Git is a very useful version control tool that helps developers to collaborate efficiently. However understanding how to truly leverage the flexibility and power that git provides can often be difficult even for experienced developers in the industry. In this article we will try to understand different features of git with the help of real world scenarios. So without further ado let’s start.

Scenario # 1: Forgot to add a few changes in previous commit / Modifying commit message

Often it is the case that we forget to add a file or a change in a commit. If you ever find yourself in such a situation you don’t need to worry. You can simply use the — amend flag to amend you last commit.

Let say you had following uncommitted files in the start:

* FileA.txt
* FileB.txt

You make some changes in them and then make the commit by following command:

git commit -m ‘First Commit’

Let say, after making a commit you realize that you need to make some further changes in FileB.txt . In that case instead of making a new commit after changing FileB.txt you can use the — amend flag to update the previous commit like following

git commit — amend

Also after some time if you realize that you also need to modify the commit message of the last commit that you made. You can also achieve that by using the amend flag using the following command:

git commit —amend -m ‘First Commit Modified’

Scenario # 2: Pushed a few changes that were not supposed to be commited

It also very common to accidentally push a few changes/files that were not supposed to be committed. Let’s say you had some config files or credentials files like AWS secrets or .pem files. These should not be pushed in the repository. One thing that many people might try in order to remove the pushed .pem from the remote repo is following:

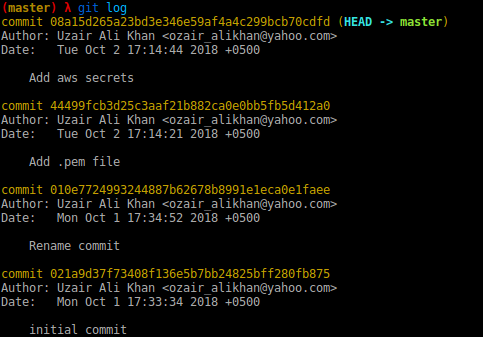
* Pull the latest code from the repo.
* Delete the .pem file
* Create another commit with the deleted .pem file
* Push the changes back to remote repo

However in the above approach we haven’t completely removed the .pem from git. You see git keeps a track of all the files that you commit. So anyone can dig into the history of git commits and access the .pem file.

But git allows you to actually modify history. So the solution would be to move back to your safe commit and force push.

git reset HEAD~n is a command that lets you move back to your previous n commits. So git reset HEAD~2 will take you back two commits and will show you the changes of the last two commits.

git reset HEAD — hard will clean all those changes. So for example lets say you have the following commits



Now let’s say you want to remove the last two commits i.e (commit ‘Add aws secrets’ and ‘Add .pem file’)

First you will go back two commits by running the following command:

git reset HEAD~2

Now you can remove those two commits by using the following command:

git reset HEAD — hard

Now for push the code to remote and your .pem files will be deleted from history as well. **[Warning: Force pushing to remote is VERY RISKY as it modifies the git history on the remote branch. As a rule of thumb you should never force push to master/stage branches. Make sure you are only force pushing to the branch that you own and you understand why you are doing it]**

Scenario # 3: Share a piece of dependent code that cannot be pushed right away

If you face a situation where you need to share some changes in the code base that you have made with your co-worker but you cannot yet push your changes to the remote repository, you can use the patch command of git. Here is how it works:

After you have made all the changes you can run the following command:

git diff > filename.patch

git diff > filename.patch will create a file that will contain all uncommitted the changes you have made so far in the code base. Then you can simple share this file with your coworker via slack,email,USB etc .

You coworker will simply download you patch file and run the following command:

git apply filename.patch

This would apply all the code changes you have made in the code base on your coworker’s copy of the code.

Scenario # 4 Cherry Picking

Cherry picking is a very powerful feature of git. It allows you to pick specific commits from one branch and merge it into another branch. So let’s say develop branch is 3 commits ahead of master, out of those 3, just 2nd commit need to be merged on master. Here is how you would handle this situation in git

* Step 1: Checkout to the master branch
* Step 2: Use git cherry-pick <commit-hash> to merge the specific commit to the master branch

Note: You can view all the commits you have made and their hashes by using the git log command

Now later on if you want to merge the rest of the develop branch with master branch simply run the following command:

git rebase develop