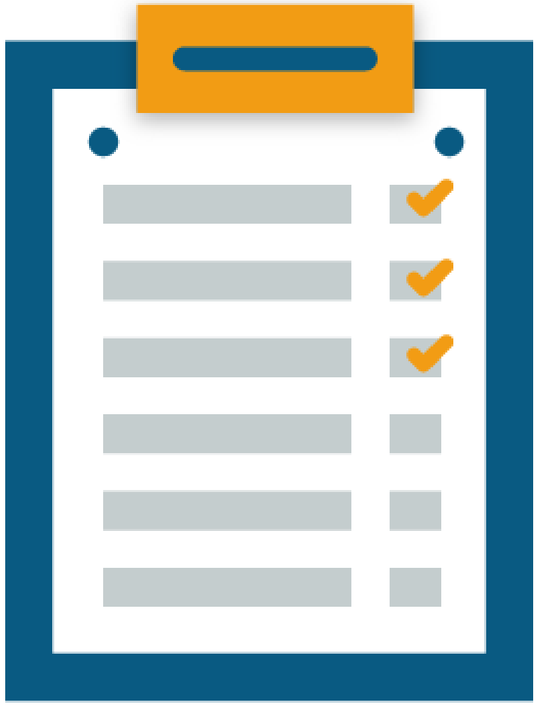
edureka!

Version Control with Git - II

Topics

Following are the topics covered in this module:

- Working with Remote repository
- Branching and merging in Git
- Merge Conflicts
- Stashing, Rebasing, Reverting and Resetting
- Demo: Basics Commands and Operations in Git.
- Demo: Working with Multiple Branches in Git

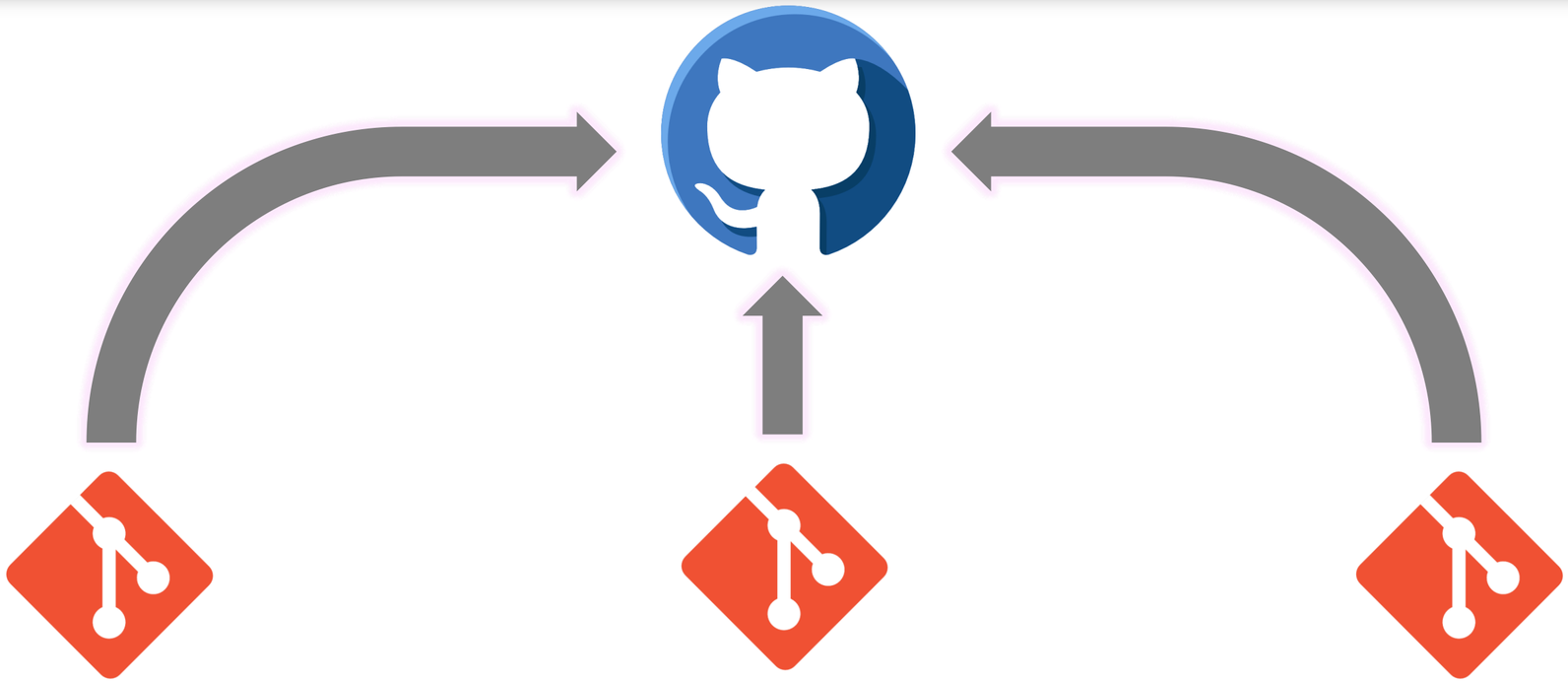


Working With Remote Repositories

edureka!

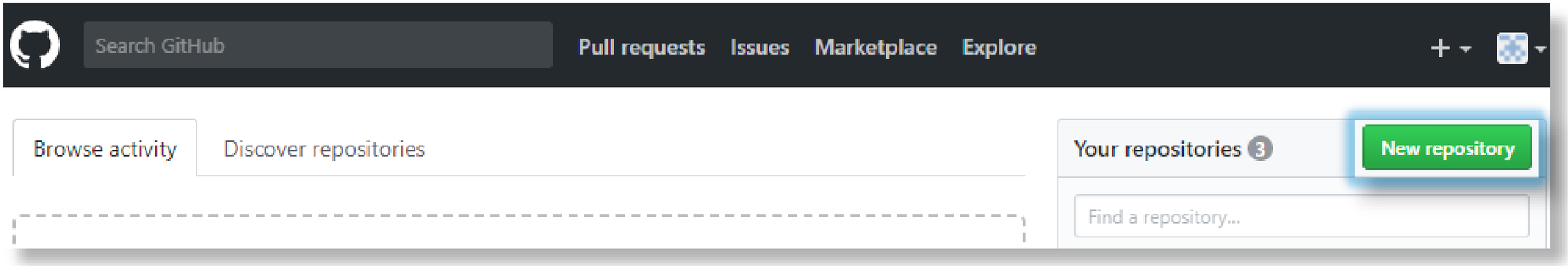
What Is A Remote Repository?

Mostly the users work on a local repository. But in order to collaborate with other people, we use a remote repository. A remote repository is place where the users upload and share their commits with other collaborators.



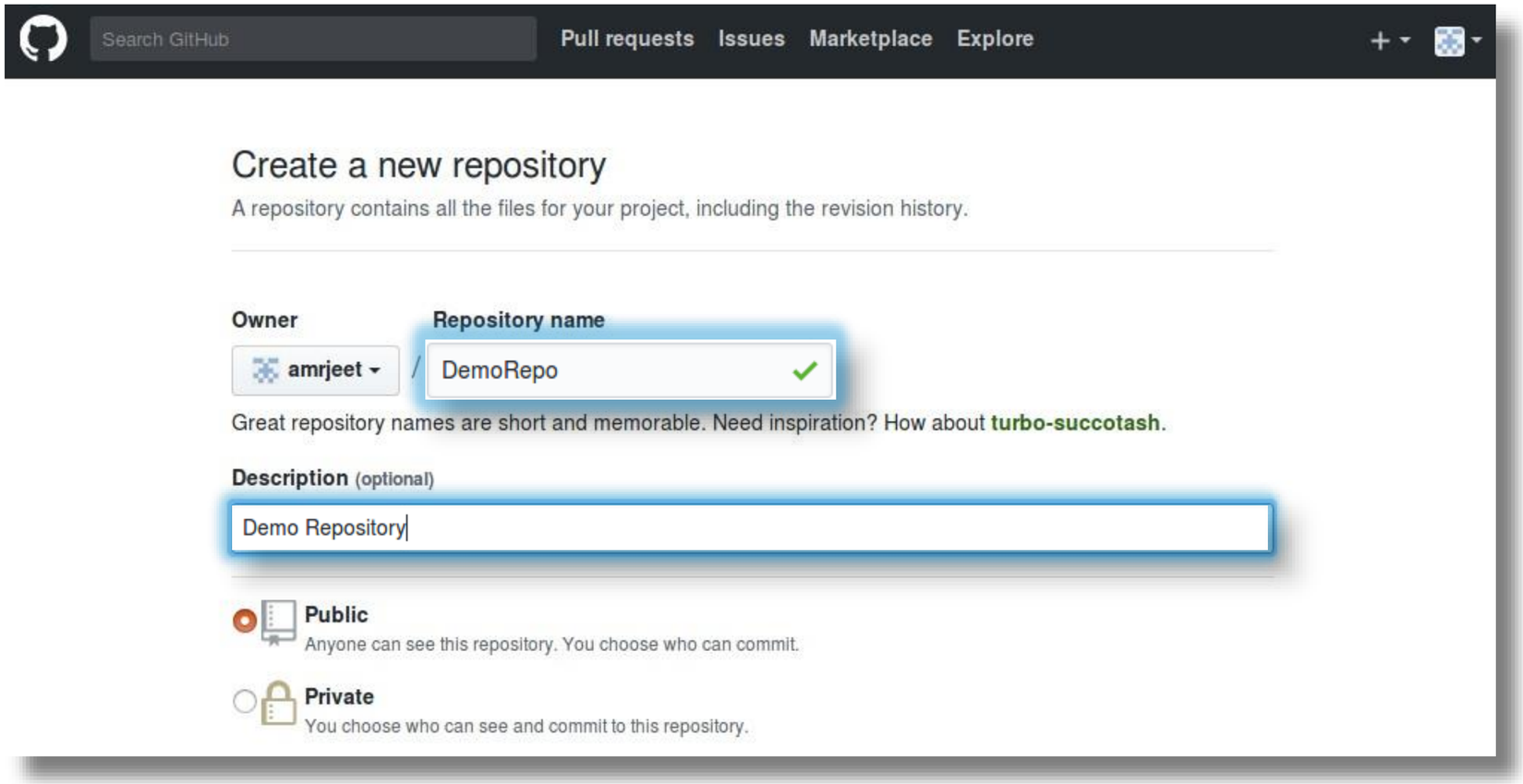
Creating A Remote Repository: New Repository

- Sign-up at **github.com**
- Click on New repository to create a new repository



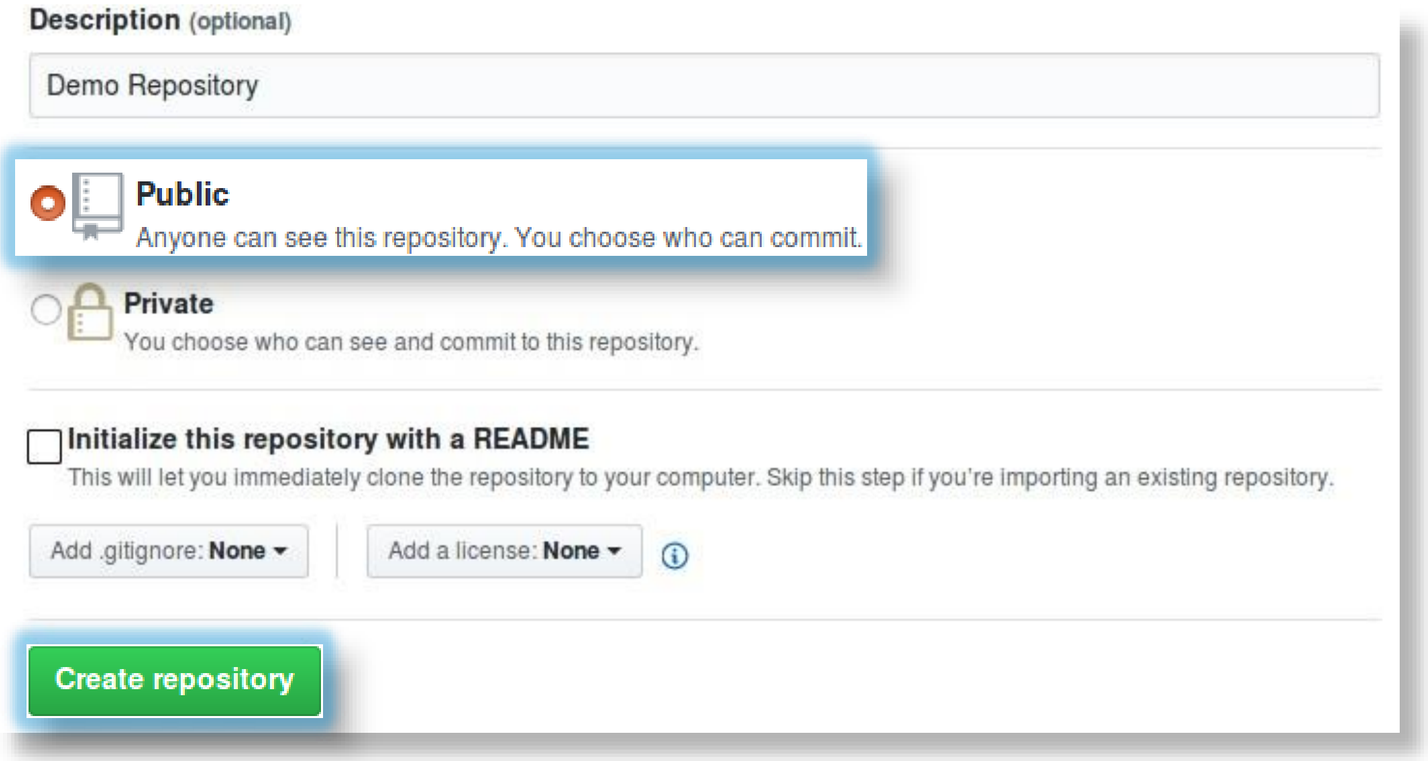
Creating A Remote Repository: Adding Description

- Under **Repository name**, give a name to your repository
- Give some Description about your repository under **Description** section.



Creating A Remote Repository: Create Repository

- For a free repository choose public
- For a private repository a monthly premium needs to be paid
- Finally click on Create Repository

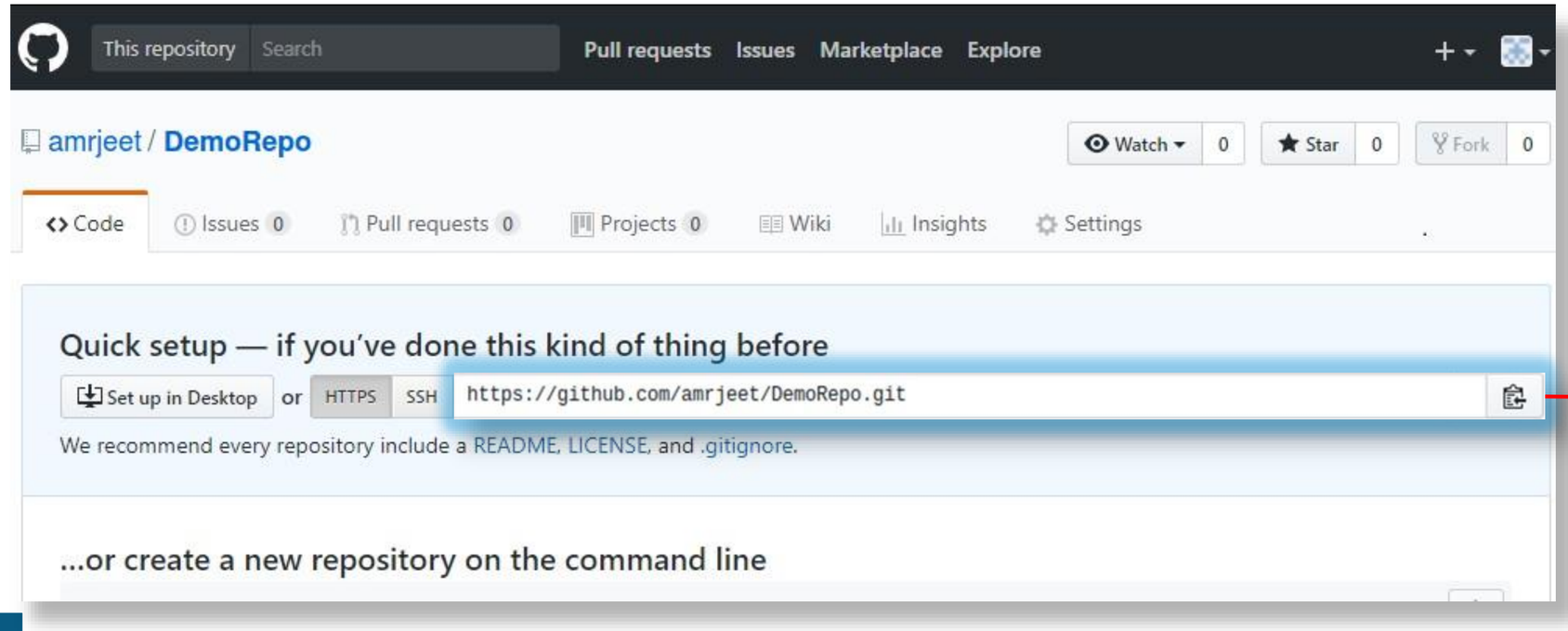


Adding Remote To Local

To add Remote repository to local use **git add remote** followed by remote link

```
Syntax: git add remote origin <remote link>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git remote add origin https://github.com/amrjeet/DemoRepo.git
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```



Push Local Repository To Remote

To push **Local repository** to remote use **push** command

```
Syntax: git push origin master
```

Origin is used as an alias for your remote

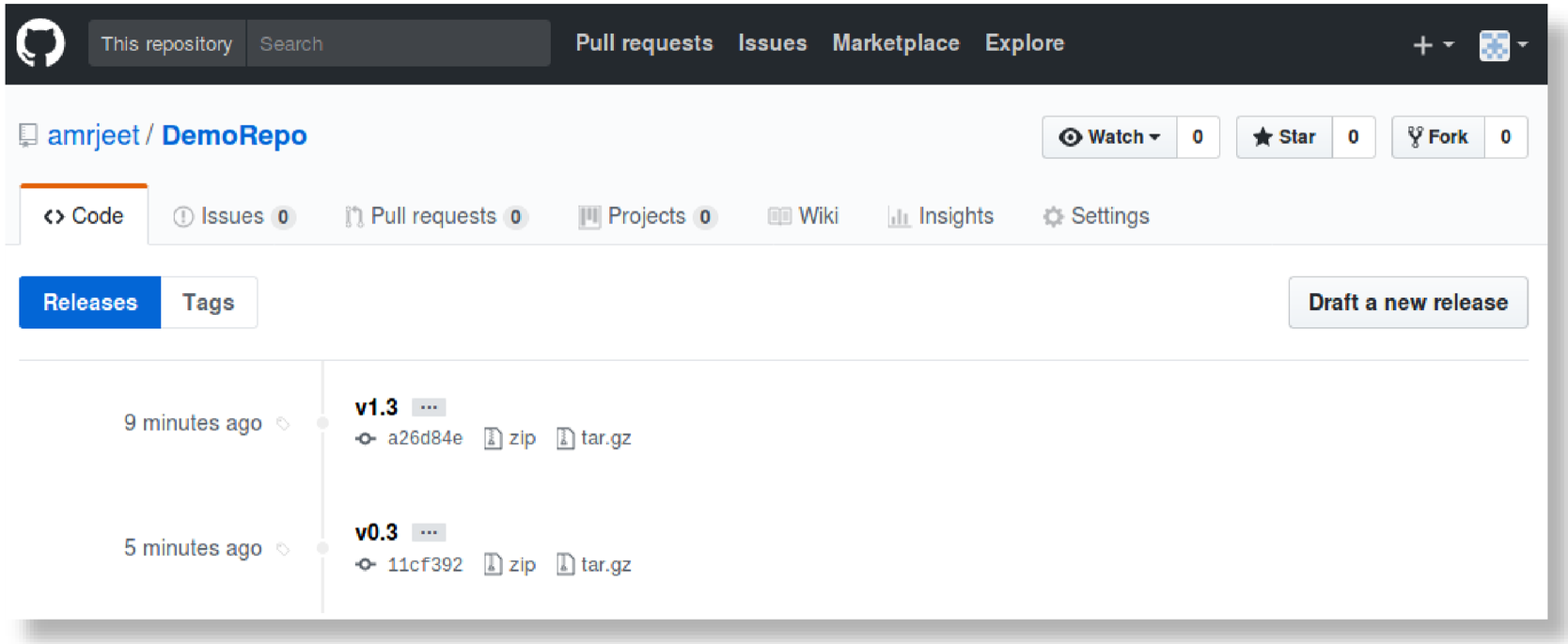
```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git push origin master
Username for 'https://github.com': amrjeet
Password for 'https://amrjeet@github.com':
Counting objects: 11, done.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 1.61 KiB | 0 bytes/s, done.
Total 11 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/amrjeet/DemoRepo.git
 * [new branch]      master -> master
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Refers to master branch in local repo

Pushing Tags

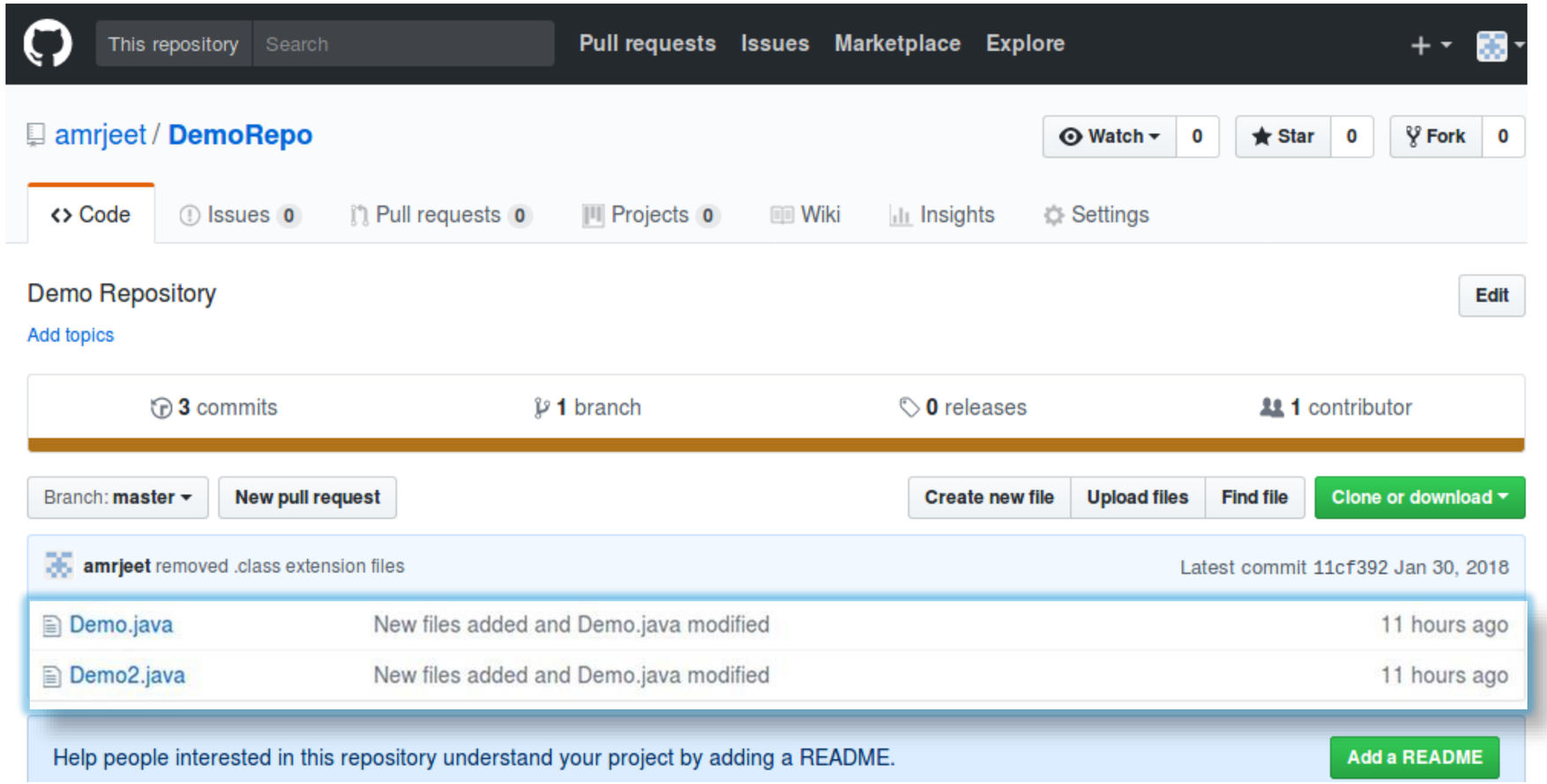
Tags can be pushed, viewed and shared on Remote

```
Syntax: git push origin --tags
```



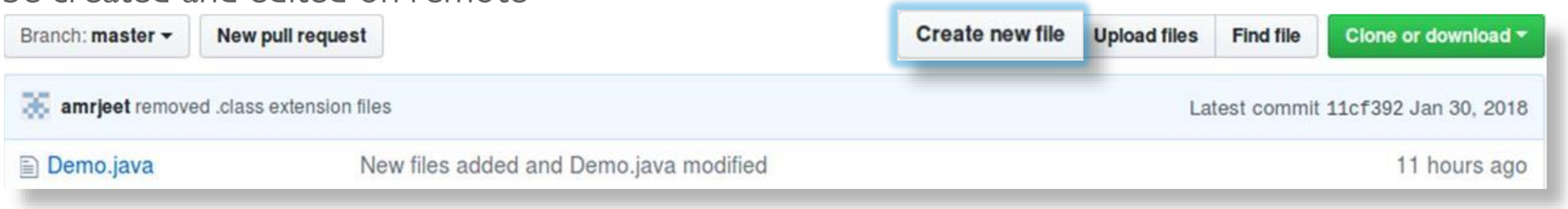
Push Local Repository To Remote

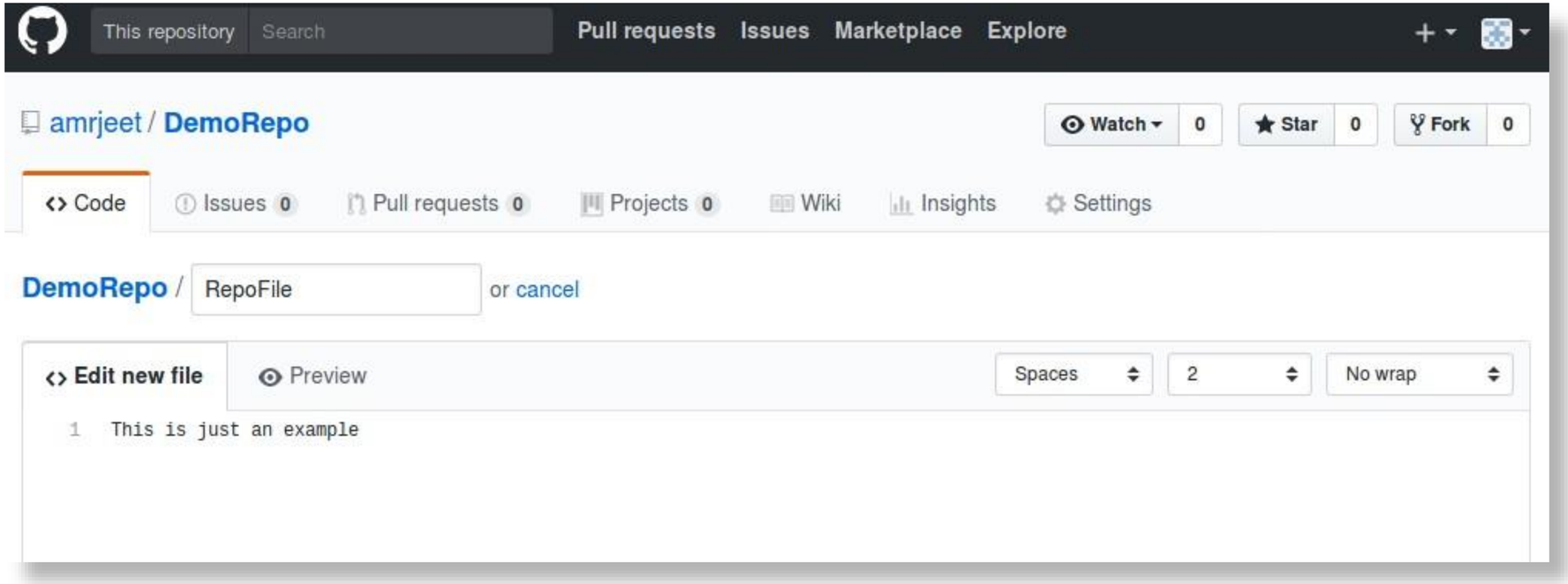
The changes can be seen in Remote repository



Working On Remote

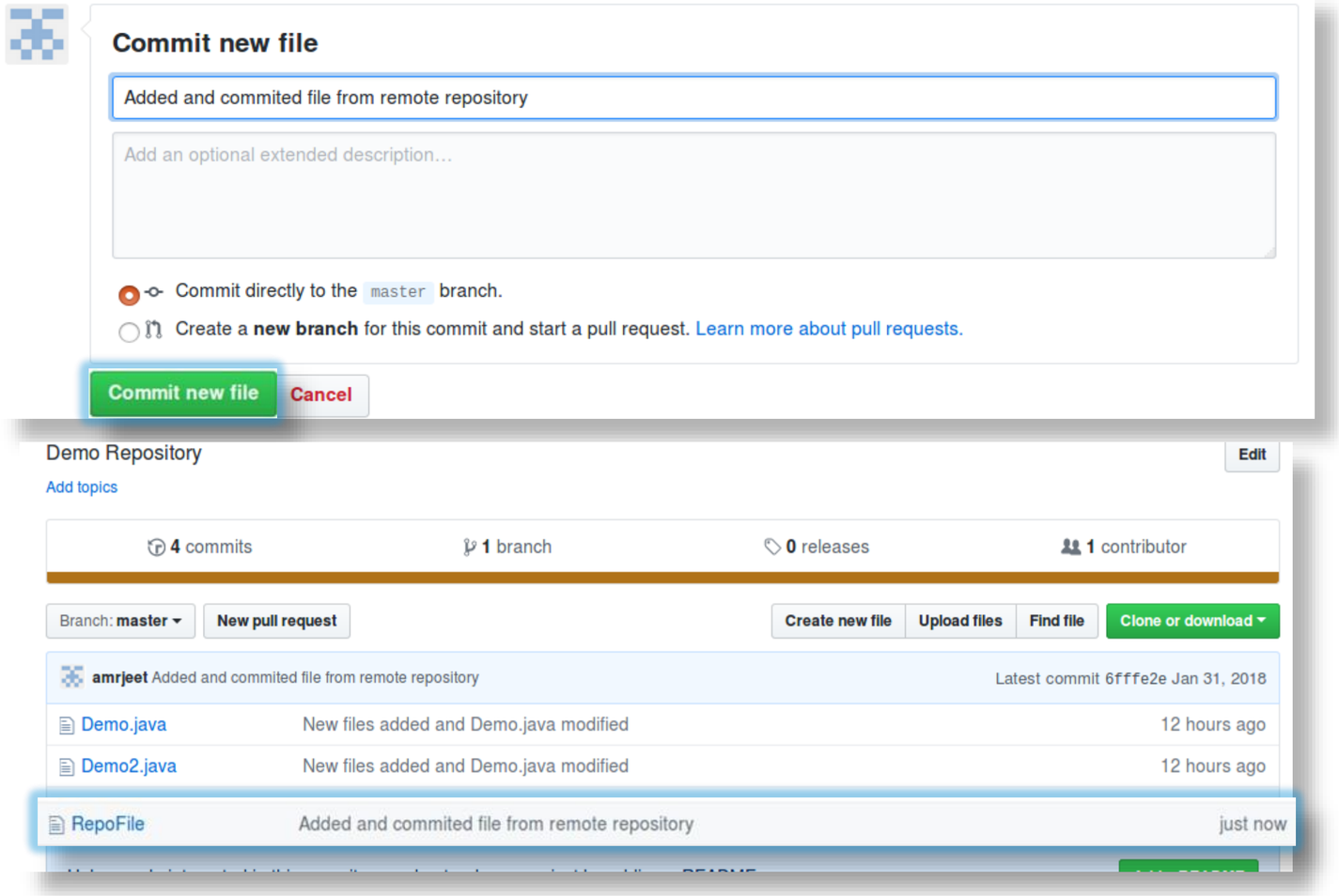
Files can be created and edited on remote





Working On Remote

These files can then be committed on the remote



Remote List

To list all the remotes attached to your Local repository

```
Syntax: git remote -v
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git remote -v
origin  https://github.com/amrjeet/DemoRepo.git (fetch)
origin  https://github.com/amrjeet/DemoRepo.git (push)
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Fetch

- Fetch command copies the changes from remote to local repository

```
Syntax: git fetch origin
```

- Fetch does not affect the present working directory

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git fetch origin
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ ls
Demo2.class  Demo2.java  Demo.class  Demo.java
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Present working directory not affected

Pull

- Pull copies all the changes from remote to local repository
- It then merges the changes with the present working directory

```
Syntax: git pull origin
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git pull origin
From https://github.com/amrjeet/DemoRepo
 * branch            HEAD       -> FETCH_HEAD
Updating 11cf392..6fffe2e
Fast-forward
 RepoFile | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 RepoFile
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ ls
Demo2.class  Demo2.java  Demo.class  Demo.java  RepoFile
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

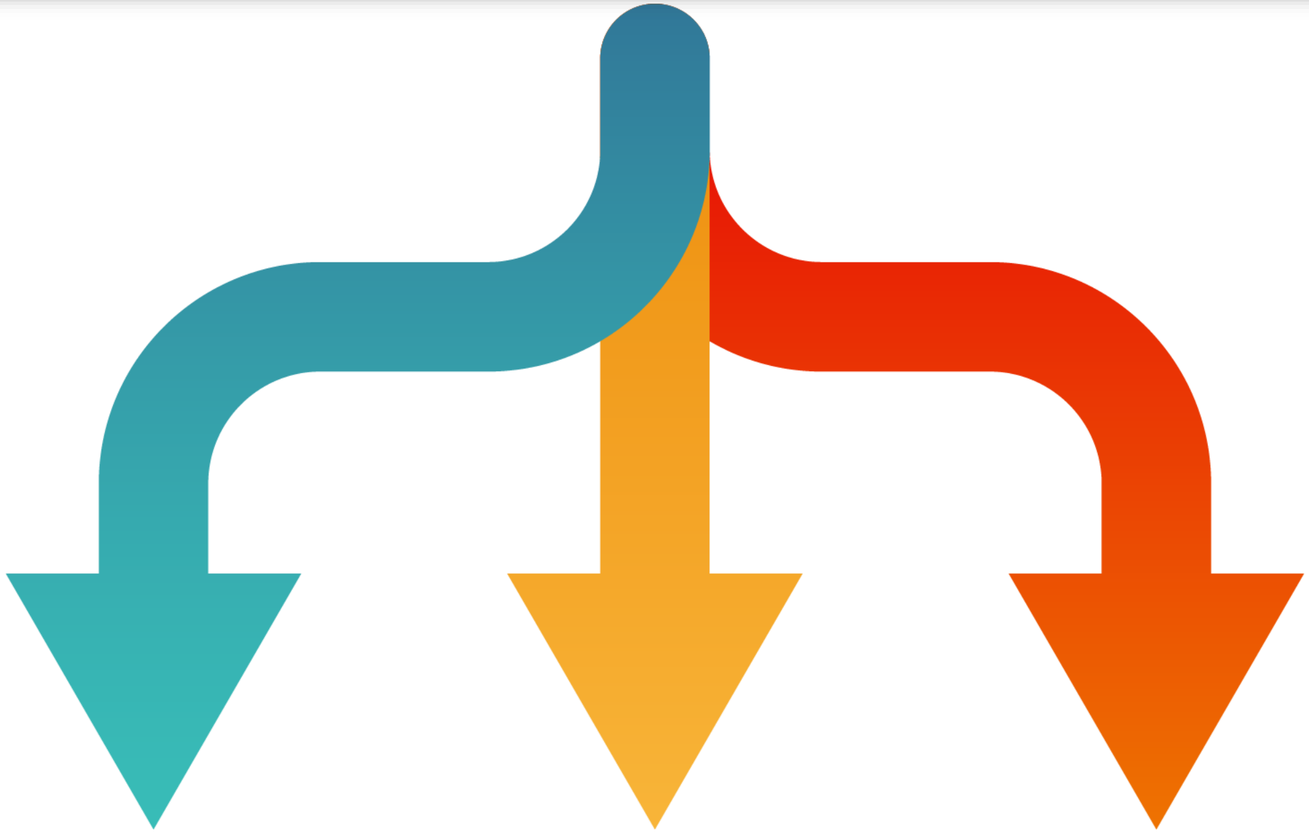
Working directory merged with pulled repository

Branching And Merging

edureka!

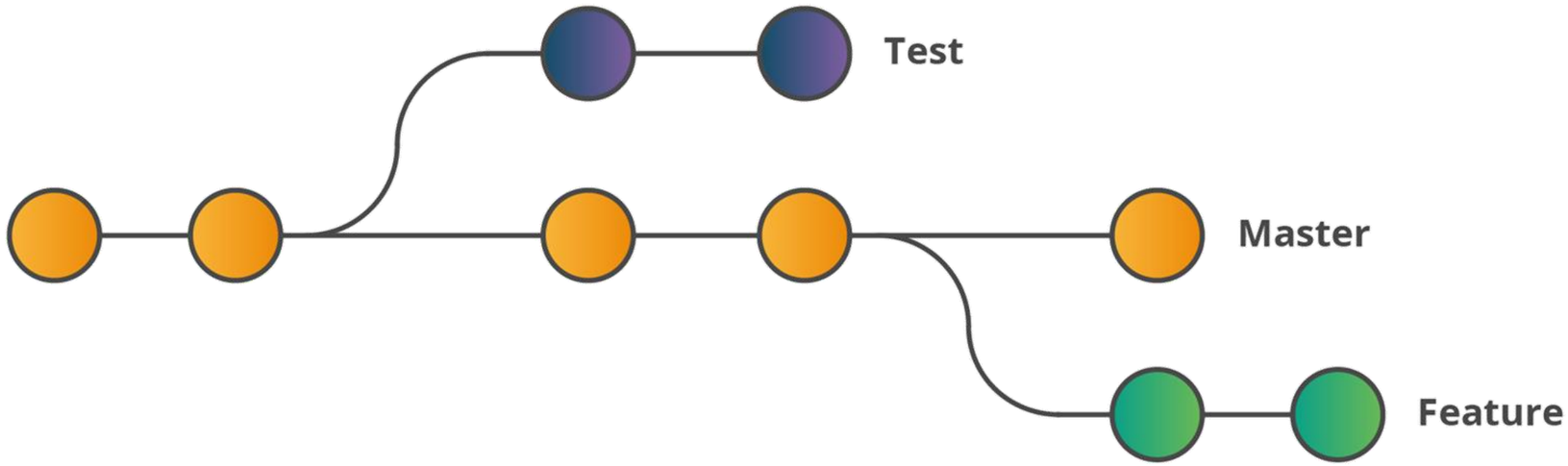
Branching

A project in its development could take multiple different paths to achieve its goal. Branching helps us take these different directions, test them out and in the end achieve the required goal.



Branching In Git

- Branching is an integral part of any Version Control(VC) System
- Unlike other VC's Git **does not** create a copy of existing files for new branch
- It **points** to snapshot of the changes you have made in the system



Creating A Branch

- To create a new branch from your current branch

```
Syntax: git branch <branchname>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git branch feat1
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

- You can then switch to this newly created branch

```
Syntax: git checkout <ranchname>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git checkout feat1
Switched to branch 'feat1'
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Create A Branch

- Creating and switching to a new branch can be done with using -b flag

```
Syntax: git checkout -b <branchname>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git checkout -b feat1
Switched to a new branch 'feat1'
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

- Branch command lists all the branches and also points to the current working branch

```
Syntax: git branch
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git branch
* feat1
  master
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Merging In Git

Merging integrates the changes made in different branches into one single branch

Merging Branches

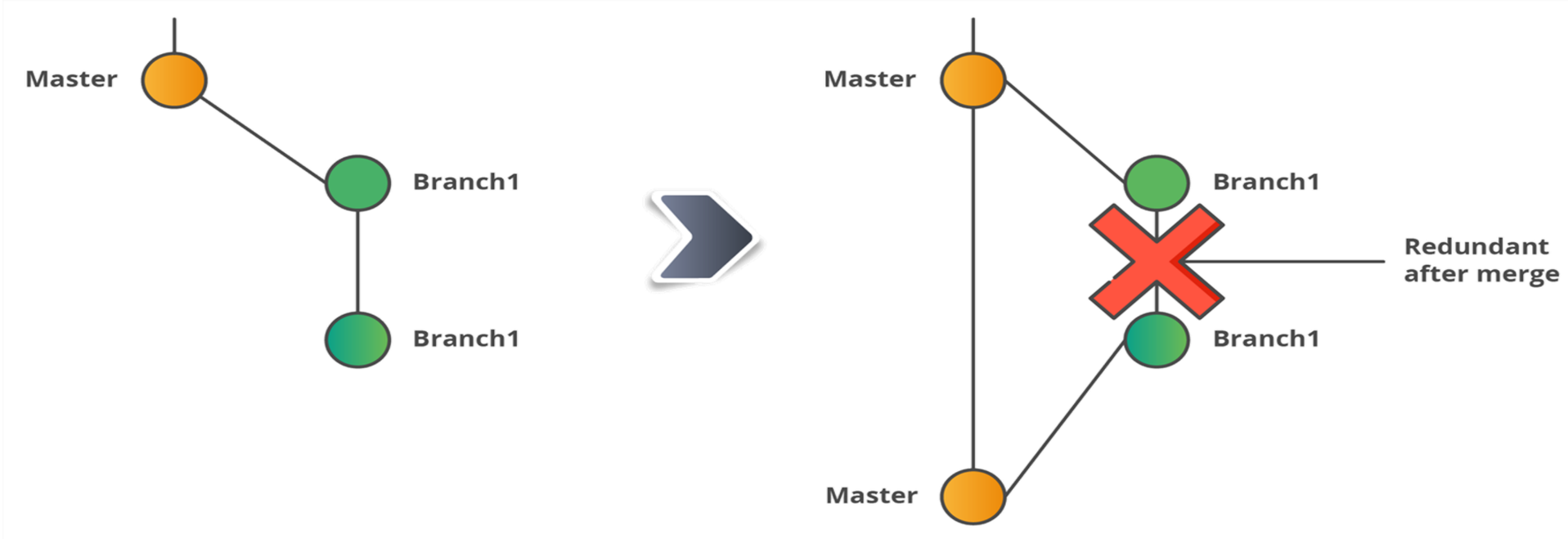
- Different modified branches can be **merged** together using merge

```
Syntax: git merge <branchname>
```

- The branch mentioned is merged into the current branch

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git merge feat1
Updating 96d29b0..0ebb201
Fast-forward
 Demo2.java | 1 +
 1 file changed, 1 insertion(+)
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Merging



Before Merging

After Merging

- All the changes made in **Branch1** after merging are available in the Merged branch(Master)
- Branch1 becomes redundant after merging, hence it can be deleted

Deleting A Branch

- **Merged** branches can be deleted using -**d** flag

```
Syntax: git branch -d <branchname>
```

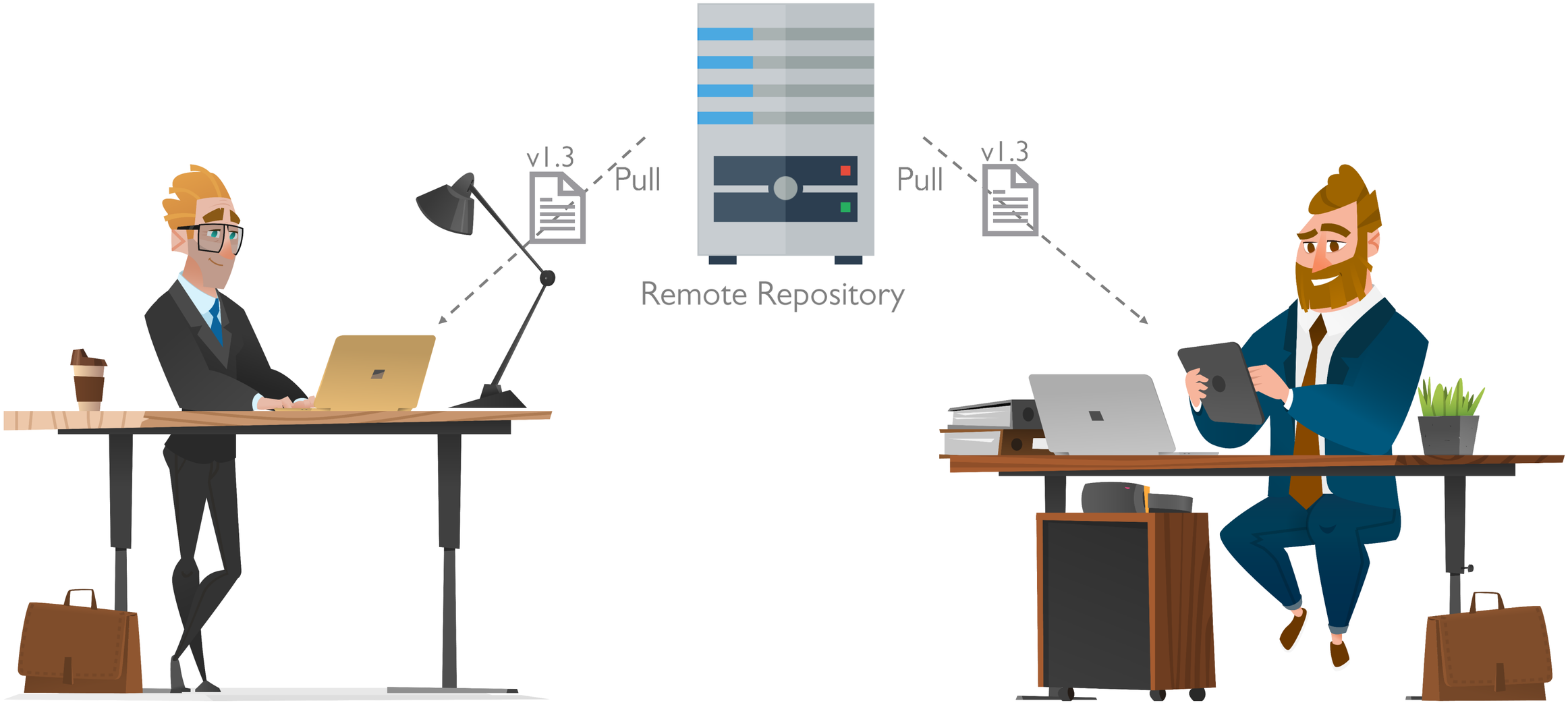
```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git branch -d feat1
Deleted branch feat1 (was 0ebb201).
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

- **Unmerged** branches can be deleted using -D flag

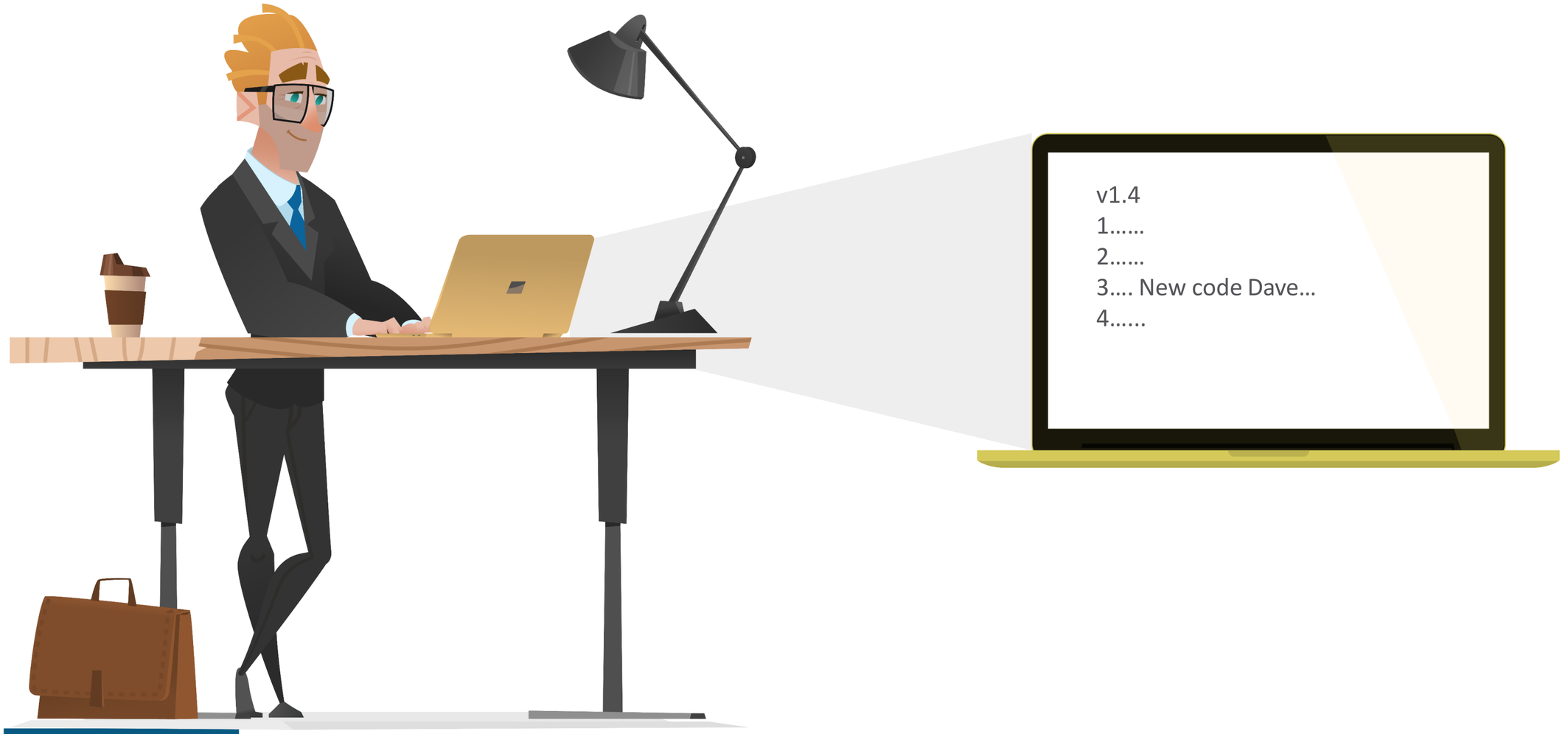
```
Syntax: git branch -D <branchname>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git branch -D feat1
Deleted branch feat1 (was fb0ba99).
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

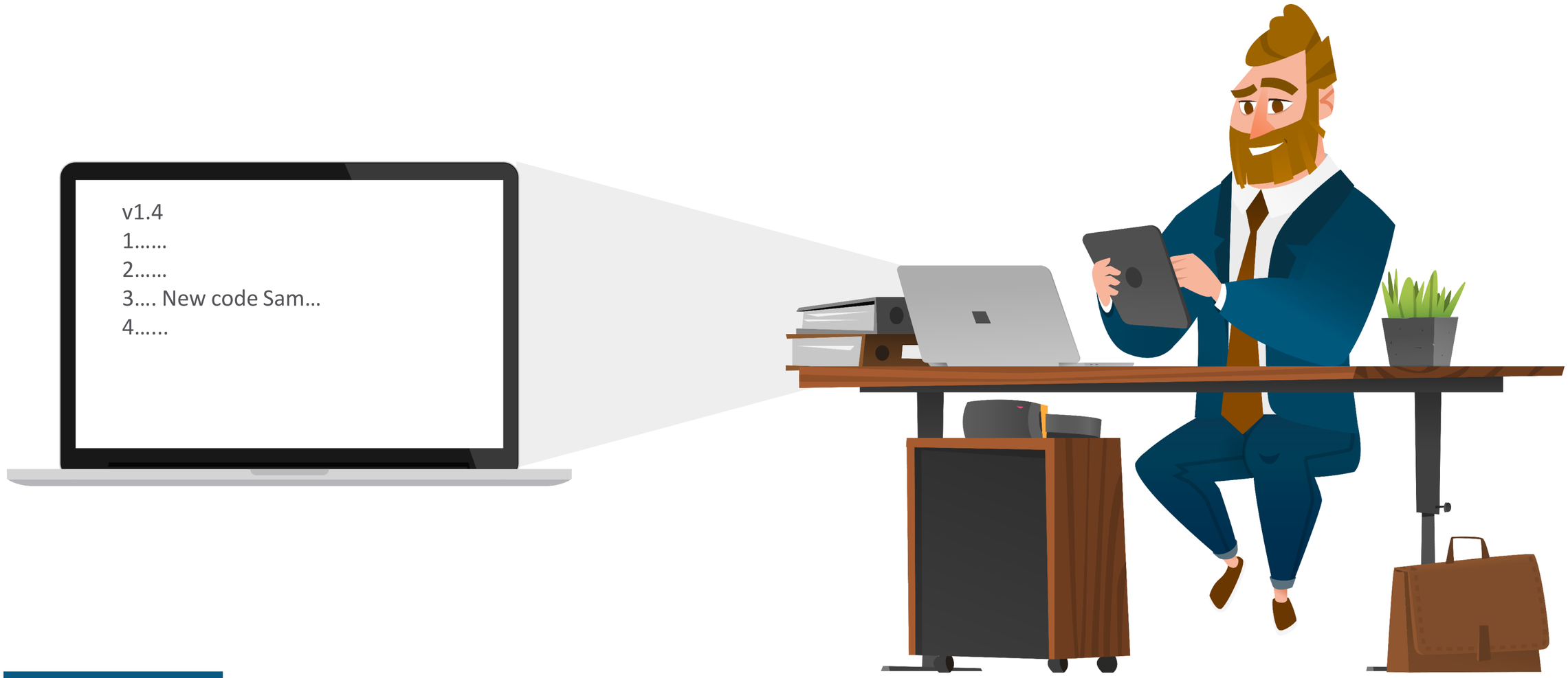
Merge Conflicts



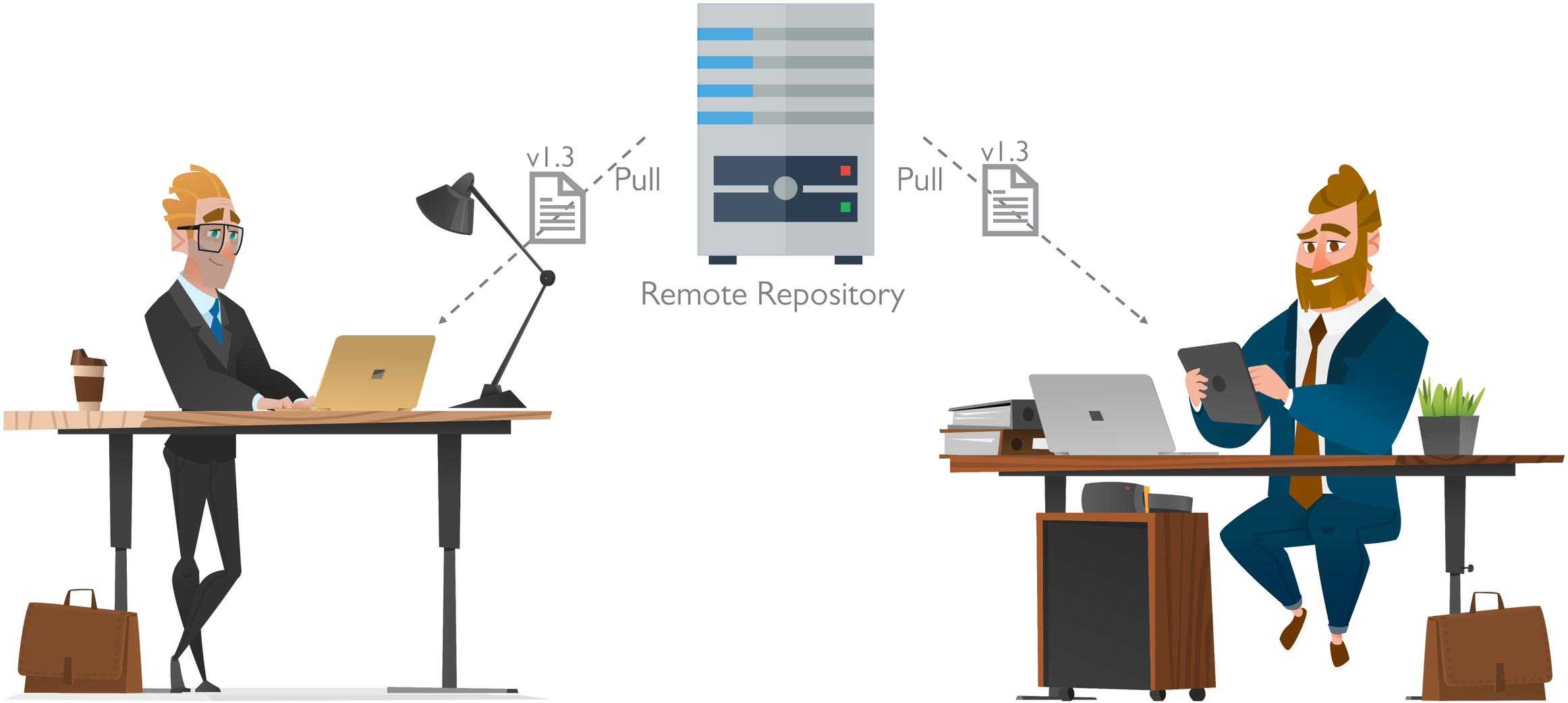
Merge Conflict



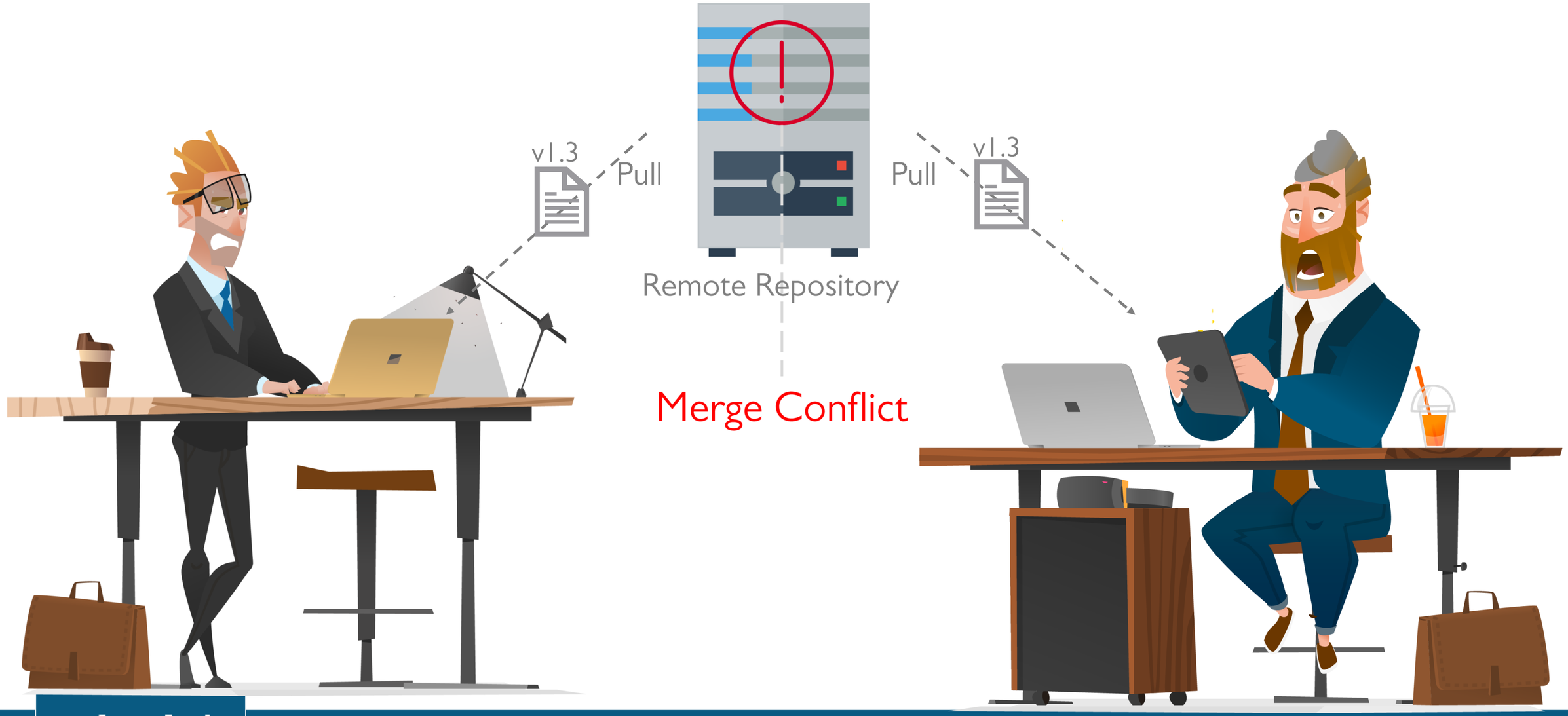
Merge Conflict



Merge Conflicts

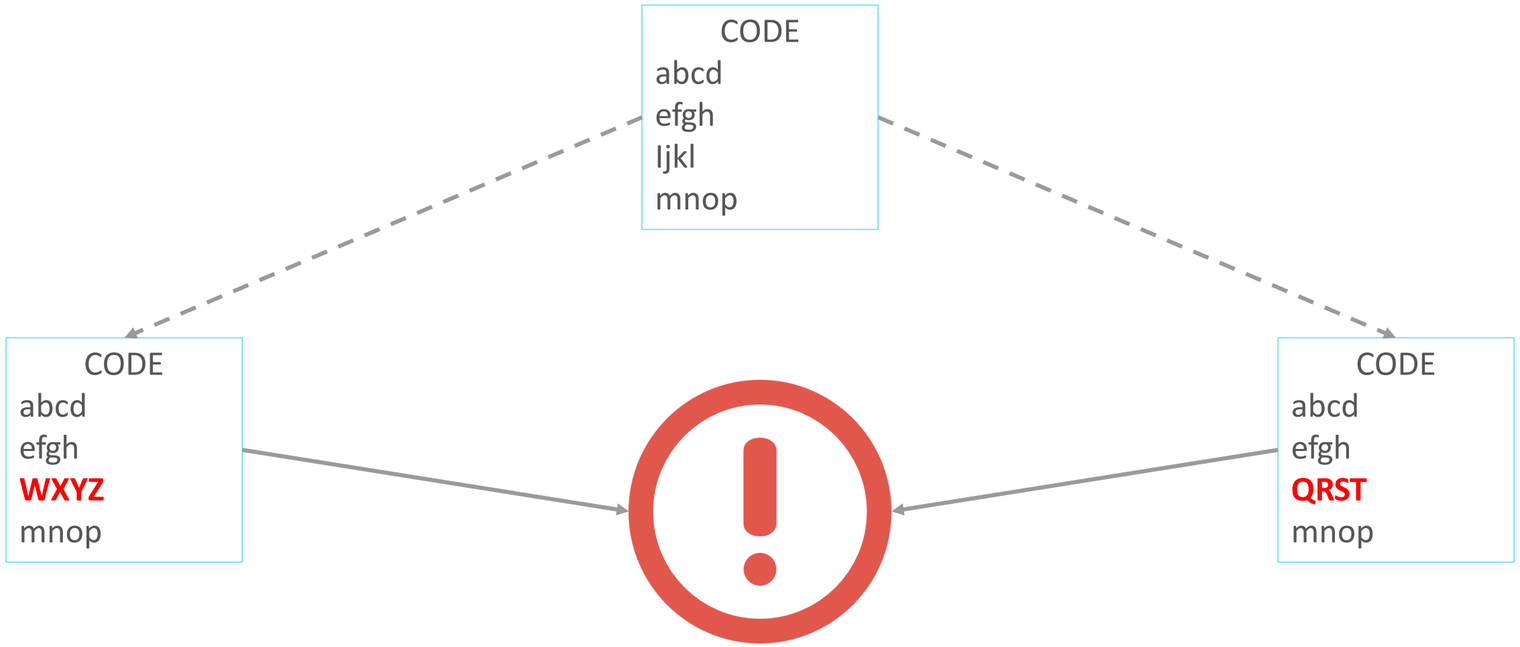


Merge Conflicts



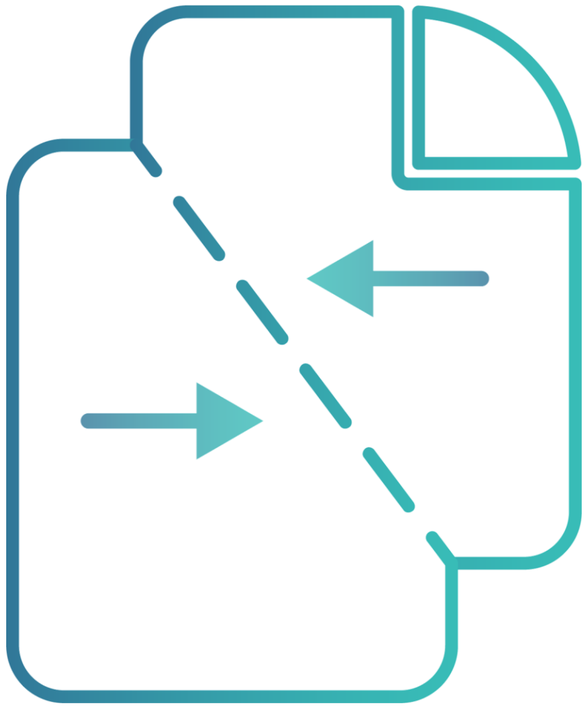
Merge Conflict

- Merge conflicts arise when two files having same content modified are merged
- Merge conflicts can occur on merging branches or when merging forked history together

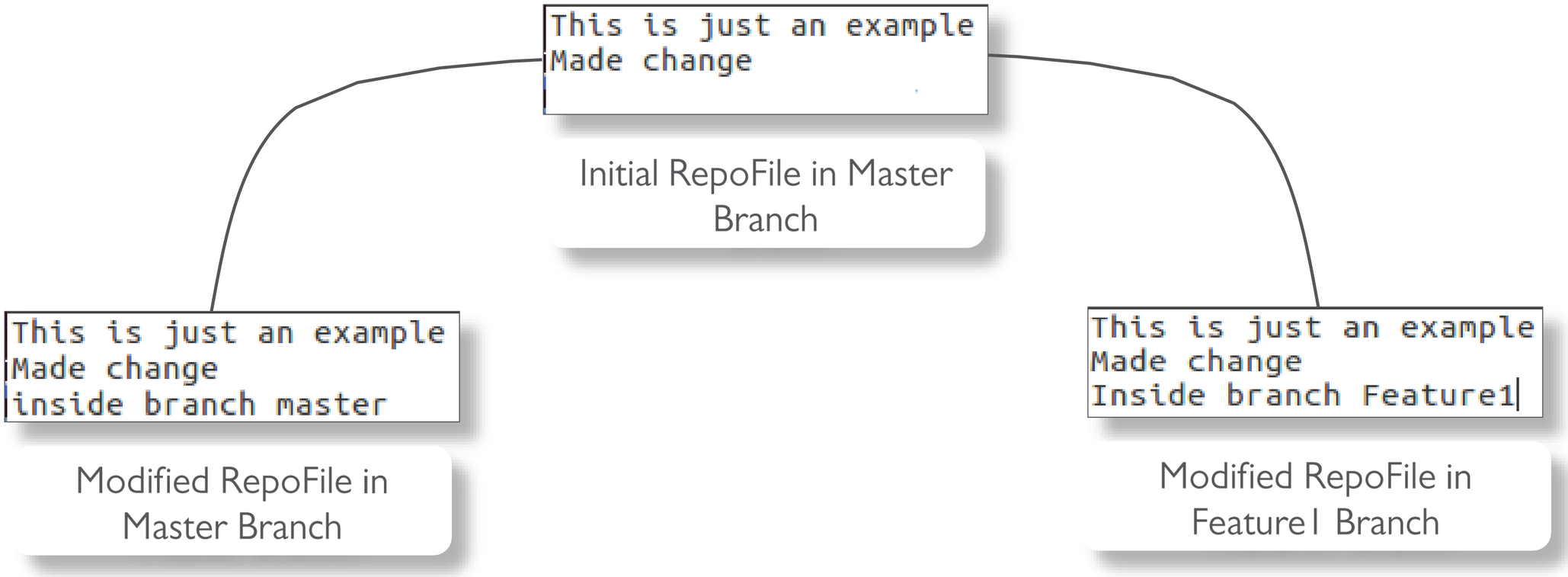


Resolving Merge Conflict

- Merge Conflicts are resolved manually by users
- Git provides different Merge-Tools to compare and choose the required changes
- User can also use third party Merge-Tools with Git



Merge Conflict



Merge Conflict

- Merge Conflict arises on merging branches

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git merge feature1
Auto-merging RepoFile
CONFLICT (content): Merge conflict in RepoFile
Automatic merge failed; fix conflicts and then commit the result.
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

- Set a default merge tool before resolving conflict

```
Syntax: git config --global merge.tool <toolname>
```

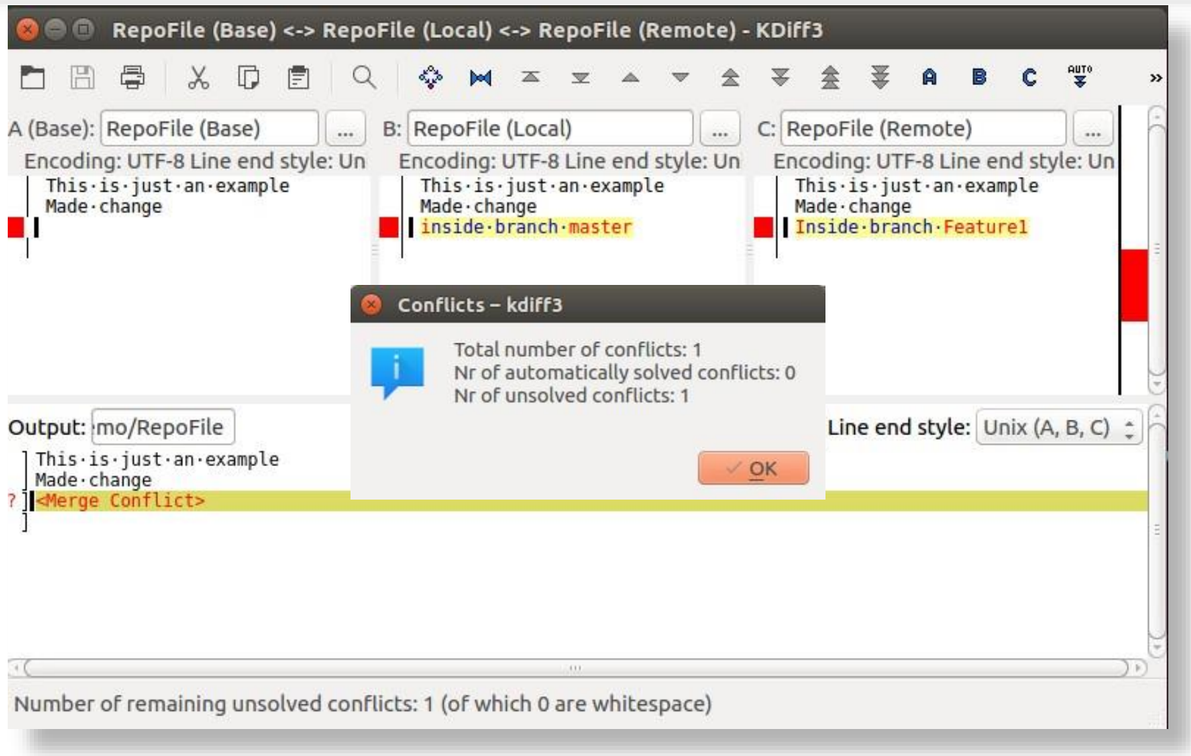
```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git config --global merge.tool kdiff3
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Resolving Merge Conflict

- Merge-tool automatically detects the conflicts and displays them

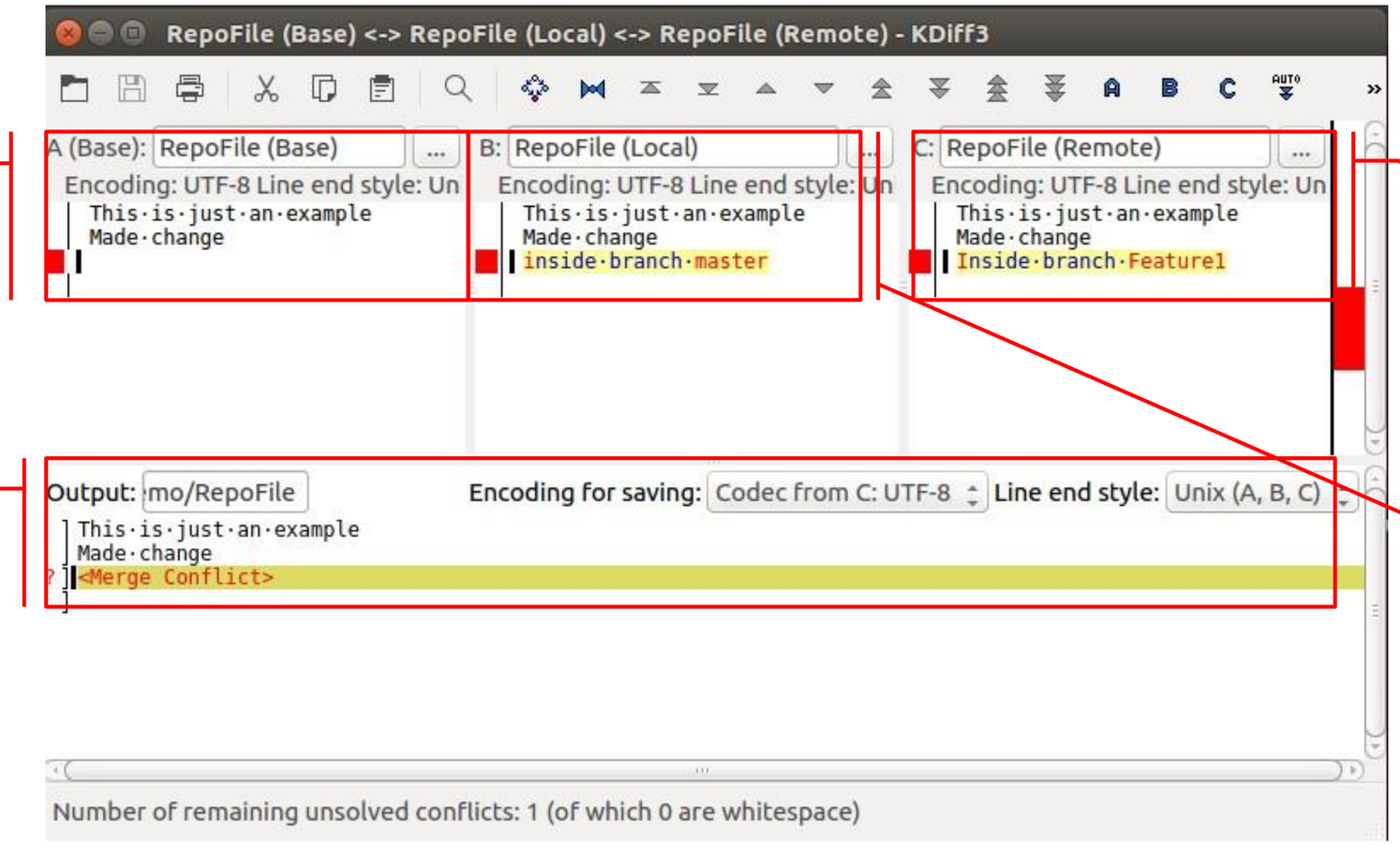
```
Syntax: git mergetool
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git mergetool
```



Resolving Merge Conflict

Initial or base version of RepoFile

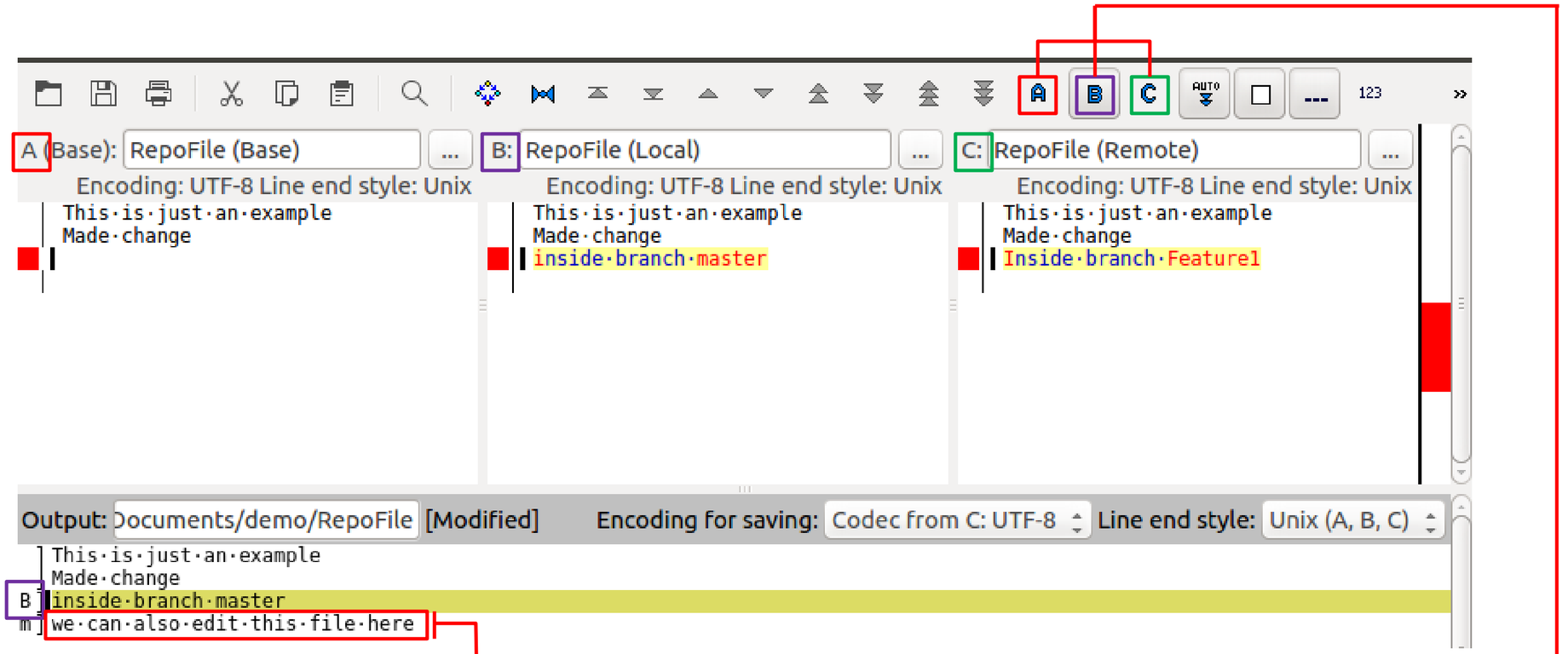


Branch version of RepoFile

Changes to be made in this area

Master branch version of RepoFile

Resolving Merge Conflict



Resolving Merge Conflict

- Commit the resolved changes
- Merge the branch again

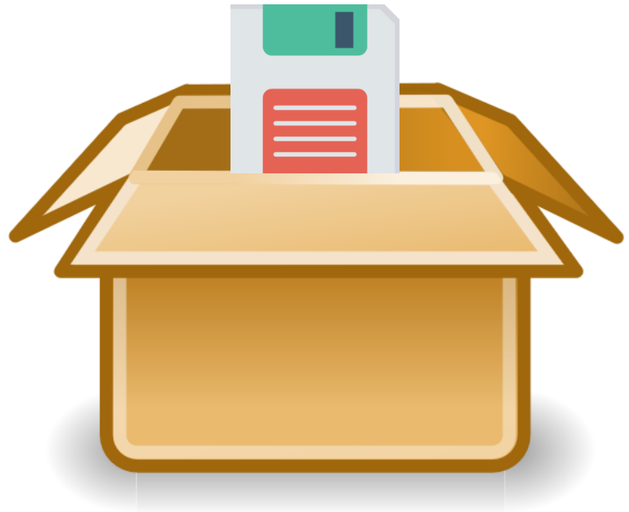
```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git commit -a -m 'resolved merge conflict'
[master b166145] resolved merge conflict
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git merge feature1
Already up-to-date.
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$ git branch --merged
  feature1
* master
  merge
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demo$
```

Git automatically saves the deleted changes in another branch

Stashing

Git Stashing

Git Stashing is a way of creating a checkpoint for non-committed changes. It saves all the changes to a temporary location so the user can perform other tasks such as switching branches, reverting etc. These changes can then be reapplied anywhere.



Stashing

- To Create a stash of your current working directory

```
Syntax: git stash save ‘message’
```

- To list all the saved stashes

```
Syntax: git stash list
```

- Stash list uses a stack structure to save the list

```
amr@amr-VirtualBox:~/Documents/demo$ git checkout master
Switched to branch 'master'
amr@amr-VirtualBox:~/Documents/demo$ git stash list
stash@{0}: On feat1: Saving changes made to the RepoFile
amr@amr-VirtualBox:~/Documents/demo$
```

Stash ID for every stash

Applying A Stash

- Saved stashes can be applied at anytime on any branch

```
Syntax: git stash apply <stash id>
```

- After applying a stash it can still be accessed elsewhere because it remains in the stash

```
amr@amr-VirtualBox:~/Documents/demo$ git stash apply stash@{0}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   RepoFile

no changes added to commit (use "git add" and/or "git commit -a")
amr@amr-VirtualBox:~/Documents/demo$
```

Popping Stash

- Pop command can be used to apply the most recent stash and removing it from the stash stack

```
Syntax: git stash pop
```

```
amr@amr-VirtualBox:~/Documents/demo$ git stash list
stash@{0}: On master: Updated new stash
stash@{1}: On feat1: Saving changes made to the RepoFile
amr@amr-VirtualBox:~/Documents/demo$
```

```
amr@amr-VirtualBox:~/Documents/demo$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   RepoFile

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (4bbae06658e7435e16ea9a29f00c254526c1e788)
amr@amr-VirtualBox:~/Documents/demo$ git stash list
stash@{0}: On feat1: Saving changes made to the RepoFile
amr@amr-VirtualBox:~/Documents/demo$
```

Deleting Stashes

- Stashes can be deleted from the stash list

```
Syntax: git stash drop <stack id>
```

```
amr@amr-VirtualBox:~/Documents/demo$ git stash drop stash@{0}
Dropped stash@{0} (6275777018dcd9bd801d02172bb45feb9538cebf)
amr@amr-VirtualBox:~/Documents/demo$
```

- The entire stack can also be deleted using one command

```
Syntax: git stash clear
```

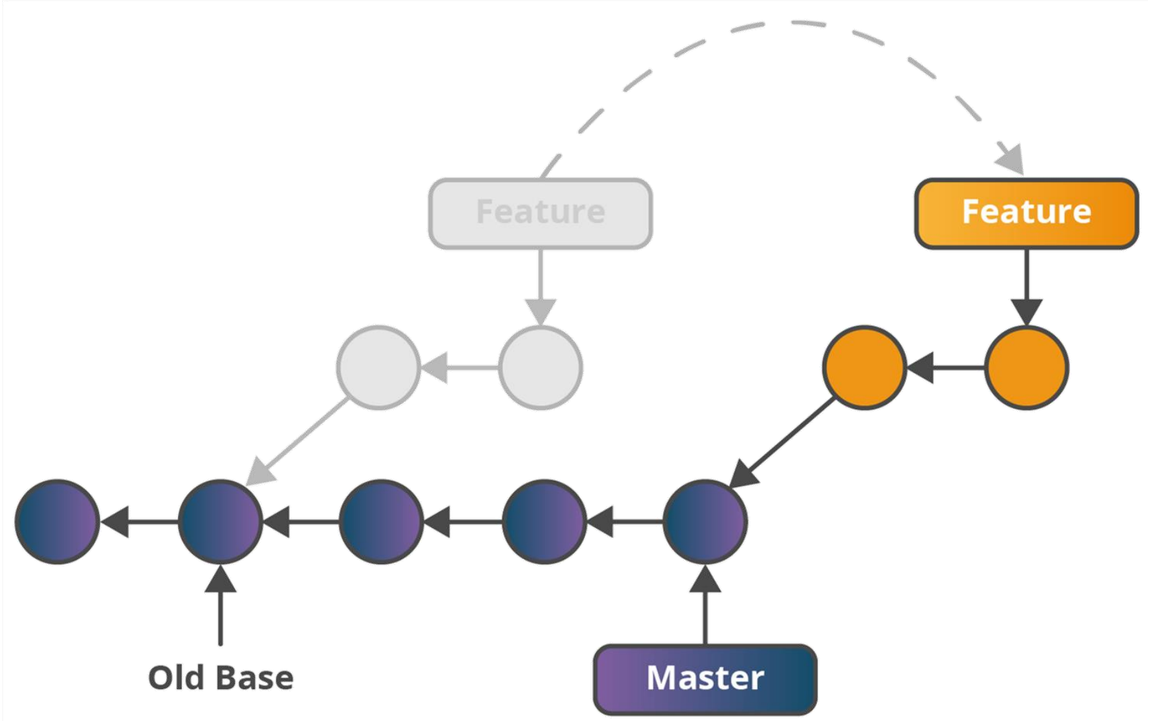
```
amr@amr-VirtualBox:~/Documents/demo$ git stash clear
amr@amr-VirtualBox:~/Documents/demo$
```

Rebasing, Reverting And Resetting

edureka!

Branch Rebasing

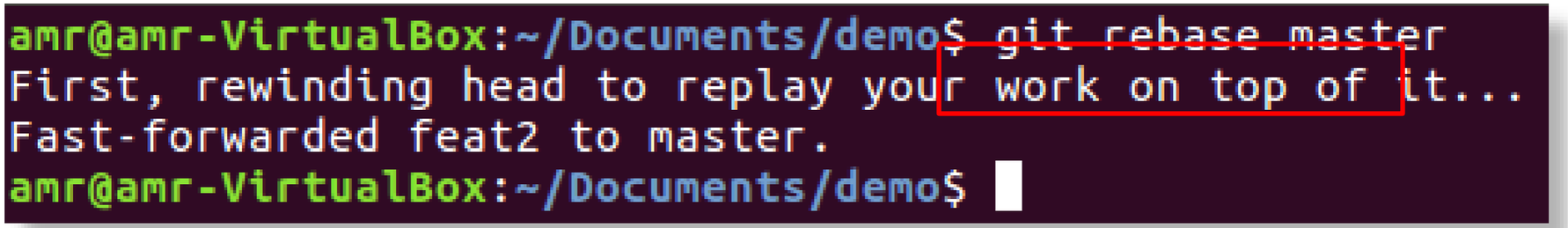
Git rebasing is used, when changes made in one branch needs to be reflected in another branch



Rebasing

To Rebase a branch

```
Syntax: git rebase <rebase branch>
```

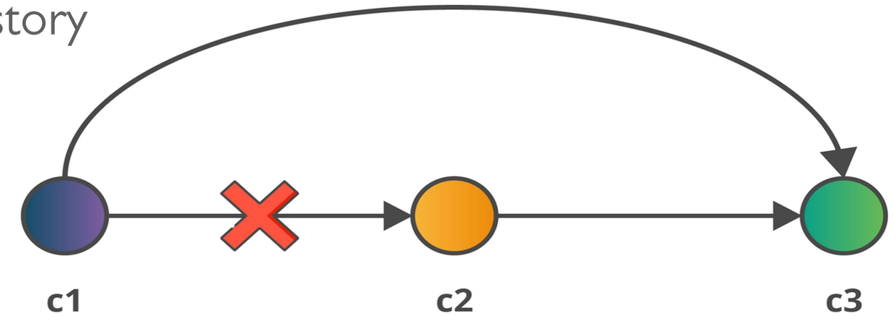


Revert

- Revert or **undo** the changes made in the previous commit
- New commit is created without the changes made in the other commit

```
Syntax: git revert <commit id>
```

- Old commit still resides in the history



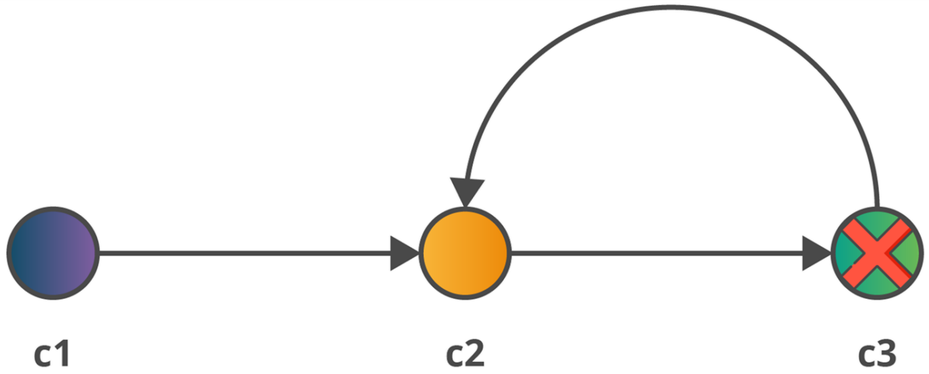
HEAD refers to the latest commit

```
amr@amr-VirtualBox:~/Documents/demo$ git revert HEAD
[feat2 a50d898] Revert "revert edit"
 1 file changed, 1 deletion(-)
amr@amr-VirtualBox:~/Documents/demo$
```

Reset

- Reset command can be used to undo changes at different levels
- Modifiers like --hard, --soft and --mixed can be used to decide the degree to which to reset

```
Syntax: git reset <modifier> <commit id>
```




```
amr@amr-VirtualBox:~/Documents/demo$ git reset v1.5
Unstaged changes after reset:
M       RepoFile
amr@amr-VirtualBox:~/Documents/demo$
```

Thank You

For more information please visit our website
www.edureka.co