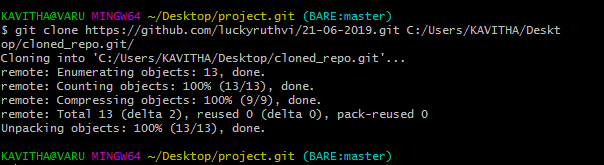
1. **Git work flow**
2. **Process of pushing code to remote repository**

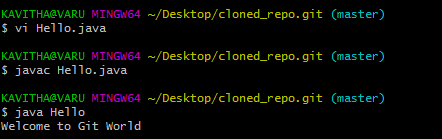
pictorial representation of the Git work-flow is as Shown below

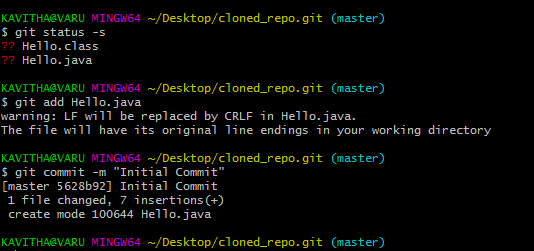


**Clone**

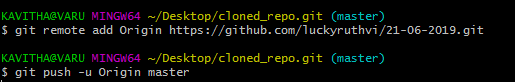
****

**Adding New File Hello.java to Cloned\_repo.git and Commit**

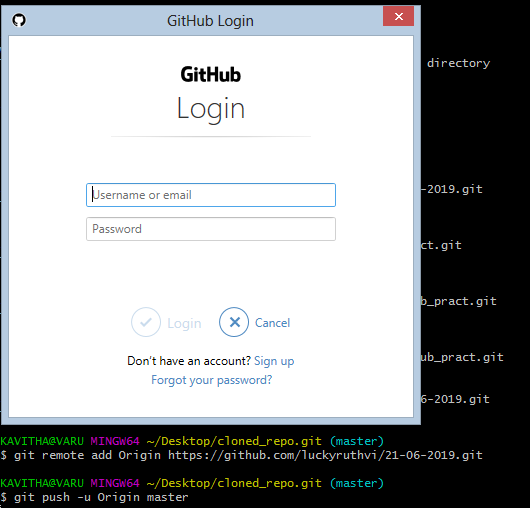
****

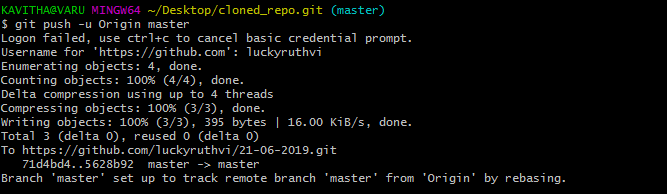
****

**Adding Remote repository and push**

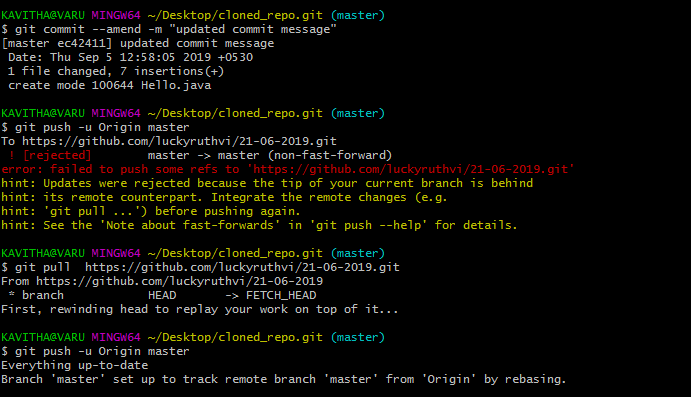
****

**While pushing your modified files to remote repo it will ask for credentials**

****

****

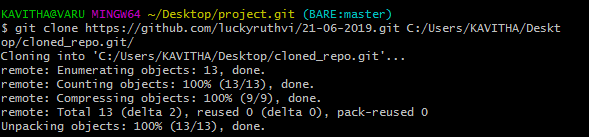
**Git amend and push**

****

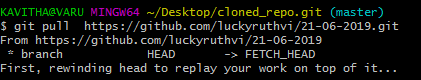
===========================================================================

**3.Git clone, pull, fetch, checkout**

**Git clone**

****

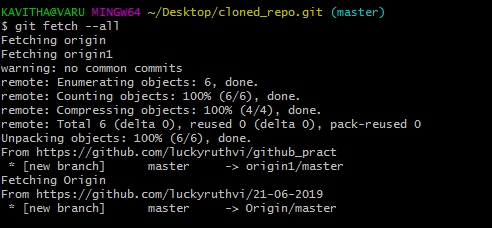
**Git pull**

****

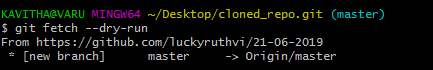
**Git fetch**

****

Fetch all of the branches from the repository. This also downloads all of the required commits and files from the other repository.

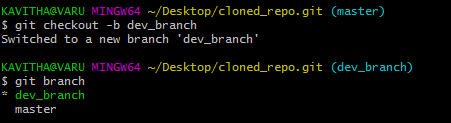
****

A power move which fetches all registered remotes and their branches.

****

The --dry-run option will perform a demo run of the command. I will output examples of actions it will take during the fetch but not apply them.

**Git checkout**

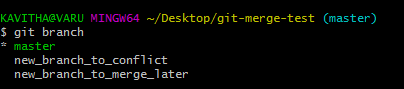


==================================================================================

**4.Git master, branching, Tag**

**Git master**

Git master is the default branch and it contains ready to deploy source code.

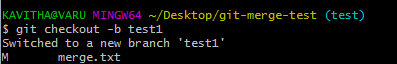


**Branching**

Creating a branch



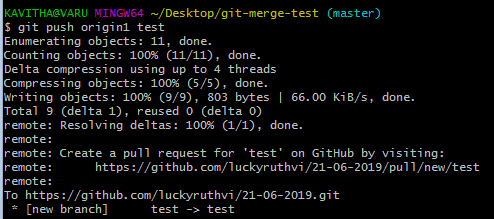
Creating and checking out into a branch



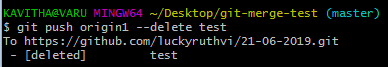
Deleting branch locally



Adding a branch to remote



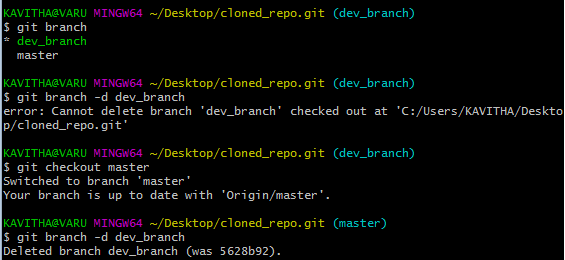
Deleting a branch in github



===========================================================

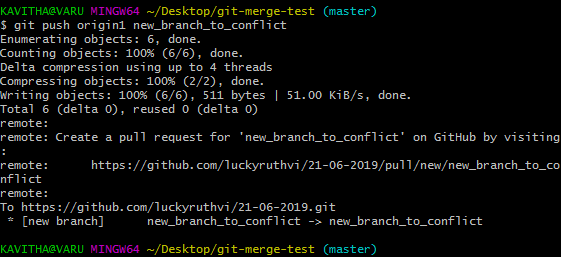
**5.Deleting branches through CLI and GitHub**

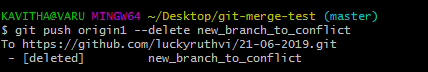
Deleting branches locally

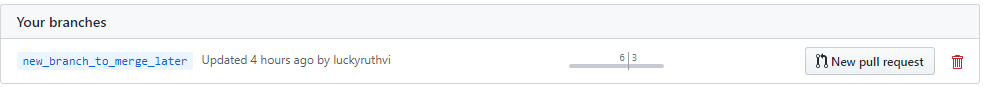
****

Before deleting any branch first we have to checkout to master branch.

Deleting Branches in Github

****

****

****

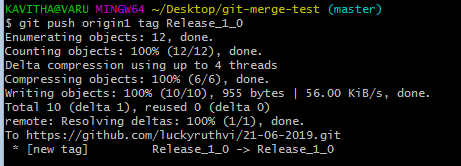
**Tag**

Tag operation allows giving meaningful names to a specific version in the repository. Suppose Tom and Jerry decide to tag their project code so that they can later access it easily.

Creating a tag

****

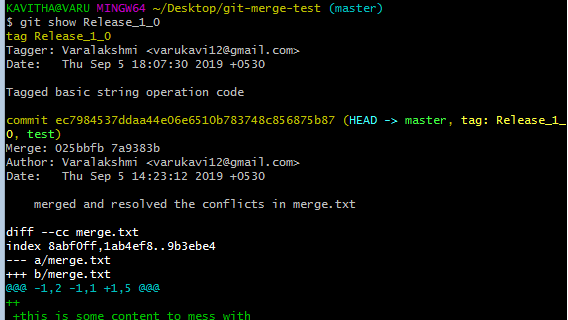
**Adding Tag to Github**

****

**To view tags**

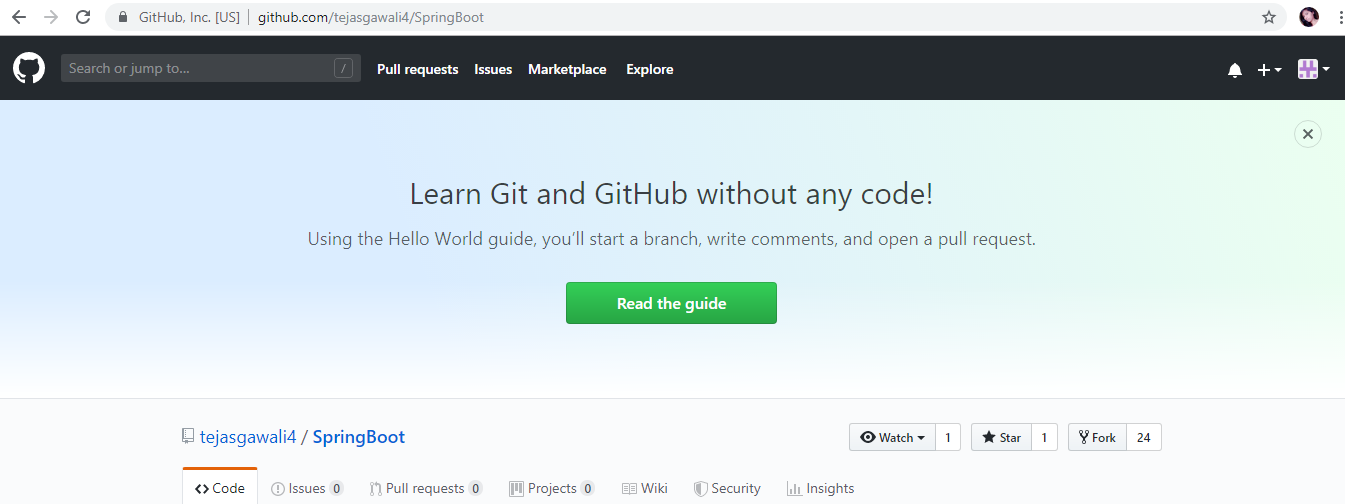
****

**To view more details about tag**

****

**6. Git fork and pull requests**

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, **forks** are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.



Pull requests

**GitHub Branching strategy :**The GitHub flow is, unsurprisingly, the branching strategy favored at GitHub. It proposes a set of simple rules that must be followed:

Code in master is deployable at all times.

When you want to start working on a new task, create a new branch off of master and give it a descriptive name.

Commit to that branch locally and regularly send your work to the same-named branch on the server.

Open a pull request when you feel your changes are ready to be merged (or even if you aren’t so sure, but would like some feedback).

After the new feature is revised and approved, you can merge it into master.

Once your changes are merged and pushed to the master, you can and should deploy immediately.

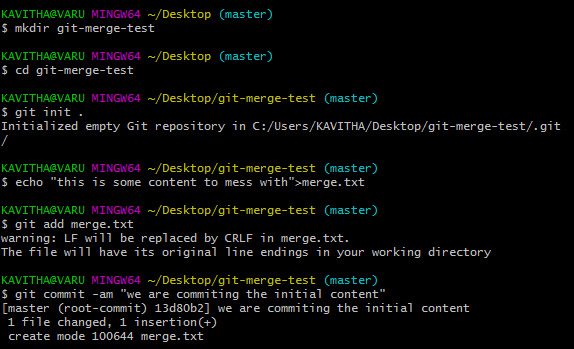
According to the GitHub flow, before you start working on something new, be it a bug fix or a new feature, you should create a new branch off of the master branch and give it a nice, descriptive name.

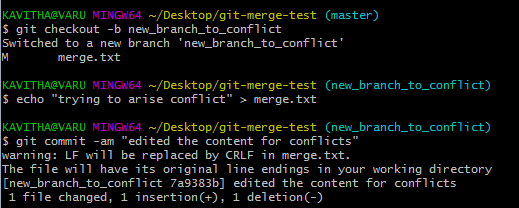
You then start working on your task, adding commits to your newly created branch as you go. You should also continuously push your commits to the branch on the server with the same name. When you think the branch is ready to be merged, you open a **pull request**. After at least one other person has reviewed and approved your changes, they’re ready to be merged into the master branch.

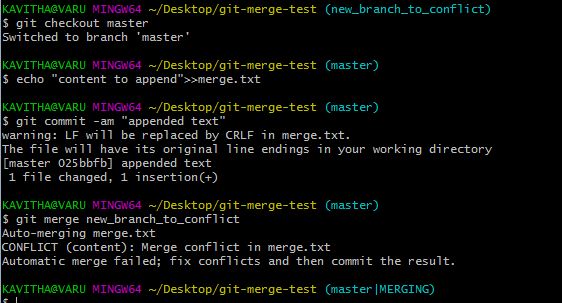
The GitHub flow is also known for encouraging continuous delivery (CD). As soon as your changes are merged, you should deploy to production.

**7. Merge conflict resolution**

## Creating a merge conflict

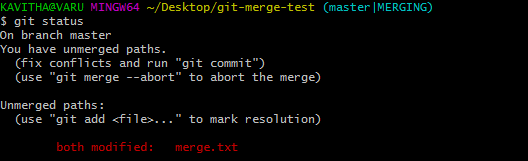




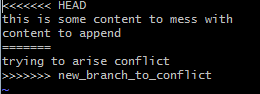




## How to identify merge conflicts



The output from git status indicates that there are unmerged paths due to a conflict. The merge.text file now appears in a modified state. Let's examine the file and see whats modified.



Here we have used the cat command to put out the contents of the merge.txt file. We can see some strange new additions

* <<<<<<< HEAD
* =======
* >>>>>>> new\_branch\_to\_merge\_later

Think of these new lines as "conflict dividers". The ======= line is the "center" of the conflict. All the content between the center and the <<<<<<< HEAD line is content that exists in the current branch master which the HEAD ref is pointing to. Alternatively all content between the center and >>>>>>> new\_branch\_to\_merge\_later is content that is present in our merging branch.

## How to resolve merge conflicts using the command line

The most direct way to resolve a merge conflict is to edit the conflicted file. Open the merge.txt file in your favorite editor. For our example lets simply remove all the conflict dividers. The modified merge.txt content should then look like:



Once the file has been edited use git add merge.txt to stage the new merged content. To finalize the merge create a new commit by executing:



Git will see that the conflict has been resolved and creates a new merge commit to finalize the merge.

## Git commands that can help resolve merge conflicts

### General tools

git status

The status command is in frequent use when a working with Git and during a merge it will help identify conflicted files.

git log --merge

Passing the --merge argument to the git log command will produce a log with a list of commits that conflict between the merging branches.

git diff

diff helps find differences between states of a repository/files. This is useful in predicting and preventing merge conflicts.

### Tools for when git fails to start a merge

git checkout

checkout can be used for undoing changes to files, or for changing branches

git reset --mixed

reset can be used to undo changes to the working directory and staging area.

### Tools for when git conflicts arise during a merge

git merge --abort

Executing git merge with the --abort option will exit from the merge process and return the branch to the state before the merge began.

git reset

Git reset can be used during a merge conflict to reset conflicted files to a know good state

## Summary

## Merge conflicts can be an intimidating experience. Luckily, Git offers powerful tools to help navigate and resolve conflicts. Git can handle most merges on its own with automatic merging features. A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts will most likely happen when working in a team environment.