18 Git Commands I Learned During My First Year as a Software Developer

A practical Git cheat sheet for junior software developers

Ahmad Abdullah Feb 5 · 10 min read

Photo by Tim van der Kuip on Unsplash

Git is one of the most widely used free and open-source distributed version control system (VCS). In the past few years, Git has transitioned from being a preferred skill to a must-have skill for developers. It is impossible to imagine a project without a VCS, which streamlines the development process.

Back in 2018, I started off my professional journey as a software developer. I was familiar with software development but Git was uncharted territory. It was confusing and intimidating. Later I found that some of the senior developers were still afraid of it. I was lucky enough to have a good mentor who guided me through every step.

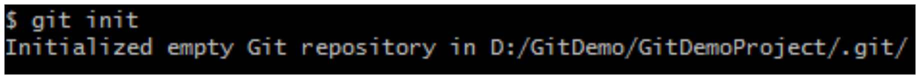
During my training, my manager asked me to work on a web development project from scratch. He gave me a vague list of requirements and asked me to figure out the rest. It was the first time I felt out of my comfort zone. It was also the first time I used Git.

Here are the 18 Git commands I learned during my first year as a software developer. Technically, there are more than 18, but I have grouped related commands under one heading. You can find the complete Git guide here.

1) Create a new repository

Setting up my first professional development project was exciting and terrifying. This not an exaggeration if you have no idea what you are doing. I was asked to add this project on Gitlab. The IT team had already set up Git on my Linux system. It was time to create a new Git repository.

```
git init
```



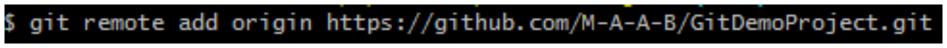
```
$ git init
Initialized empty Git repository in D:/GitDemo/GitDemoProject/.git/
```

Git command to init or create new repository — Image by author

Executing this command creates a local repository with a default *main*(or *master*) branch.

To connect this local repository to Github, we need to create a remote repository on Github. And connect the remote repository's origin with the local repository.

```
git remote add origin https://github.com/YOUR-USERNAME/YOUR-
REPOSITORY
```

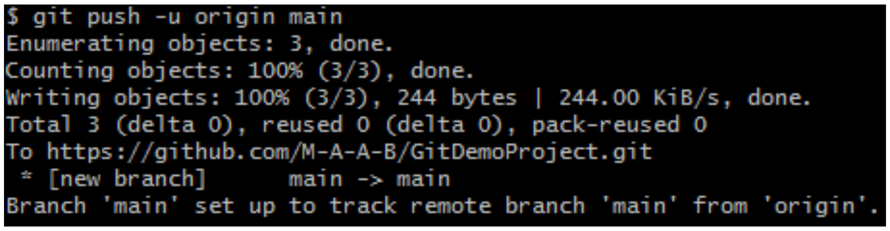


```
$ git remote add origin https://github.com/M-A-A-B/GitDemoProject.git
```

Git command to add remote origin to the local repository — Image by author

Finally, push the main branch on Github.

```
git push -u REMOTE-NAME BRANCH-NAME
```



```
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 244 bytes | 244.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/M-A-A-B/GitDemoProject.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

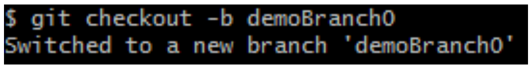
Git command to push the local main branch — Image by author

Note: Execute all Git commands via terminal(Linux) or Git Bash(Windows) in the project working directory.

2) Make a new branch

The gold standard for implementing a new feature is to create a new branch and put all the code in it. This keeps the existing code safe from bad implementation.

```
git checkout -b NEW-BRANCH-NAME
```

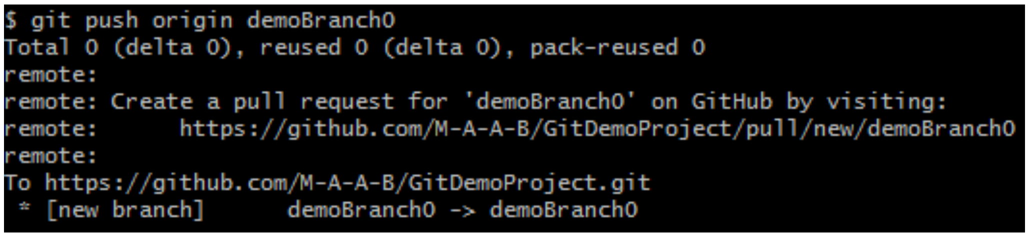


```
$ git checkout -b demoBranch0
Switched to a new branch 'demoBranch0'
```

Git command to create a new branch — Image by author

The checkout command creates the new branch if it does not exist in the repository. While the `-b` option switches or checks out from the current branch to the newly created branch. It is always better to first switch to the main(master) branch before creating new branches. The main branch usually has the most up to date code.

Once we create a new local branch, it should be pushed to the remote repository — as done before.



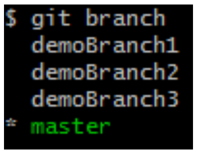
```
$ git push origin demoBranch0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'demoBranch0' on GitHub by visiting:
remote:      https://github.com/M-A-A-B/GitDemoProject/pull/new/demoBranch0
remote:
To https://github.com/M-A-A-B/GitDemoProject.git
 * [new branch]      demoBranch0 -> demoBranch0
```

Git command to push the newly created branch — Image by author

3) Switch branch

When you start working on a new project, it is always better to have a general idea about the branches in the repository.

```
git branch
```

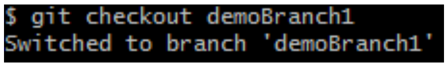


```
$ git branch
  demoBranch1
  demoBranch2
  demoBranch3
* master
```

Git command to list all branches in the repository — Image by author

After listing all the branches, use the following command to switch to the required branch:

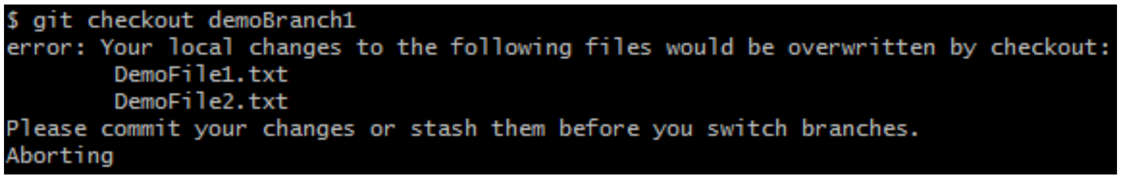
```
git checkout BRANCH-NAME
```



```
$ git checkout demoBranch1
Switched to branch 'demoBranch1'
```

Git command to switch the branch — Image by author

The checkout command works well if you have no uncommitted updated files in the current branch. Otherwise, these uncommitted changes will cause an error. It is always better to commit or stash the changes in your current branch before switching to another branch.



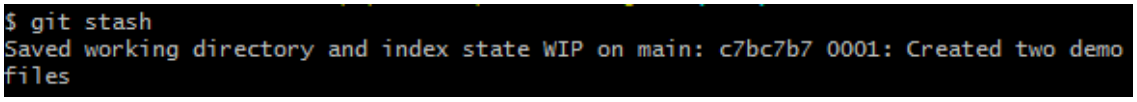
```
$ git checkout demoBranch1
error: Your local changes to the following files would be overwritten by checkout:
        DemoFile1.txt
        DemoFile2.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

Checkout error due to uncommitted changes—Image by author

4) Stash/un-stash files

One of the ways to resolve the checkout error is to stash the changes. This is usually done temporarily to save the work you have done so far in the current branch — if you are not ready to commit these changes.

```
git stash
```

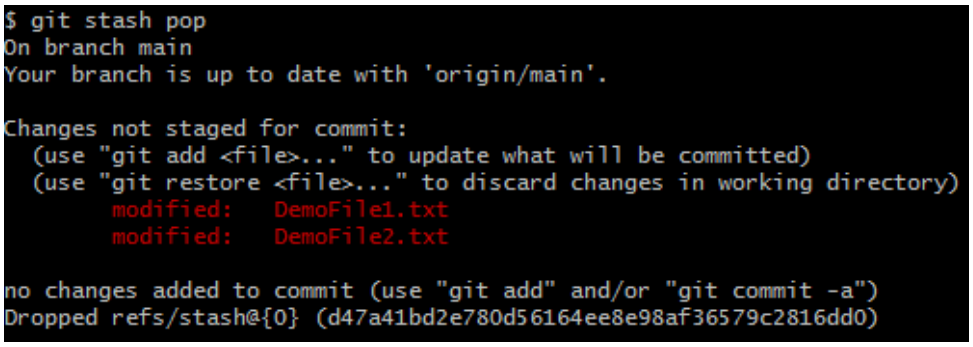


```
$ git stash
Saved working directory and index state WIP on main: c7bc7b7 0001: Created two demo
files
```

Git command to stash uncommitted changes — Image by author

To recover or undo the stashed changes, we can come back to the branch where we stashed the changes and pop them.

```
git stash pop
```



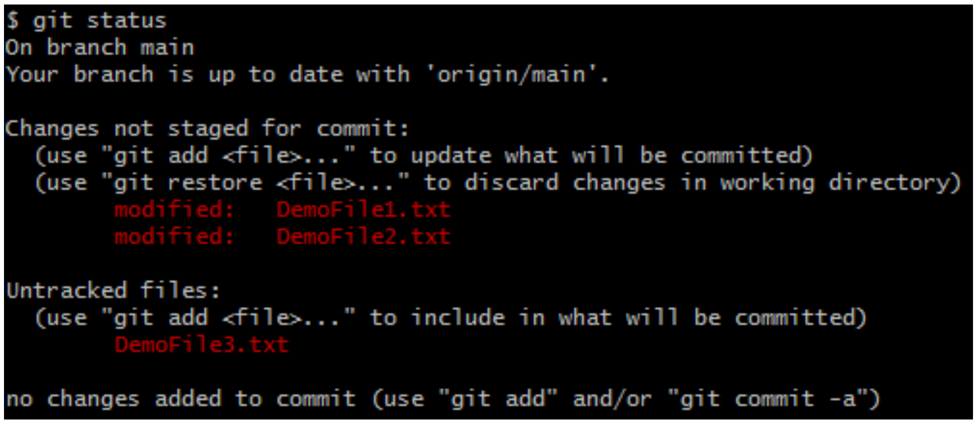
```
$ git stash pop
On branch main
Your branch is up to date with 'origin/main'.

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:   DemoFile1.txt
        modified:   DemoFile2.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (d47a41bd2e780d56164ee8e98af36579c2816dd0)
```

Git command to recover the stashed changes— Image by author

5) Check branch status

I have a weird habit of checking branch status quite frequently. It gives all the necessary information about the current status of the branch. We can check all the staged or untracked changes.

```
git status
```



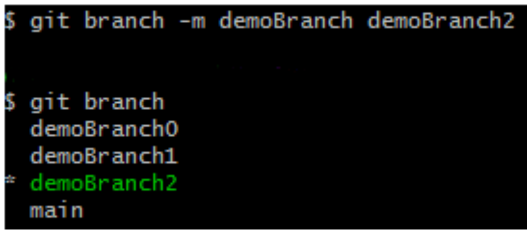
```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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:   DemoFile1.txt
        modified:   DemoFile2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DemoFile3.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Git command to check the status of a branch — Image by author

6) Rename local branch

Branch renaming is not one of the more frequently used Git commands but it’s handy when there is a typo. Or in my case, renaming branches to make a consistent Git directory structure.

```
git branch -m OLD-BRANCH-NAME NEW-BRANCH-NAME
```



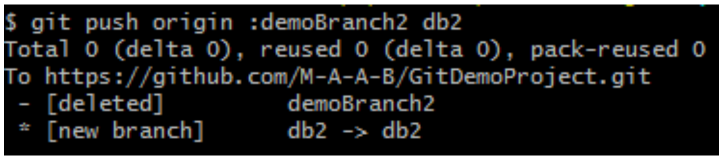
```
$ git branch -m demoBranch demoBranch2

$ git branch
  demoBranch0
  demoBranch1
* demoBranch2
  main
```

Git command to rename a branch — Image by author

7) Rename remote branch

Once you have renamed the local branch, it is time to change the name of the corresponding remote branch.

```
git push origin :OLD-BRANCH-NAME NEW-BRANCH-NAME
```



```
$ git push origin :demoBranch2 db2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/M-A-A-B/GitDemoProject.git
 - [deleted]         demoBranch2
 * [new branch]      db2 -> db2
```

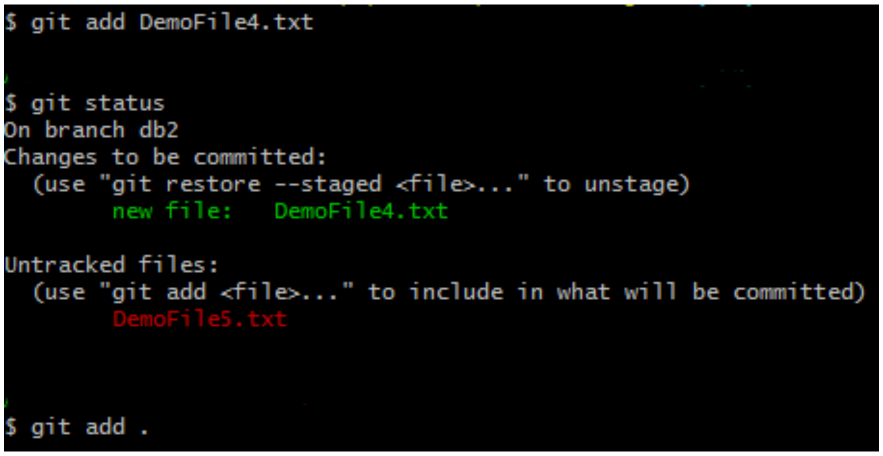
Git command to rename remote branch — Image by author

This command deletes the branch with the old name and creates a new branch with the same code base.

8) Synchronise branch changes

Once a new file has been created or an existing file has been updated in your project, we have to add those files to the next commit.

```
# To add all changes
git add .
# To add a specific file
git add FILE-PATH
```

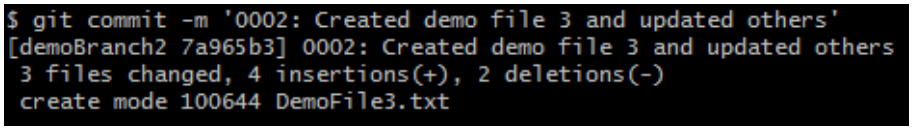


```
$ git add DemoFile4.txt

$ git status
On branch db2
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   DemoFile4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DemoFile5.txt


$ git add .
```

Git command to add a single file or all files — Image by author

When all required changes have been added to the commit, it is time to commit these changes and write a unique commit message. Commit message can be any string that can describe your work in a few words.

```
git commit -m "COMMIT-MESSAGE"
```

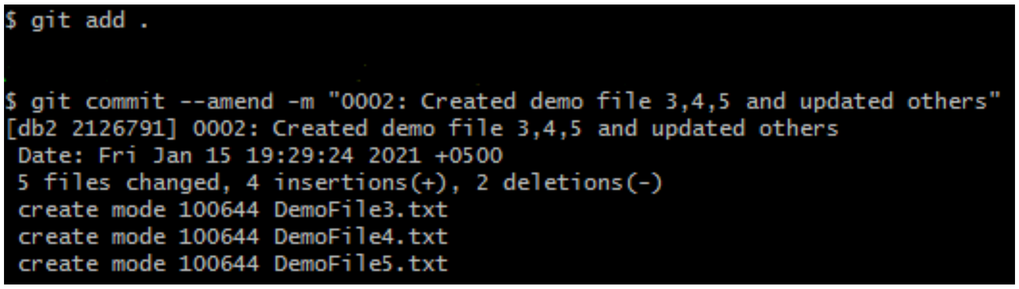


```
$ git commit -m '0002: Created demo file 3 and updated others'
[demoBranch2 7a965b3] 0002: Created demo file 3 and updated others
 3 files changed, 4 insertions(+), 2 deletions(-)
 create mode 100644 DemoFile3.txt
```

Git command to commit changes — Image by author

Finally, push this commit on the remote repository using the `git push` command. You can also update or amend your commit.

```
git add FILE-PATH
git commit --amend -m "CHANGE-COMMIT-MESSAGE"
```



```
$ git add .


$ git commit --amend -m "0002: Created demo file 3,4,5 and updated others"
[db2 2126791] 0002: Created demo file 3,4,5 and updated others
 Date: Fri Jan 15 19:29:24 2021 +0500
 5 files changed, 4 insertions(+), 2 deletions(-)
 create mode 100644 DemoFile3.txt
 create mode 100644 DemoFile4.txt
 create mode 100644 DemoFile5.txt
```

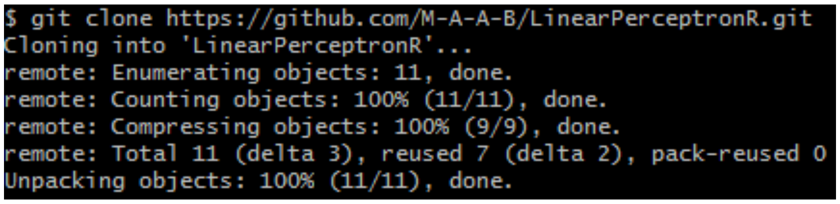
Git command to amend the previous commit — Image by author

This has been one of my favorite Git commands. Code review is very important when you are working in a team. Senior developers or team leads often point out important issues in the code. Instead of creating a new commit to fix these issues, we followed the convention of having one commit per branch — `git amend` proved handy.

9) Clone repository

The first git command I executed for my first live project was `git clone`. I was asked to clone the project on my local system to understand the code, add a few functionalities, and push the code back via a merge request.

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```



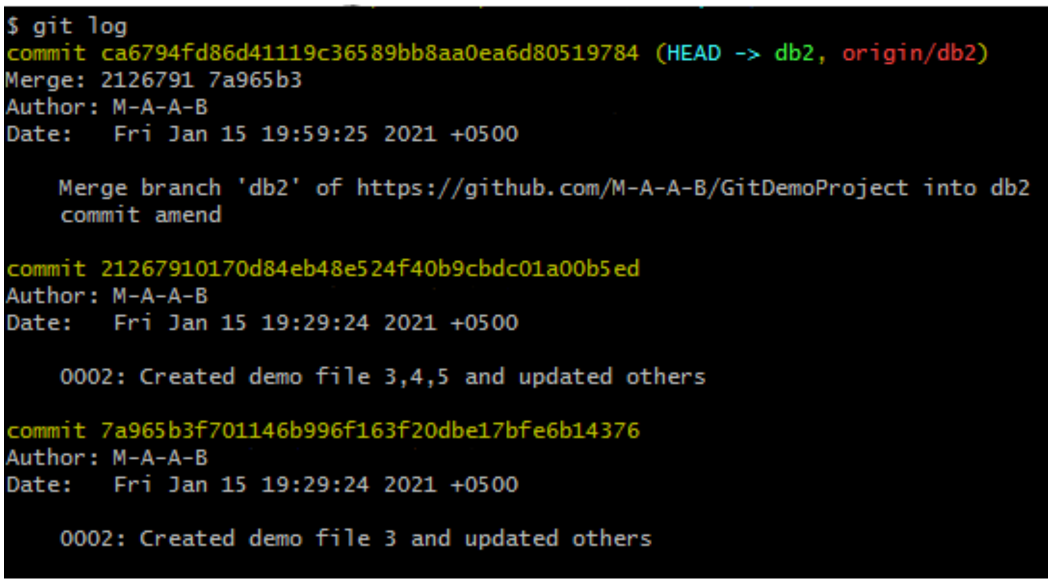
```
$ git clone https://github.com/M-A-A-B/LinearPerceptronR.git
Cloning into 'LinearPerceptronR'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 11 (delta 3), reused 7 (delta 2), pack-reused 0
Unpacking objects: 100% (11/11), done.
```

Git command to clone a repository — Image by author

10) Checking commit log

Logs are an important part of software development. A software maintains log files to keep track of all steps followed during its lifecycle. Git also provides a log to keep track of all the commits.

```
git log
```



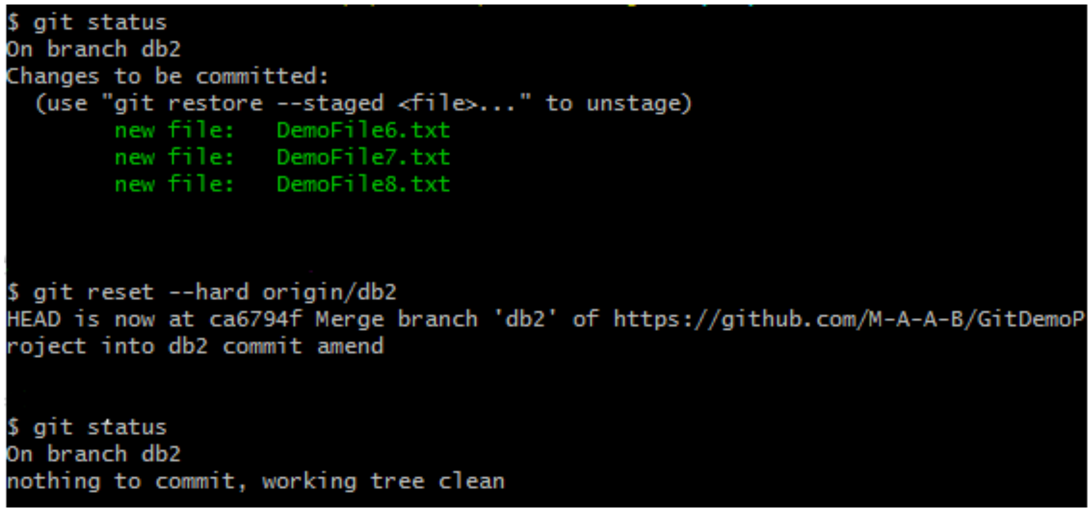
```
$ git log
commit ca6794fd86d41119c36589bb8aa0ea6d80519784 (HEAD -> db2, origin/db2)
Merge: 2126791 7a965b3
Author: M-A-A-B
Date:   Fri Jan 15 19:59:25 2021 +0500

    Merge branch 'db2' of https://github.com/M-A-A-B/GitDemoProject into db2
    commit amend

commit 21267910170d84eb48e524f40b9cbdc01a00b5ed
Author: M-A-A-B
Date:   Fri Jan 15 19:29:24 2021 +0500

    0002: Created demo file 3,4,5 and updated others

commit 7a965b3f701146b996f163f20dbe17bfe6b14376
Author: M-A-A-B
Date:   Fri Jan 15 19:29:24 2021 +0500

    0002: Created demo file 3 and updated others
```

Git command to display the commit log — Image by author

11) Reset to the last commit

Having the power to reset or undo a task is a lifesaver. In my early days as a developer, I once made the mistake of making changes to live production code. It was a blunder. To my surprise, I was not fired. What saved me from doing any more damage was the reset command. It allowed me to revert all changes in an instant to the last working commit.

```
git reset -hard origin/BRANCH-NAME
```



```
$ git status
On branch db2
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   DemoFile6.txt
        new file:   DemoFile7.txt
        new file:   DemoFile8.txt


$ git reset --hard origin/db2
HEAD is now at ca6794f Merge branch 'db2' of https://github.com/M-A-A-B/GitDemoP
roject into db2 commit amend


$ git status
On branch db2
nothing to commit, working tree clean
```

Git command to reset the codebase to last working commit — Image by author

12) Merge local repository with the remote repository

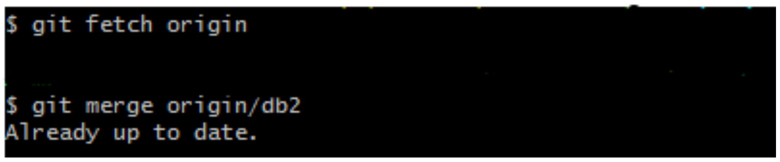
This is where the magic happens. When we are working on software, we usually maintain three copies of the codebase.

One is the local copy on which the developer works and performs the required tasks. Second is the staging copy or staging server where these tasks are deployed and shared with the client. The client gives feedback on all the new changes made to the software. The development team then works on these changes and deploy them back on the staging server. This cycle continues till the client approves these changes to be deployed on the third and final production server.

Every time a new feature is deployed or an existing feature is updated we perform the merge operation. Updated code files from the Github remote repository are merged with the local repository on these three servers.

The most common issue while merging repositories is the merge conflict. These conflicts must be resolved to complete the merge operation.

```
# Fetch remote from github
git fetch REMOTE-NAME
# Merge remote branch with local branch
git merge REMOTE-NAME/BRANCH-NAME
```



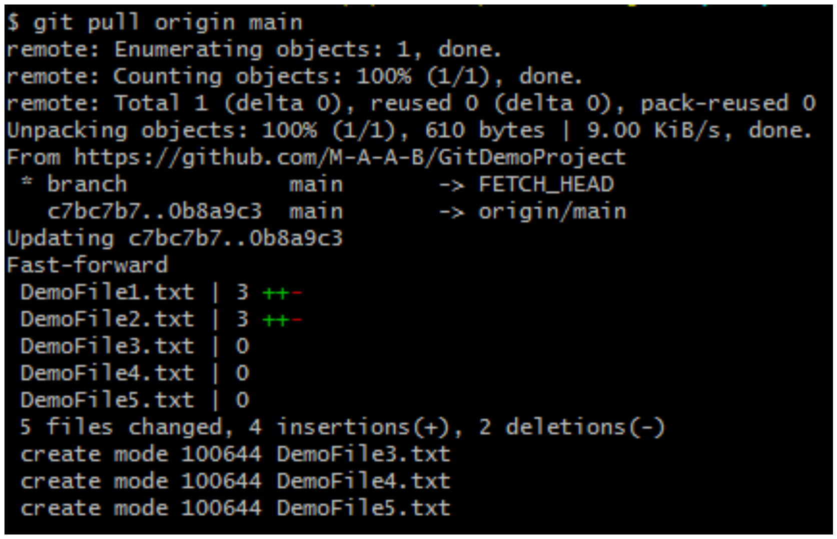
```
$ git fetch origin


$ git merge origin/db2
Already up to date.
```

Git commands to merge the remote branch with local branch — Image by author

The merge operation is performed in two steps. We fetch or download the remote from Github which contains the required codebase. Then merge the remote and local branch histories.

Another way to perform merge is `git pull`. Pulling works in the same way as merge, with the added benefit of fetching. Instead of performing the two operations separately like shown above, `git pull` performs both *fetch* and *merge* — joining two or more branches.

```
git pull REMOTE-NAME BRANCH-NAME
```



```
$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 610 bytes | 9.00 KiB/s, done.
From https://github.com/M-A-A-B/GitDemoProject
 * branch            main       -> FETCH_HEAD
   c7bc7b7..0b8a9c3  main       -> origin/main
Updating c7bc7b7..0b8a9c3
Fast-forward
 DemoFile1.txt | 3 ++-
 DemoFile2.txt | 3 ++-
 DemoFile3.txt | 0
 DemoFile4.txt | 0
 DemoFile5.txt | 0
 5 files changed, 4 insertions(+), 2 deletions(-)
 create mode 100644 DemoFile3.txt
 create mode 100644 DemoFile4.txt
 create mode 100644 DemoFile5.txt
```

Git command to pull changes from the remote branch and merge them with local branch — Image by author

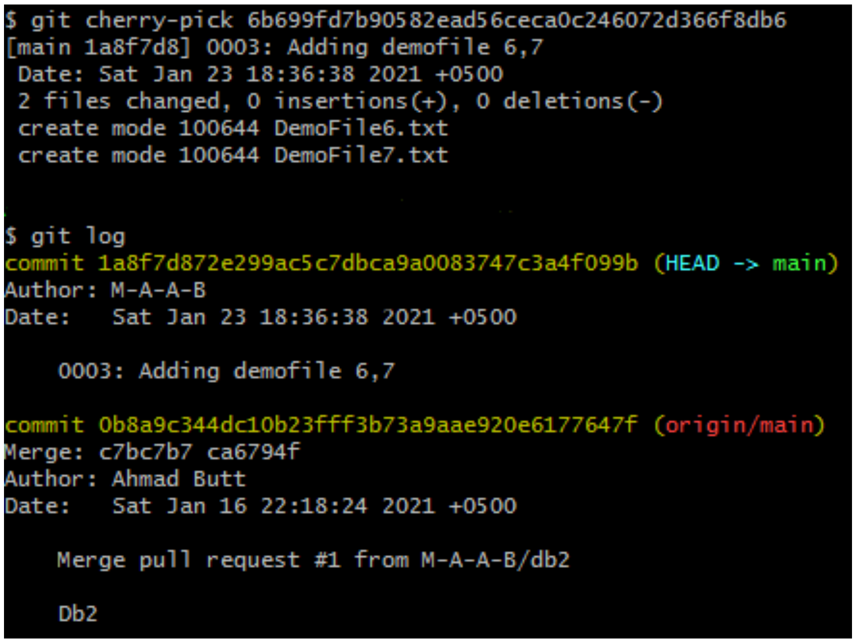
13) Move a commit from one branch to another

When you are collaborating on a project, it is preferred that each developer works on a separate feature — too good to be true. Depending on the complexity of a task, it is divided among multiple developers.

High cohesion and low coupling are often ignored or impractical. This creates many problems during development.

I have worked on several such features. In most cases, I was required to get the unfinished code from some other branch and try to expand on it while keeping the unfinished parts in mind as well. `git cherry-pick` played an important role.

```
git cherry-pick COMMIT-HASH
```



```
$ git cherry-pick 6b699fd7b90582ead56ceca0c246072d366f8db6
[main 1a8f7d8] 0003: Adding demofile 6,7
 Date: Sat Jan 23 18:36:38 2021 +0500
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 DemoFile6.txt
 create mode 100644 DemoFile7.txt


$ git log
commit 1a8f7d872e299ac5c7dbca9a0083747c3a4f099b (HEAD -> main)
Author: M-A-A-B
Date:   Sat Jan 23 18:36:38 2021 +0500

    0003: Adding demofile 6,7

commit 0b8a9c344dc10b23fff3b73a9aae920e6177647f (origin/main)
Merge: c7bc7b7 ca6794f
Author: Ahmad Butt
Date:   Sat Jan 16 22:18:24 2021 +0500

    Merge pull request #1 from M-A-A-B/db2

    Db2
```

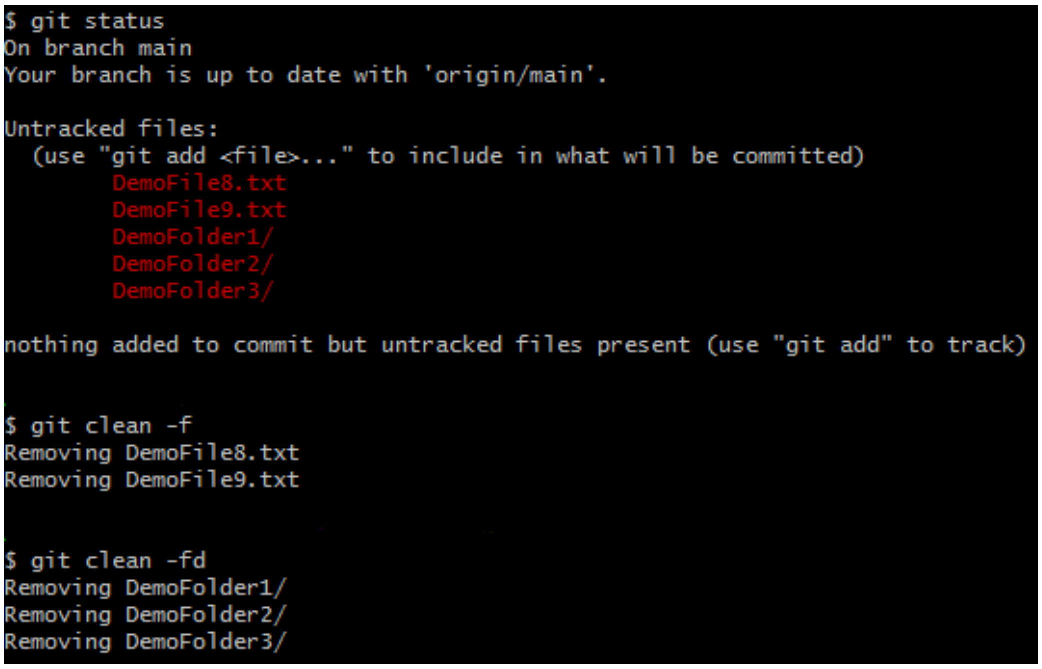
Git command to move a commit from one branch to another — Image by author

This command applies the selected commit on the current branch, which is also quite handy during bug hotfixes. Though cherry-picking is useful, it is not always the best practice. It can cause duplicate commits, which is why *merge* is preferred in most scenarios.

14) Remove untracked files & directories

Files and directories which have not been committed yet can be easily removed from the working directory using `git clean`. I use it to remove unwanted files and directories created by my IDE.

```
# To remove untracked files
git clean -f
# TO remove untracked directories
git clean -fd
```



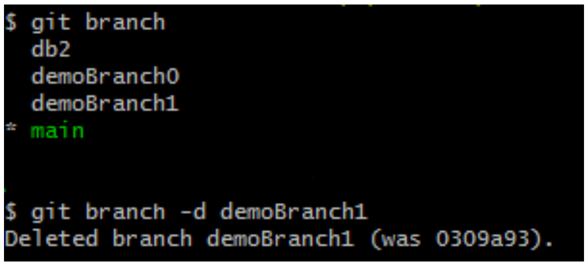
```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DemoFile8.txt
        DemoFile9.txt
        DemoFolder1/
        DemoFolder2/
        DemoFolder3/

nothing added to commit but untracked files present (use "git add" to track)


$ git clean -f
Removing DemoFile8.txt
Removing DemoFile9.txt


$ git clean -fd
Removing DemoFolder1/
Removing DemoFolder2/
Removing DemoFolder3/
```

Git command to remove unwanted files and folders — Image by author

15) Delete a branch on a local repository

If a branch is no longer required it is always better to clean the repository by deleting that branch. To remove the branch in the local repository, use `git branch` with `-d` option.

```
git branch -d BRANCH-NAME
# To forcefully delete a local branch. Be careful
git branch -D BRANCH-NAME
```



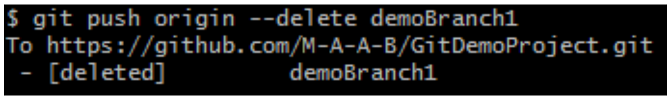
```
$ git branch
  db2
  demoBranch0
  demoBranch1
* main


$ git branch -d demoBranch1
Deleted branch demoBranch1 (was 0309a93).
```

Git command to delete a local branch — Image by author

16) Delete a branch on a remote repository

Deleting a branch on the remote repository is similar to pushing an update on remote using the same `git push` command with `--delete` option.

```
git push REMOTE-NAME --delete BRANCH-NAME
```



```
$ git push origin --delete demoBranch1
To https://github.com/M-A-A-B/GitDemoProject.git
 - [deleted]         demoBranch1
```

Git command to delete a remote branch — Image by author

17) Ignore Git permission changes

I worked on a Linux based system where setting file permissions using `chmod` are very important for security. During development, we normally change the mode of files to *777* to make them executable. Git picks up on these permission changes and shows them as updated files in `git status` .

```
git config core.fileMode false
```

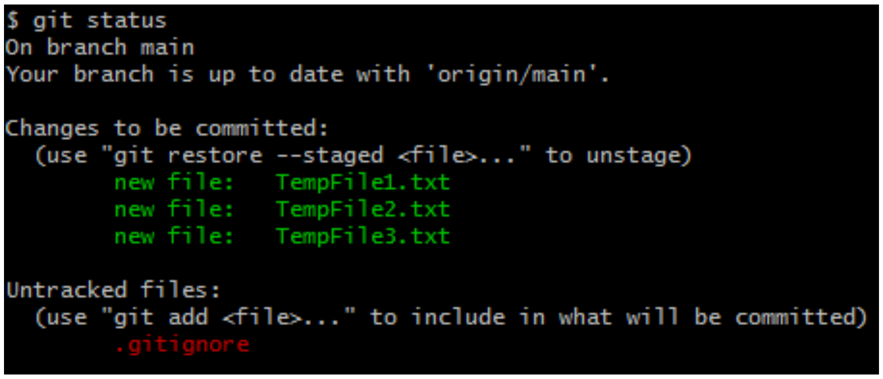
Git allows us to ignore these changes by changing the `fileMode` to *false* in its configuration.

18) Fix .gitignore

.gitignore file is a blessing that helps us to ignore committing unwanted files to the repository. But if the files are already being tracked by Git, it becomes a pain.

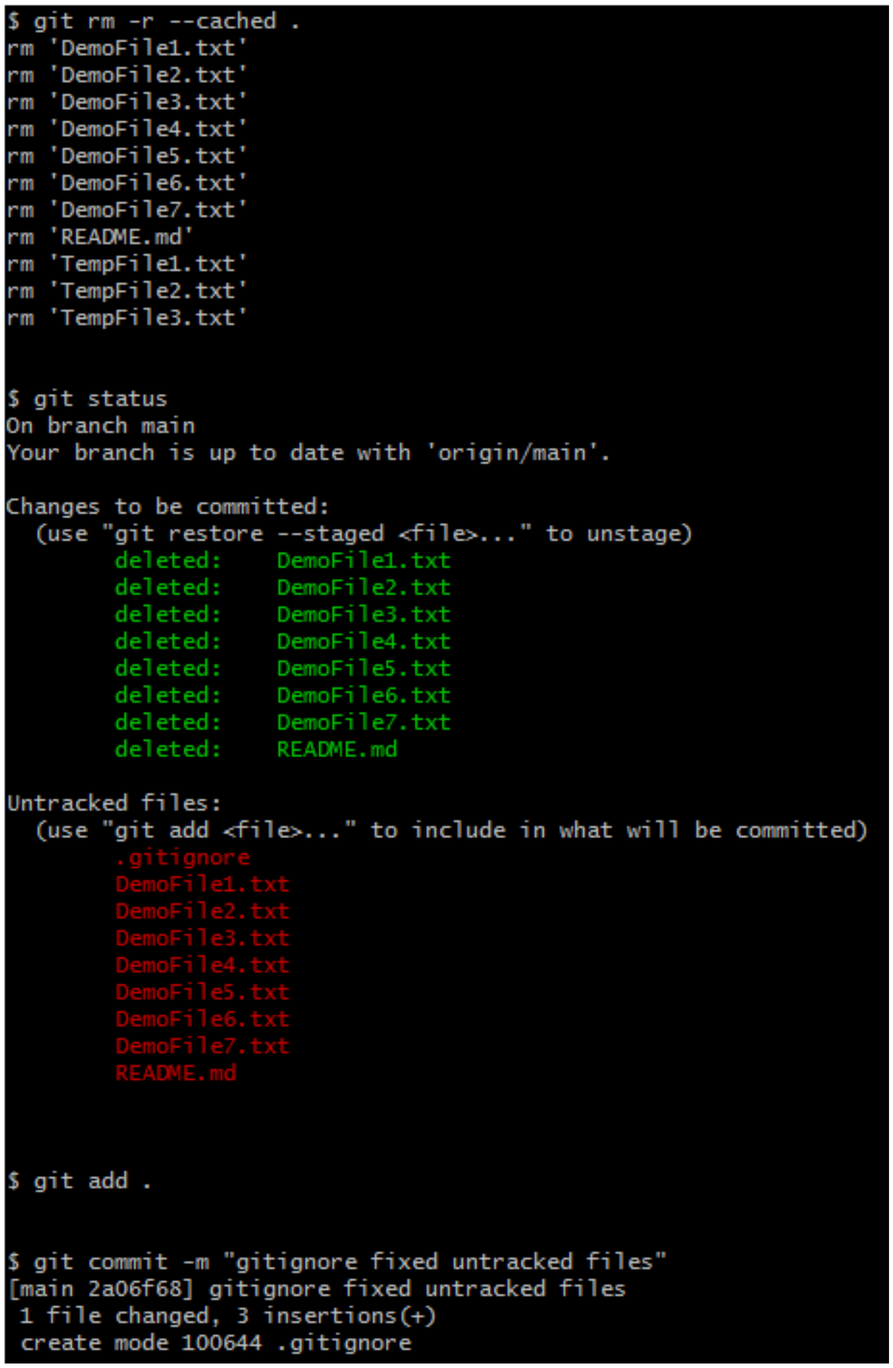
Thankfully, there is an easy fix. It requires removing the cached .gitignore index and adding it to the repository again.

```
# To create a new .gitignore file
touch .gitignore

# To untrack the unnecessary tracked files in your gitignore which
removes everything from its index. Specific filenames can also be
used instead of dot(.).
git rm -r --cached .
git add .
git commit -m "gitignore fixed untracked files"
```



```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   TempFile1.txt
        new file:   TempFile2.txt
        new file:   TempFile3.txt

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

Git command to check the status — Image by author



```
$ git rm -r --cached .
rm 'DemoFile1.txt'
rm 'DemoFile2.txt'
rm 'DemoFile3.txt'
rm 'DemoFile4.txt'
rm 'DemoFile5.txt'
rm 'DemoFile6.txt'
rm 'DemoFile7.txt'
rm 'README.md'
rm 'TempFile1.txt'
rm 'TempFile2.txt'
rm 'TempFile3.txt'


$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    DemoFile1.txt
        deleted:    DemoFile2.txt
        deleted:    DemoFile3.txt
        deleted:    DemoFile4.txt
        deleted:    DemoFile5.txt
        deleted:    DemoFile6.txt
        deleted:    DemoFile7.txt
        deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        DemoFile1.txt
        DemoFile2.txt
        DemoFile3.txt
        DemoFile4.txt
        DemoFile5.txt
        DemoFile6.txt
        DemoFile7.txt
        README.md



$ git add .


$ git commit -m "gitignore fixed untracked files"
[main 2a06f68] gitignore fixed untracked files
 1 file changed, 3 insertions(+)
 create mode 100644 .gitignore
```

Git commands to fix .gitignore — Image by author

Concluding Thoughts

This is the tip of the iceberg. Like anything else, Git requires practice and experimentation. Mastering Git is a lot harder. Within one year, I was only able to learn the basics of it.

For junior developers, it also requires overcoming the fear of learning something new. A developer's portfolio is incomplete without Git — like going to a war without any weapons.