
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main +  git commit -m "newfile.txt updated'"
main 11d5a68] newfile.txt updated'
 1 file changed, 1 insertion(+), 1 deletion(-)
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  git log --oneline
1d5a68 (HEAD -> main) newfile.txt updated'
f44d41 (day11) Added rebase test file.
4bdd96 Added new file
c5e488 (origin/main) Add Day-02.txt file
```

6. Use `git stash pop` to bring back the changes you stashed and apply them on top of the new commits:

```
git stash pop
```



```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  git stash pop
CONFLICT (modify/delete): Git/newfile.txt deleted in Stashed changes and modified in Updated upstream.  Version Updated upstream of Git/newfile.txt left in
tree.
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   stashfile.txt

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add/rm <file>..." as appropriate to mark resolution)
        deleted by them: newfile.txt

The stash entry is kept in case you need it again.
 ✘ esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main ±+  ls
newfile.txt        rebasetestfile.txt stashfile.txt
```

Task 2: Rebase and Commit Messages

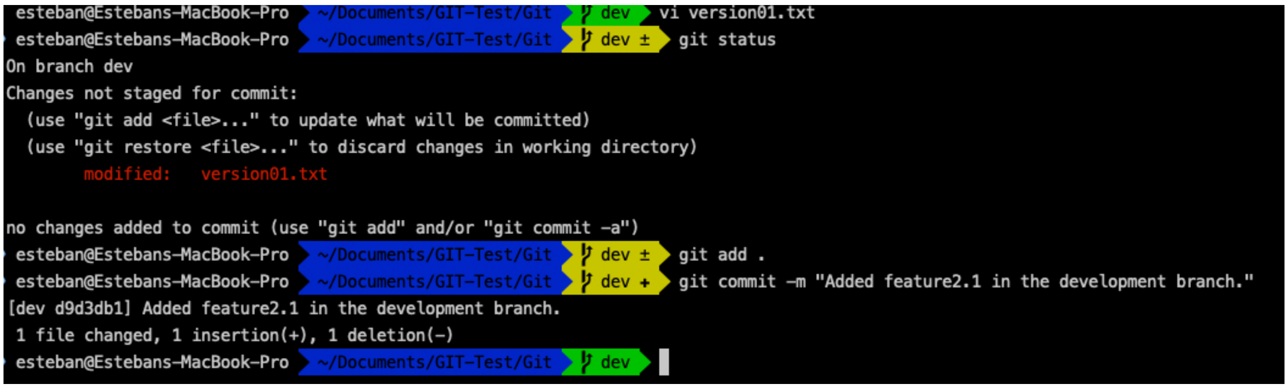
In this task, we will work with the `version01.txt` file in the development branch and perform a rebase operation. Follow the steps below:

1. Open the `version01.txt` file in the development branch.

2. Add the following lines after "This is the bug fix in the development branch" that you added on Day 10 and reverted to this commit:

```
After bug fixing, this is the new feature with minor alteration
```

3. Commit this change with the message "Added feature2.1 in the development branch."



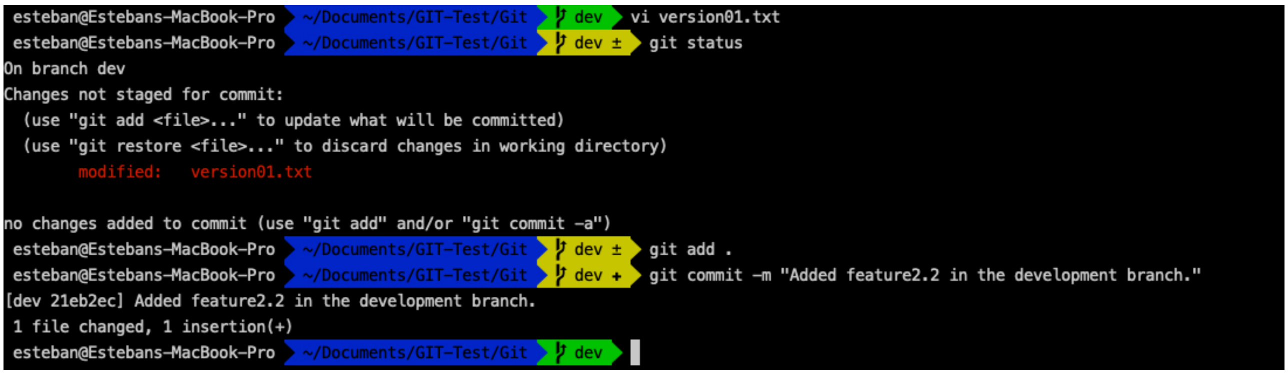
```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev  vi version01.txt
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   version01.txt

no changes added to commit (use "git add" and/or "git commit -a")
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git add .
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev +  git commit -m "Added feature2.1 in the development branch."
[dev d9d3db1] Added feature2.1 in the development branch.
 1 file changed, 1 insertion(+), 1 deletion(-)
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev 
```

4. Add the following line after the previous commit:

```
This is the advancement of the previous feature
```

5. Commit this change with the message "Added feature2.2 in the development branch."



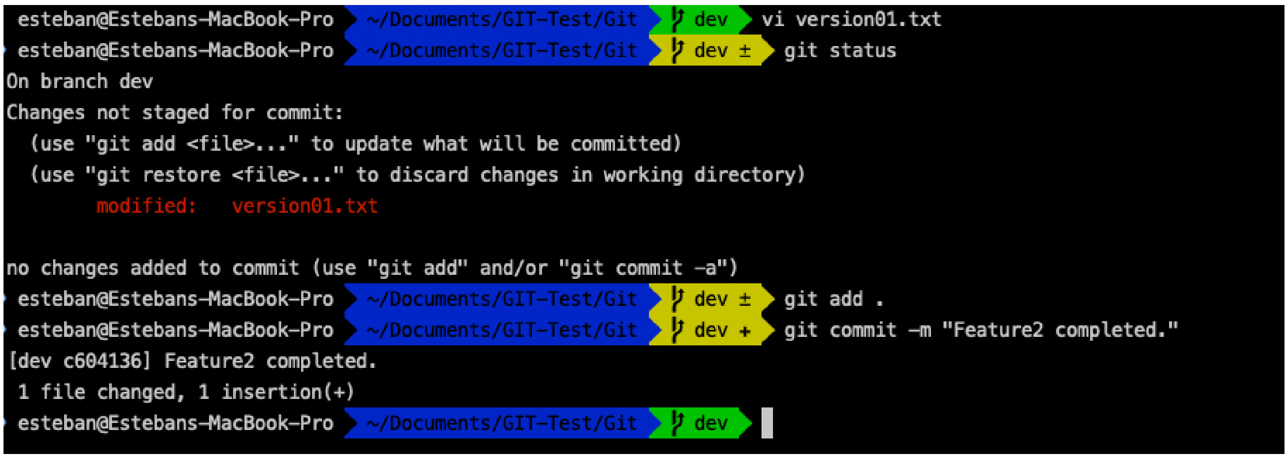
```
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev  vi version01.txt
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   version01.txt

no changes added to commit (use "git add" and/or "git commit -a")
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git add .
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev +  git commit -m "Added feature2.2 in the development branch."
[dev 21eb2ec] Added feature2.2 in the development branch.
 1 file changed, 1 insertion(+)
esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev 
```

6. Add the following line after the previous commit:

```
 Feature 2 is completed and ready for release
```

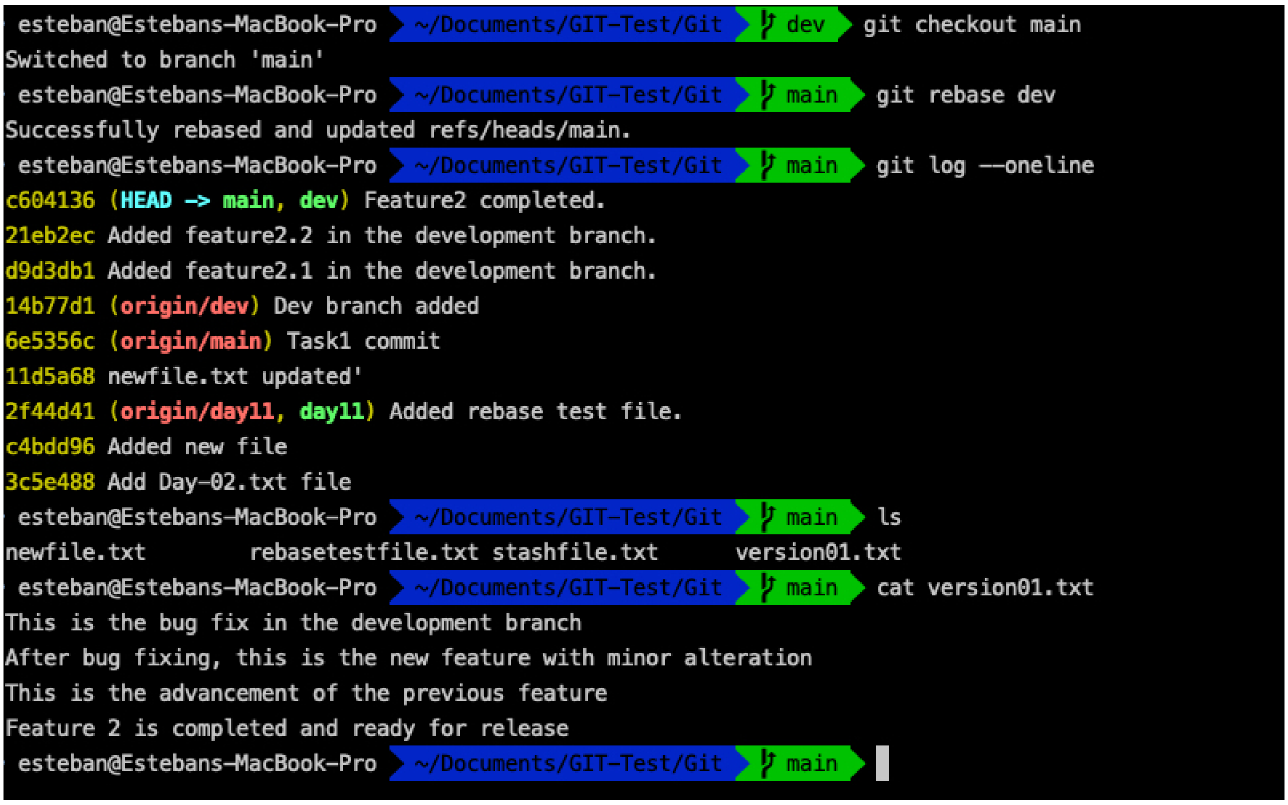
7. Commit this change with the message "Feature2 completed."



```
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev  vi version01.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   version01.txt

no changes added to commit (use "git add" and/or "git commit -a")
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev ±  git add .
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev +  git commit -m "Feature2 completed."
[dev c604136] Feature2 completed.
 1 file changed, 1 insertion(+)
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev 
```

8. Switch to the `main` branch and use `git rebase` to add all the commits from the `dev` branch into the `main` branch



```
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  dev  git checkout main
Switched to branch 'main'
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  git rebase dev
Successfully rebased and updated refs/heads/main.
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  git log --oneline
c604136 (HEAD -> main, dev) Feature2 completed.
21eb2ec Added feature2.2 in the development branch.
d9d3db1 Added feature2.1 in the development branch.
14b77d1 (origin/dev) Dev branch added
6e5356c (origin/main) Task1 commit
11d5a68 newfile.txt updated'
2f44d41 (origin/day11, day11) Added rebase test file.
c4bdd96 Added new file
3c5e488 Add Day-02.txt file
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  ls
newfile.txt         rebasetestfile.txt stashfile.txt      version01.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main  cat version01.txt
This is the bug fix in the development branch
After bug fixing, this is the new feature with minor alteration
This is the advancement of the previous feature
Feature 2 is completed and ready for release
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  main 
```

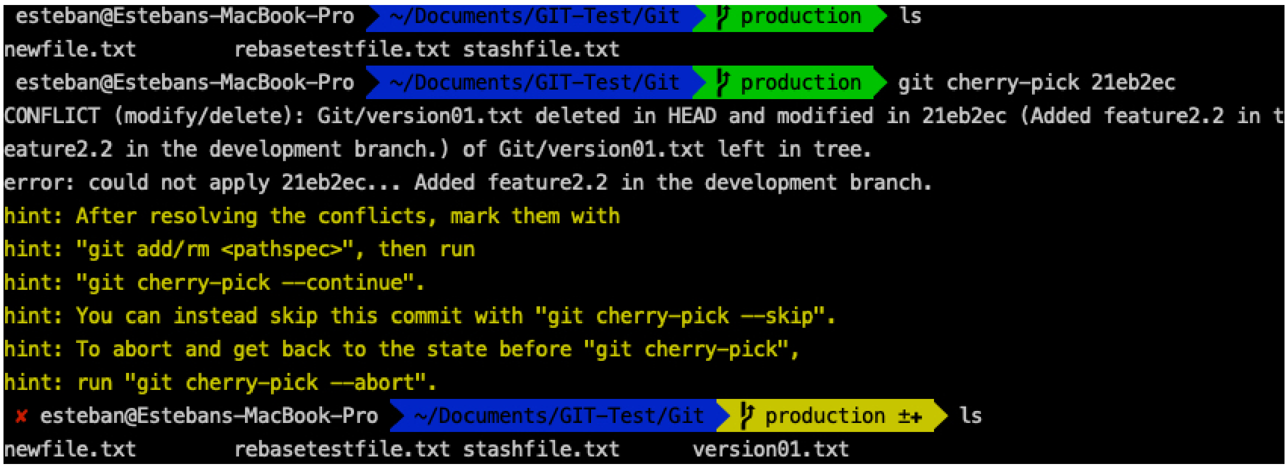
Task 3: Cherry-picking and Optimizing Features

In this task, we will cherry-pick a specific commit from one branch to another and make further changes to it. Follow the steps below:

1. Switch to the `production` branch.

2. Cherry-pick the commit "Added feature2.2 in development branch" using the command:

```
git cherry-pick <commit-hash>
```

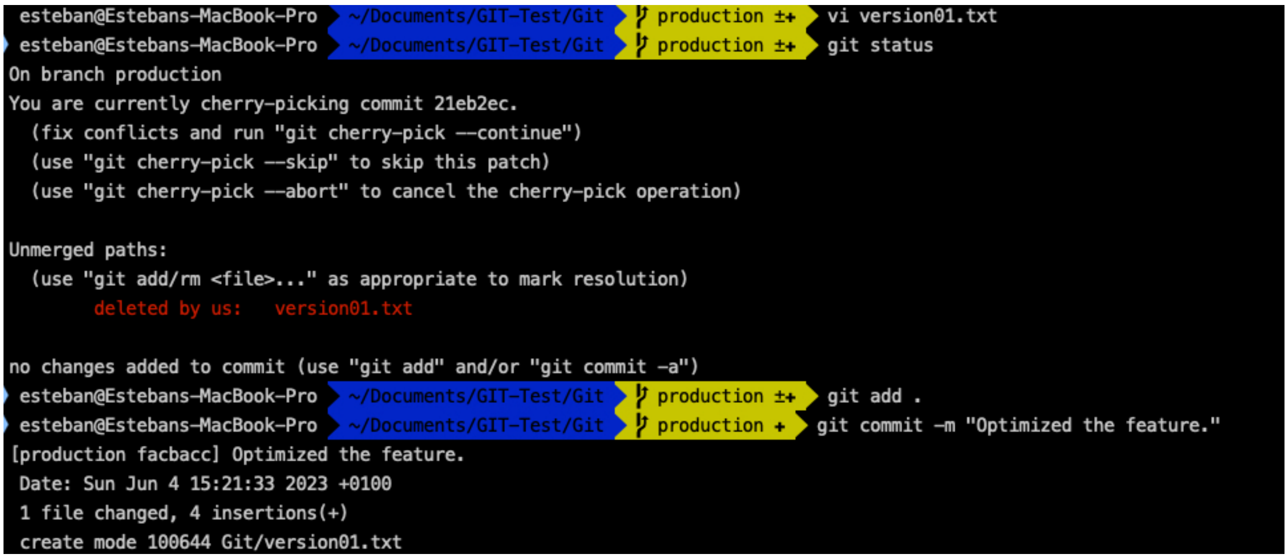


```
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production  ls
newfile.txt         rebasetestfile.txt stashfile.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production  git cherry-pick 21eb2ec
CONFLICT (modify/delete): Git/version01.txt deleted in HEAD and modified in 21eb2ec (Added feature2.2 in t
eature2.2 in the development branch.) of Git/version01.txt left in tree.
error: could not apply 21eb2ec... Added feature2.2 in the development branch.
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
 ✘ esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production ±+  ls
newfile.txt         rebasetestfile.txt stashfile.txt       version01.txt
```

3. Add the following lines after "This is the advancement of the previous feature":

```
Added few more changes to make it more optimized.
```

4. Commit this change with the message "Optimized the feature."



```
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production ±+  vi version01.txt
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production ±+  git status
On branch production
You are currently cherry-picking commit 21eb2ec.
  (fix conflicts and 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)

Unmerged paths:
  (use "git add/rm <file>..." as appropriate to mark resolution)
        deleted by us:   version01.txt

no changes added to commit (use "git add" and/or "git commit -a")
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production ±+  git add .
 esteban@Estebans-MacBook-Pro  ~/Documents/GIT-Test/Git  production +  git commit -m "Optimized the feature."
[production facbacc] Optimized the feature.
 Date: Sun Jun 4 15:21:33 2023 +0100
 1 file changed, 4 insertions(+)
 create mode 100644 Git/version01.txt
```

Congratulations on completing Day 11 of the #90DaysOfDevOps challenge! Today, we explored advanced Git techniques such as stashing, cherry-picking, and rebasing. These techniques are powerful tools in managing changes and collaborating effectively within Git repositories.

Stay tuned for Day 12 of the #90DaysOfDevOps challenge, where we will wrap up our Linux and Git session, diving deeper into their features and techniques that will serve as a strong foundation for our next adventure with Python from Day 13.