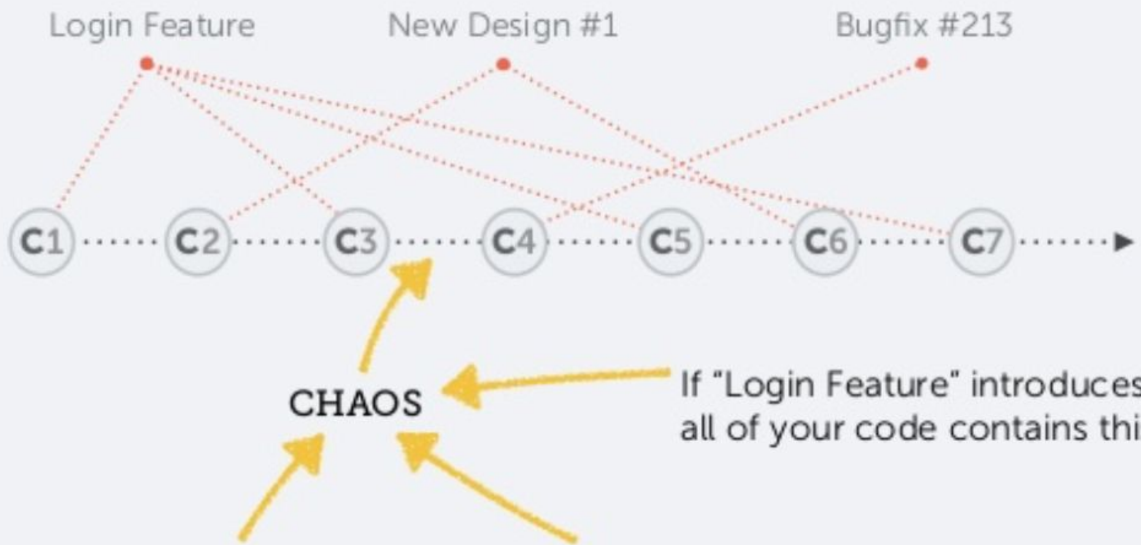
Hacktober FEST 2019

presented by DigitalOcean and DEV

Topics for today’s session:

- Git Branches
- Cherry Picking
- Hashes and Git Ref
- Merging
- Resolving Merge conflicts
- Git rebase
- Git revert

The World Without Branches



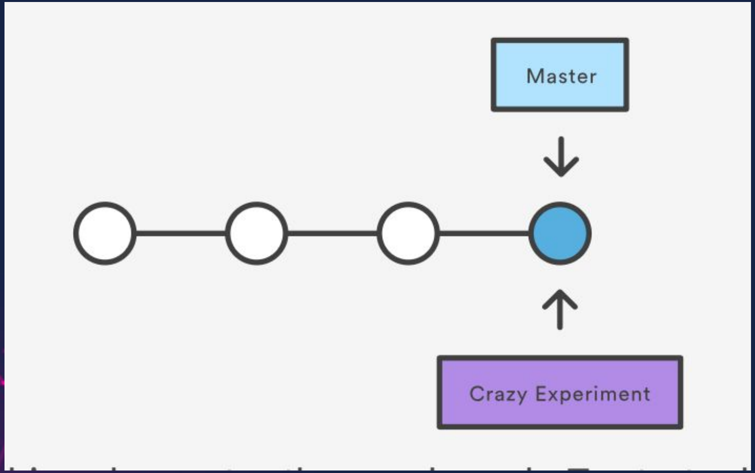
If "Login Feature" introduces a bug, all of your code contains this bug!

What if you need to release "Login Feature"? It already contains "New Design #1" - which you maybe don't want to release...

What happens if your client doesn't like "New Design #1"? How do you get (only) that code out?

Git Branches

- In Git, you are always working on a branch.
- At any time, you can "check out" only one branch :P
- Git branch does not affect a repository, it just creates a pointer to a commit!



Hands on session

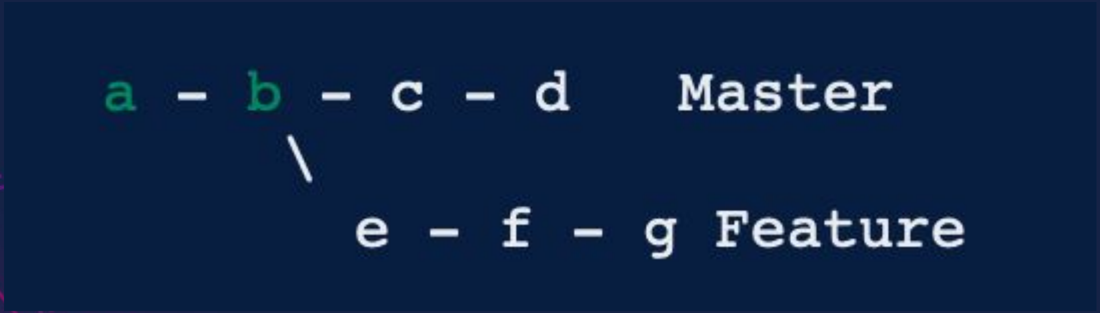
- https://github.com/anumehaagrawal/Hacktoberfest-test-repo - Fork this repository and then clone it.
- Create two branches - test & test 1
- Create two files, one in each branch
- git log --graph --simplify-by-decoration --all - List the commits and pointers to the commit

Git commands

- git branch - Lists all the branches
- git checkout -b <branch-name> - Creates a new branch
- git branch -m <branch> - Renames a branch
- git checkout <branch> - Changes HEAD pointer to this branch
- git branch -d <branch> - Deletes a branch

Git cherry picking

- Cherry picking is the act of picking a commit from a branch and applying it to another.
- Useful for team collaboration.
- git cherry-pick commitHashOfg



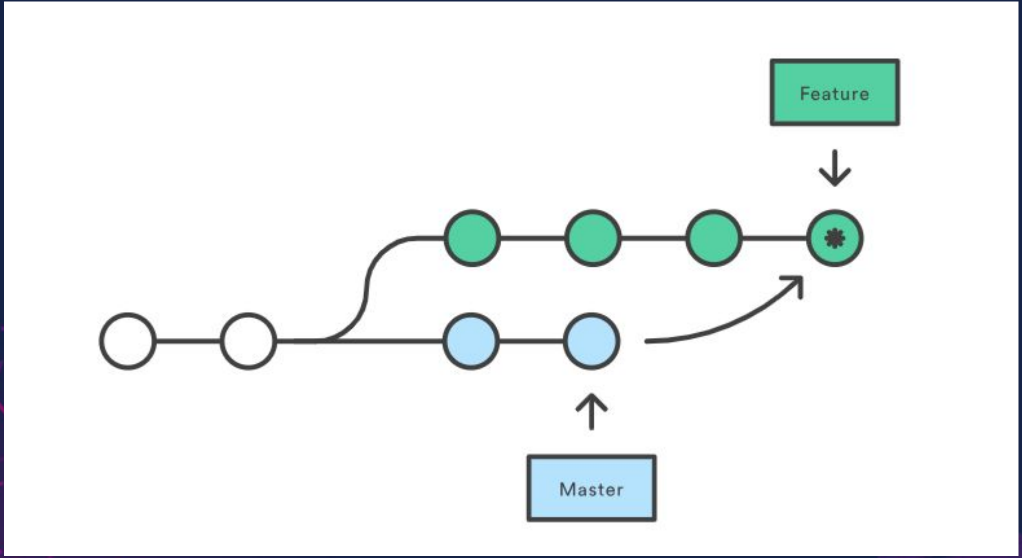
```
a - b - c - d   Master
     \
      e - f - g Feature
```

Hashes and git ref

- The most direct way to reference a commit is via its SHA-1 hash.
- You can fetch information for a commit by just a few characters - git show 2bf669
- git rev-parse <branch-name>
- refs - user-friendly alias for a commit hash
- Refs are stored as normal text files in the .git/refs directory,

Git Merge Command

Git merge is used to merge two different branches into one.



Git Merge Syntax

Two ways to do it:

First way

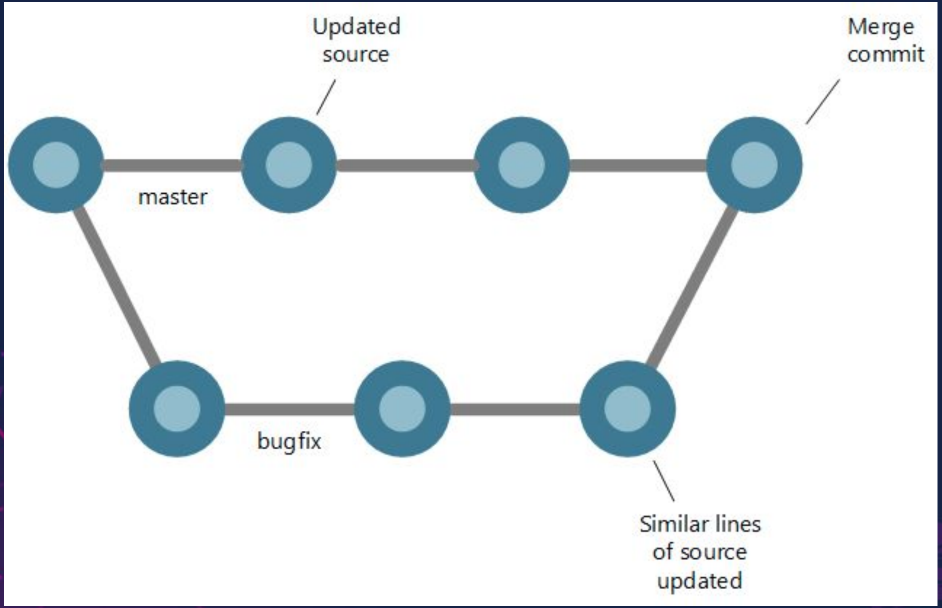
- Goto the base branch using `git checkout base`
- Run `git merge branch_to_merge`

Second way

- Condense the two commands into one,
  `git merge base_branch branch_to_merge`

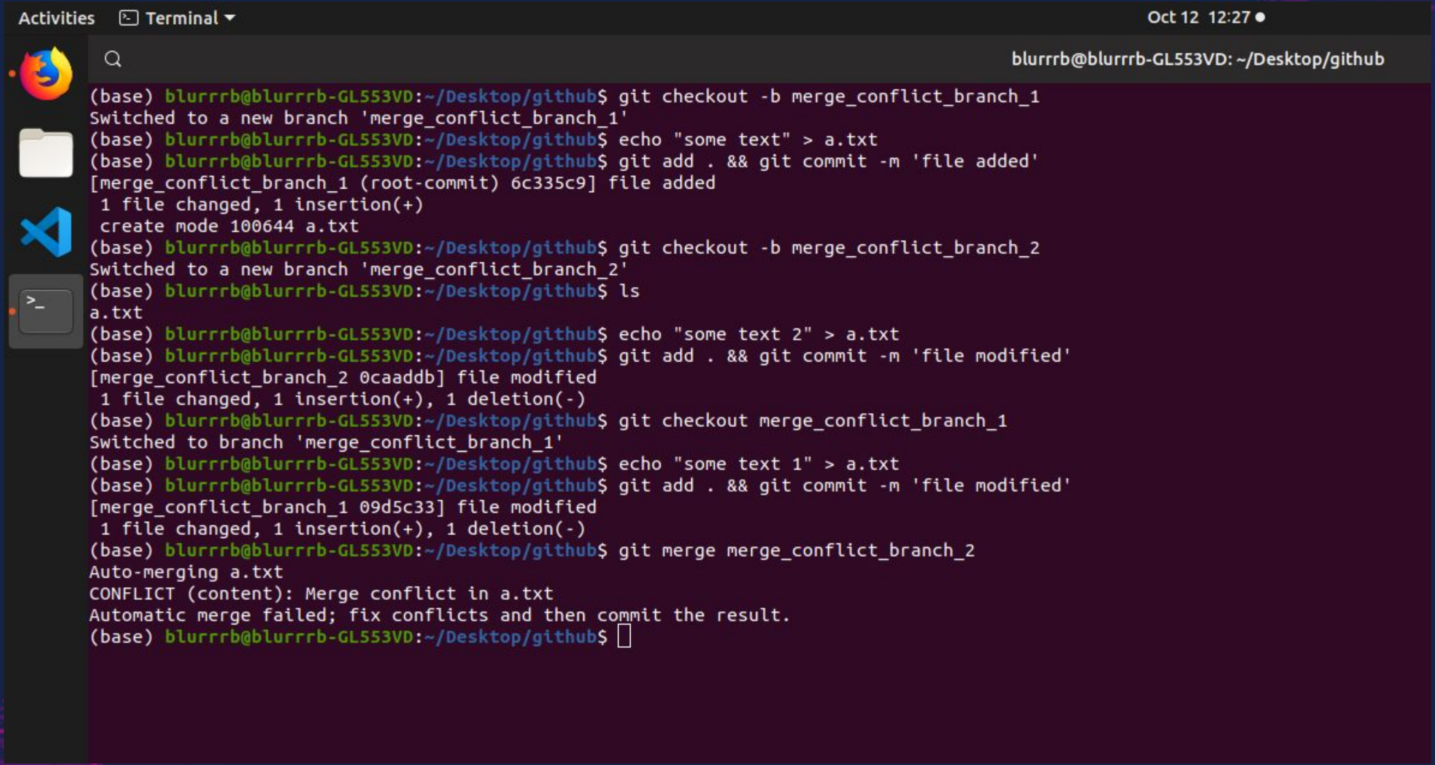
Merge Conflicts

- Two branches, same file but different modifications



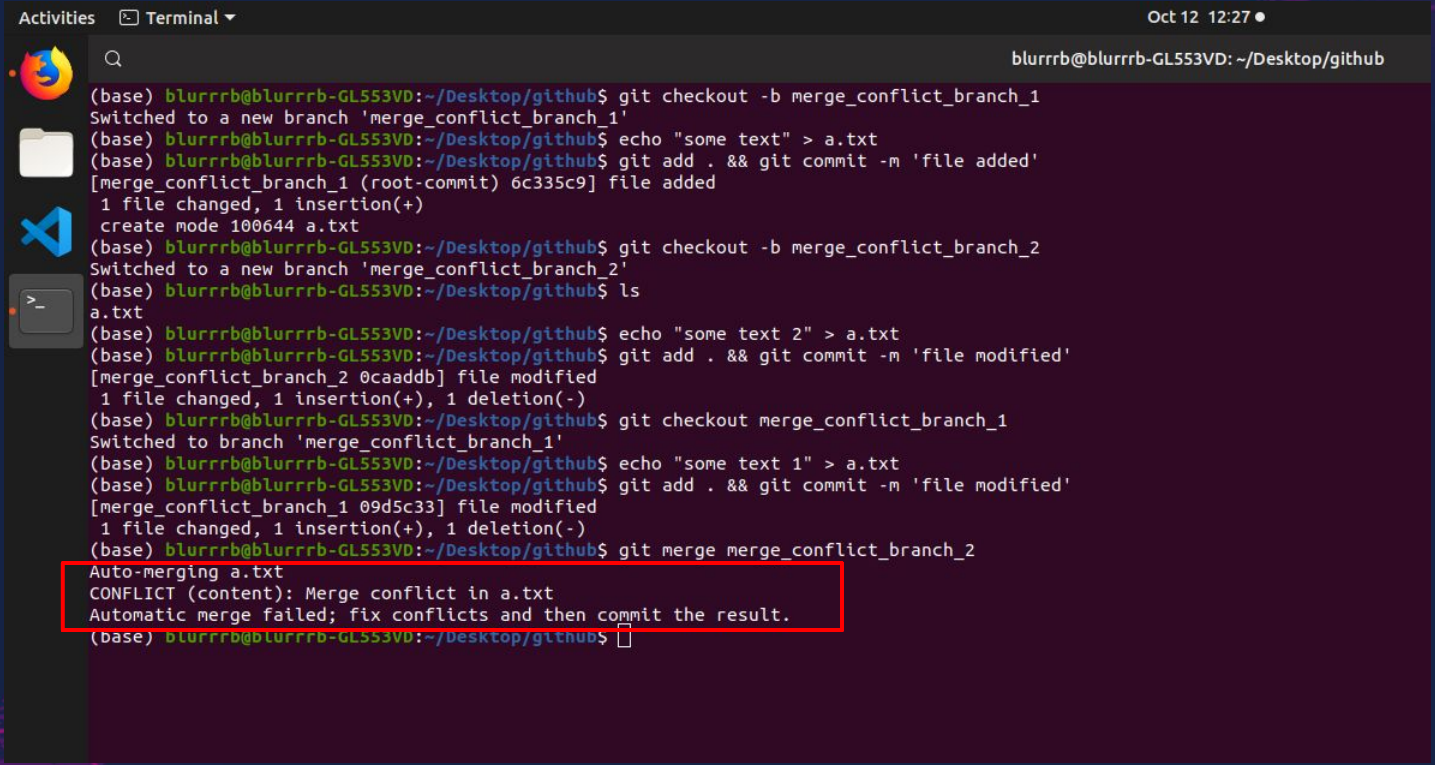
Lets create a merge conflict!

- git checkout -b merge_conflict_branch_1 - create a branch
- echo “some text” > a.txt - create a file
- git add . && git commit -m ‘file added’ - stage changes and commit
- git checkout -b merge_conflict_branch_2 - create another branch
- edit first line of a.txt to “some text 2”
- git add . && git commit -m ‘file modified’ - stage changes and commit
- git checkout merge_conflict_branch_1 - change branch
- edit first line of a.txt to “some text 1”
- git add . && git commit -m ‘file modified’ - stage changes and commit
- git merge merge_conflict_branch_2 - merge merge_conflict_branch_2 into merge_conflict_branch_1



```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout -b merge_conflict_branch_1
Switched to a new branch 'merge_conflict_branch_1'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file added'
[merge_conflict_branch_1 (root-commit) 6c335c9] file added
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout -b merge_conflict_branch_2
Switched to a new branch 'merge_conflict_branch_2'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ ls
a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text 2" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file modified'
[merge_conflict_branch_2 0caaddb] file modified
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout merge_conflict_branch_1
Switched to branch 'merge_conflict_branch_1'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text 1" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file modified'
[merge_conflict_branch_1 09d5c33] file modified
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git merge merge_conflict_branch_2
Auto-merging a.txt
CONFLICT (content): Merge conflict in a.txt
Automatic merge failed; fix conflicts and then commit the result.
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ 
```

Hacktoberfest 2019
presented by DigitalOcean and DEV

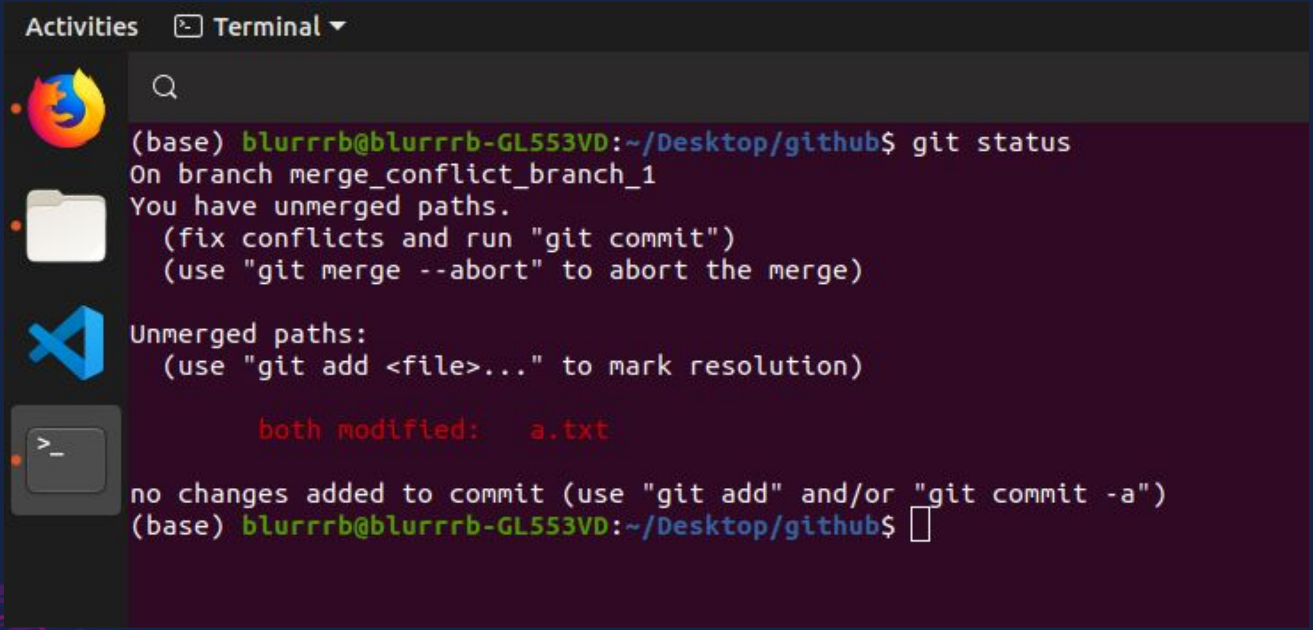


```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout -b merge_conflict_branch_1
Switched to a new branch 'merge_conflict_branch_1'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file added'
[merge_conflict_branch_1 (root-commit) 6c335c9] file added
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout -b merge_conflict_branch_2
Switched to a new branch 'merge_conflict_branch_2'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ ls
a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text 2" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file modified'
[merge_conflict_branch_2 0caaddb] file modified
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git checkout merge_conflict_branch_1
Switched to branch 'merge_conflict_branch_1'
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ echo "some text 1" > a.txt
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git add . && git commit -m 'file modified'
[merge_conflict_branch_1 09d5c33] file modified
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git merge merge_conflict_branch_2
Auto-merging a.txt
CONFLICT (content): Merge conflict in a.txt
Automatic merge failed; fix conflicts and then commit the result.
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

Hacktoberfest 2019 presented by DigitalOcean and DEV

Merge failed

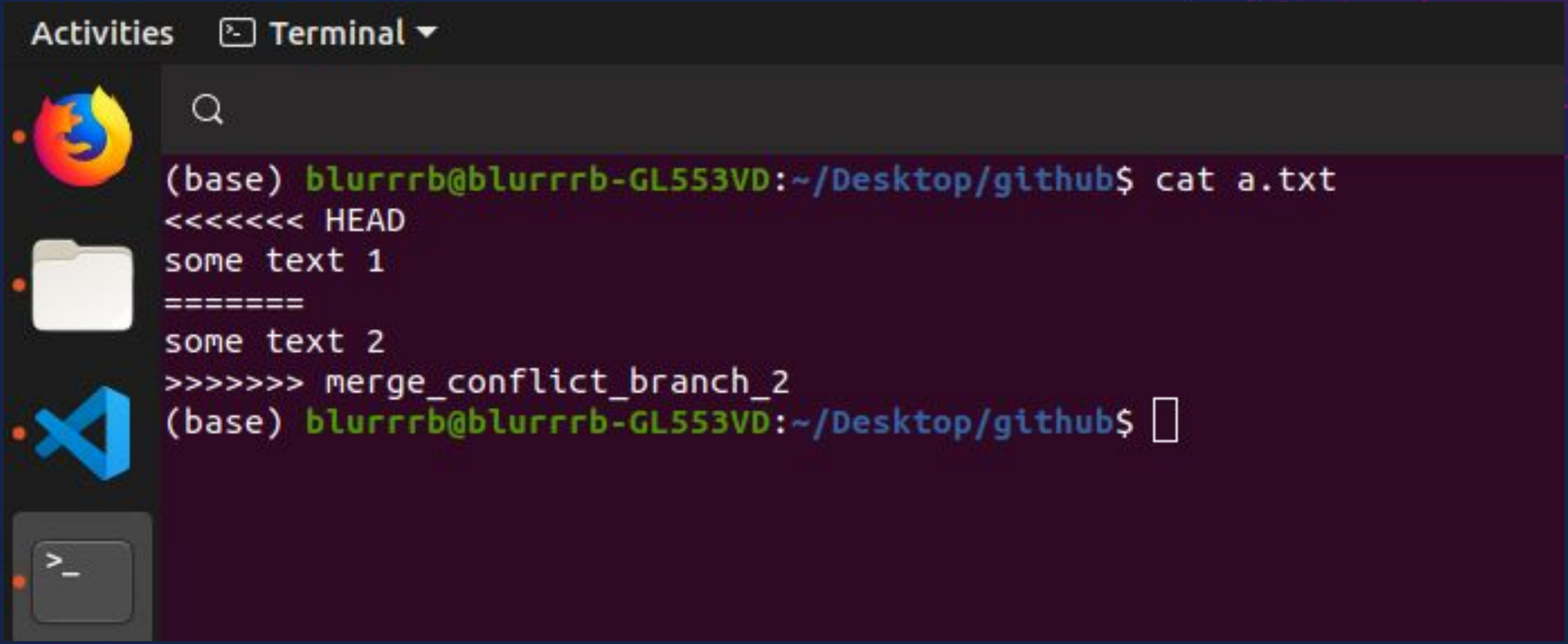
Lets see what git status has to say



```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git status
On branch merge_conflict_branch_1
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

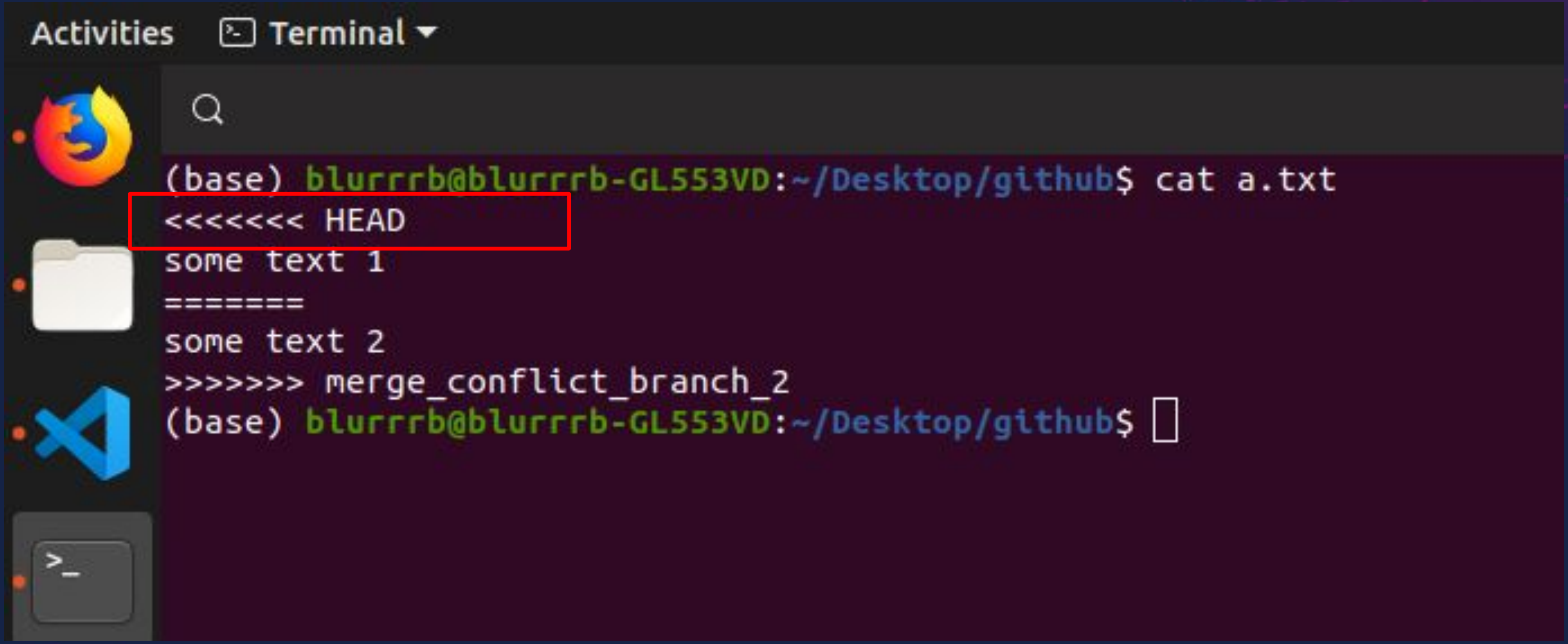
Merge failed and it has something to do with a.txt

Open a.txt to inspect its contents



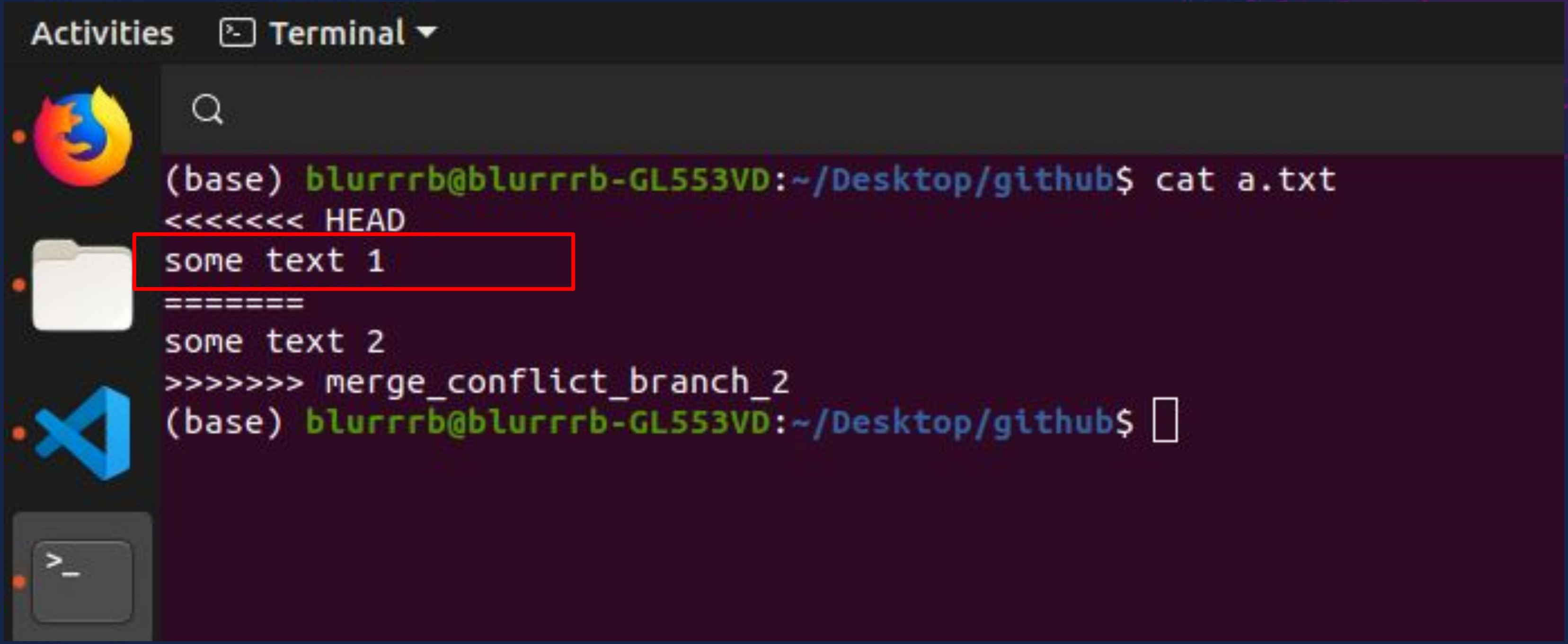
```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

Hmmm. We never typed some of these lines. Where did these come from?



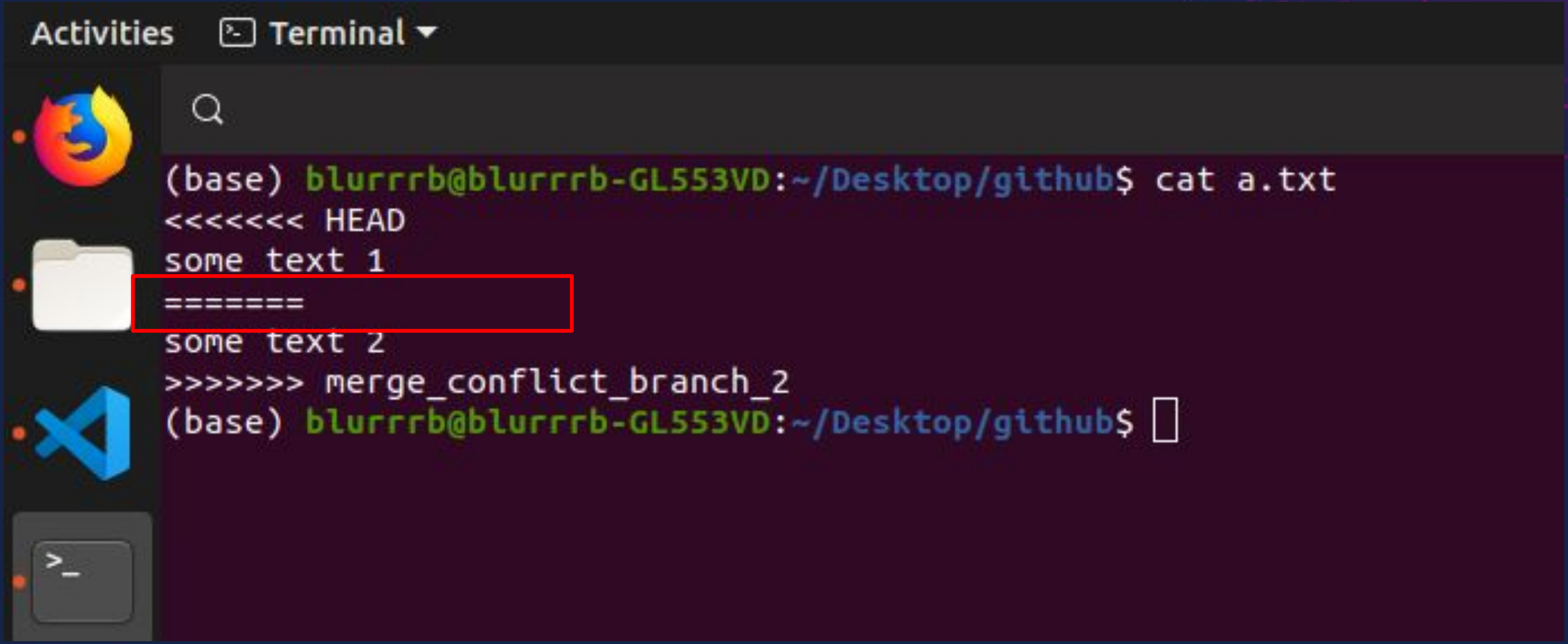
```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

<<<<<<< HEAD - indicates that conflict occured at the HEAD of current branch



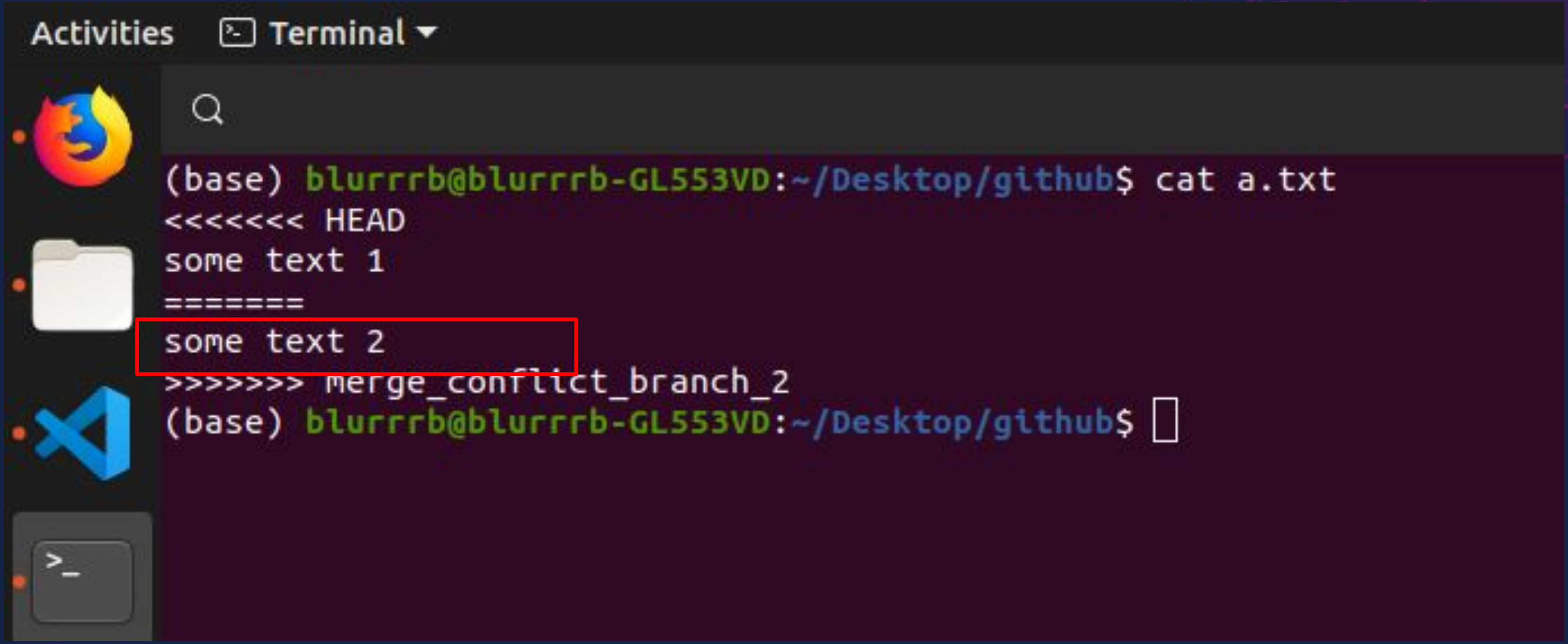
```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

Changes in the current branch



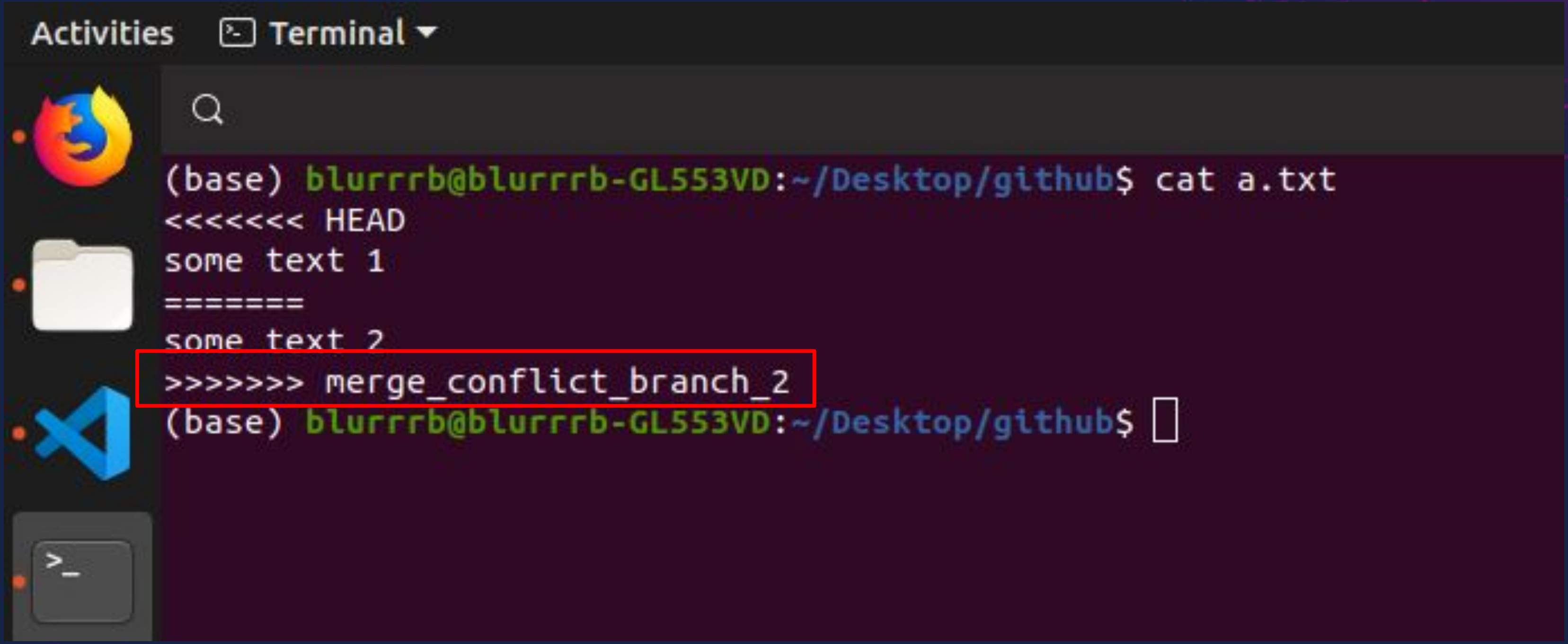
```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ 
```

Separator for the changes



```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ 
```

Changes in the 2nd branch

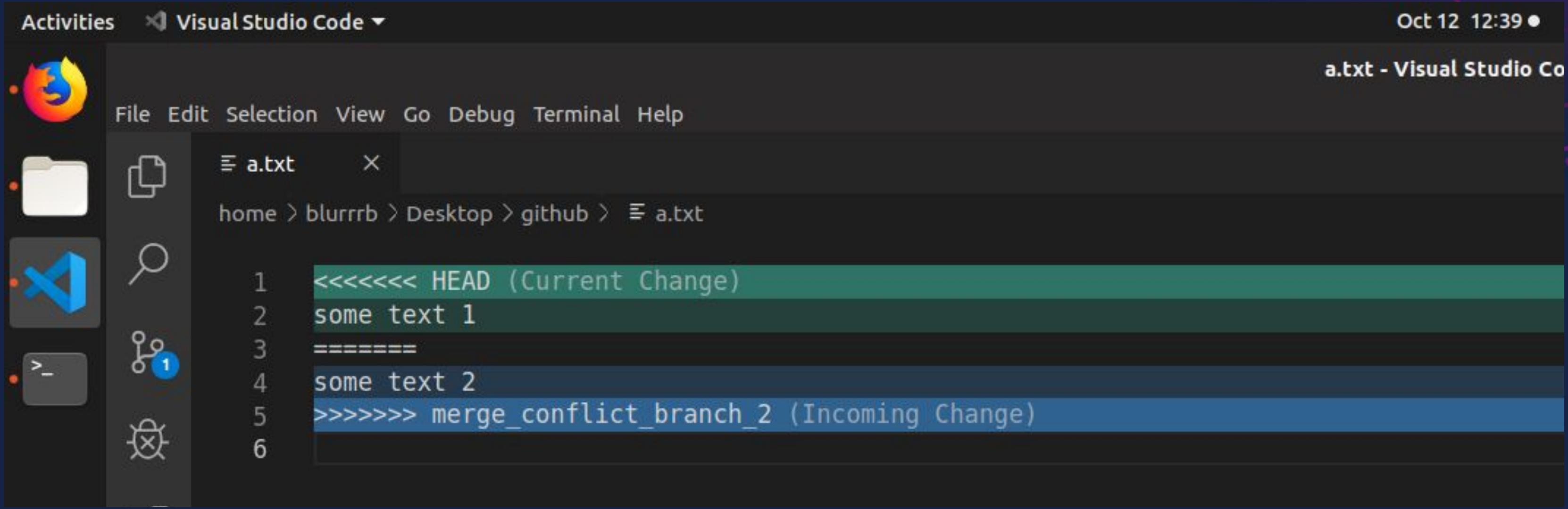


```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ cat a.txt
<<<<<<< HEAD
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

Branch name whose incoming changes caused the conflict

Resolving the conflict

- Open the file
- Remove >>>>>>Head, ==========, <<<<<<<<< Conflict branch
- Things you can do
  - Remove changes from first branch
  - Remove changes from second branch
  - Remove changes from both branches
  - Add new changes
  - Any combination of the above
- git add . && git commit -m ‘merge conflicts resolved’ - stage changes and commit away


Activities
Visual Studio Code
Oct 12 12:39
a.txt - Visual Studio Co
File Edit Selection View Go Debug Terminal Help
a.txt
home > blurrrb > Desktop > github > a.txt

```
<<<<<<< HEAD (Current Change)
some text 1
=======
some text 2
>>>>>>> merge_conflict_branch_2 (Incoming Change)
```

Editors like vs code make it even easier to resolve merge conflicts!

Git Rebase

Rebase is used to rewrite the commit history.

Things you can do:

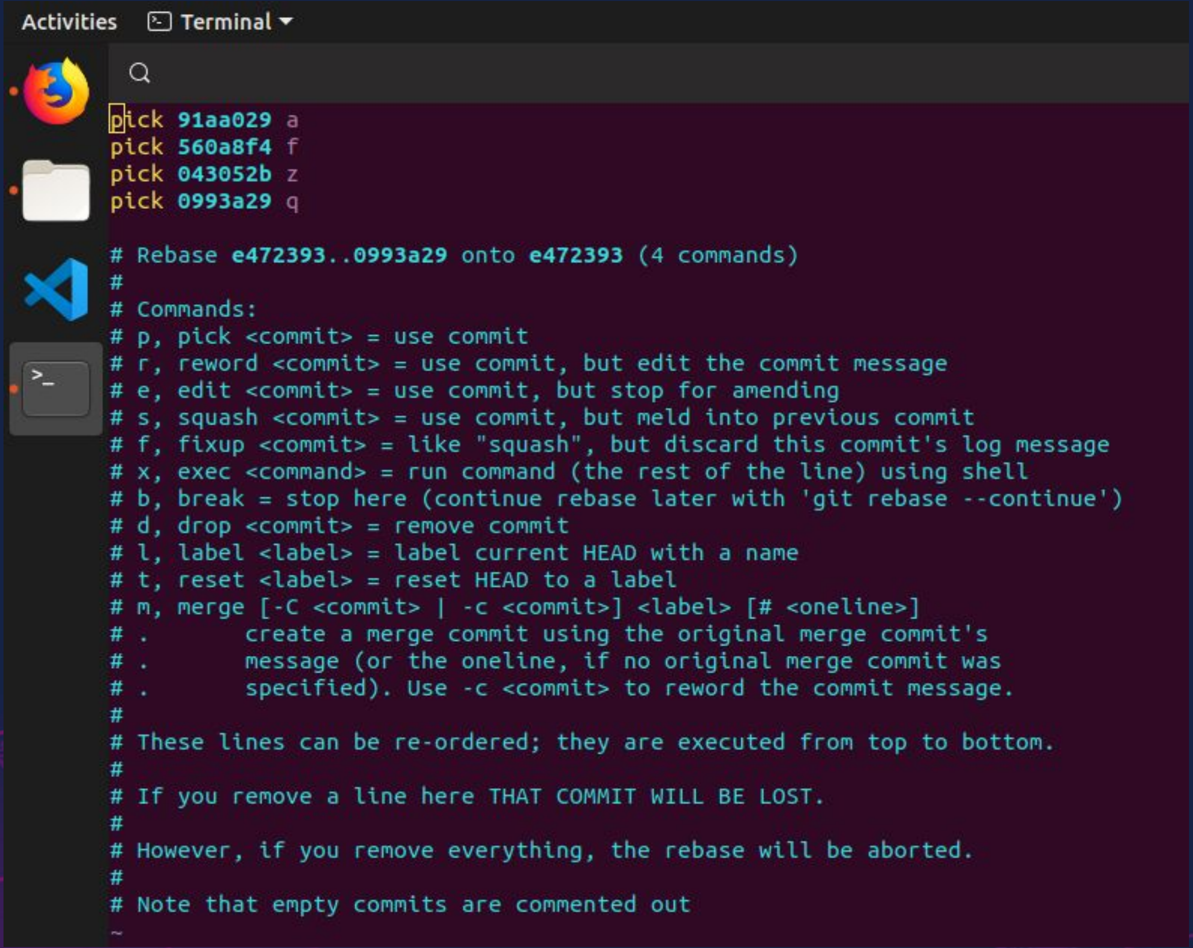
- Change commit messages
- Merge commits
- Reorder commits
- Remove commits

Git Rebase Syntax

- git rebase <branch> - moves your commits on current branch to the head of <branch>. Similar, to git merge but doesn't create a merge commit.
- git rebase -i <branch> - opens an editor to modify the commit history.

Demo: git rebase -i branch

- Go inside a git repo.
- Create multiple commits
- Run ‘git commit -i HEAD~n’. n is the number of commits from HEAD you want to modify.



```
pick 91aa029 a
pick 560a8f4 f
pick 043052b z
pick 0993a29 q

# Rebase e472393..0993a29 onto e472393 (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
```

You should see something similar to this.

To reorder commits, just reorder the corresponding lines.

To change commit message change pick to reword. Or use short form ‘r’ for reword.

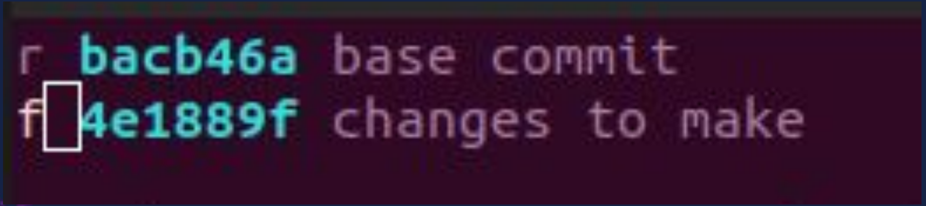
To merge commits, change ‘pick’ to ‘fixup’. The commits set to ‘fixup’ will be merged with the previous ‘pick’ commit.

Run git log after this to check the changes made

What to do when you commit a change and just realize you forgot to add something?

It can be done easily using git rebase

- Commit the new changes
- git rebase -i HEAD~2 - rebase last 2 commits.
- Change first ‘pick’ to ‘r’ for ‘reword’ and second ‘pick’ to ‘f’ for ‘fixup’ and save the file.



```
r bacb46a base commit
f 4e1889f changes to make
```

Shortcut for the above steps:

- Stage the changes using git add .
- git commit --amend - amend the changes

What to do in case you want to throw away some changes?

It can be done easily using git rebase

- Run git rebase -i HEAD~1 and change ‘pick’ to ‘d’ for ‘drop’ for the commit.

Shortcut:

- Run git log.
- Find out the hash for the commit you want to return to or you can use the HEAD~n notation.
- Run git reset --hard <hash or HEAD~n>

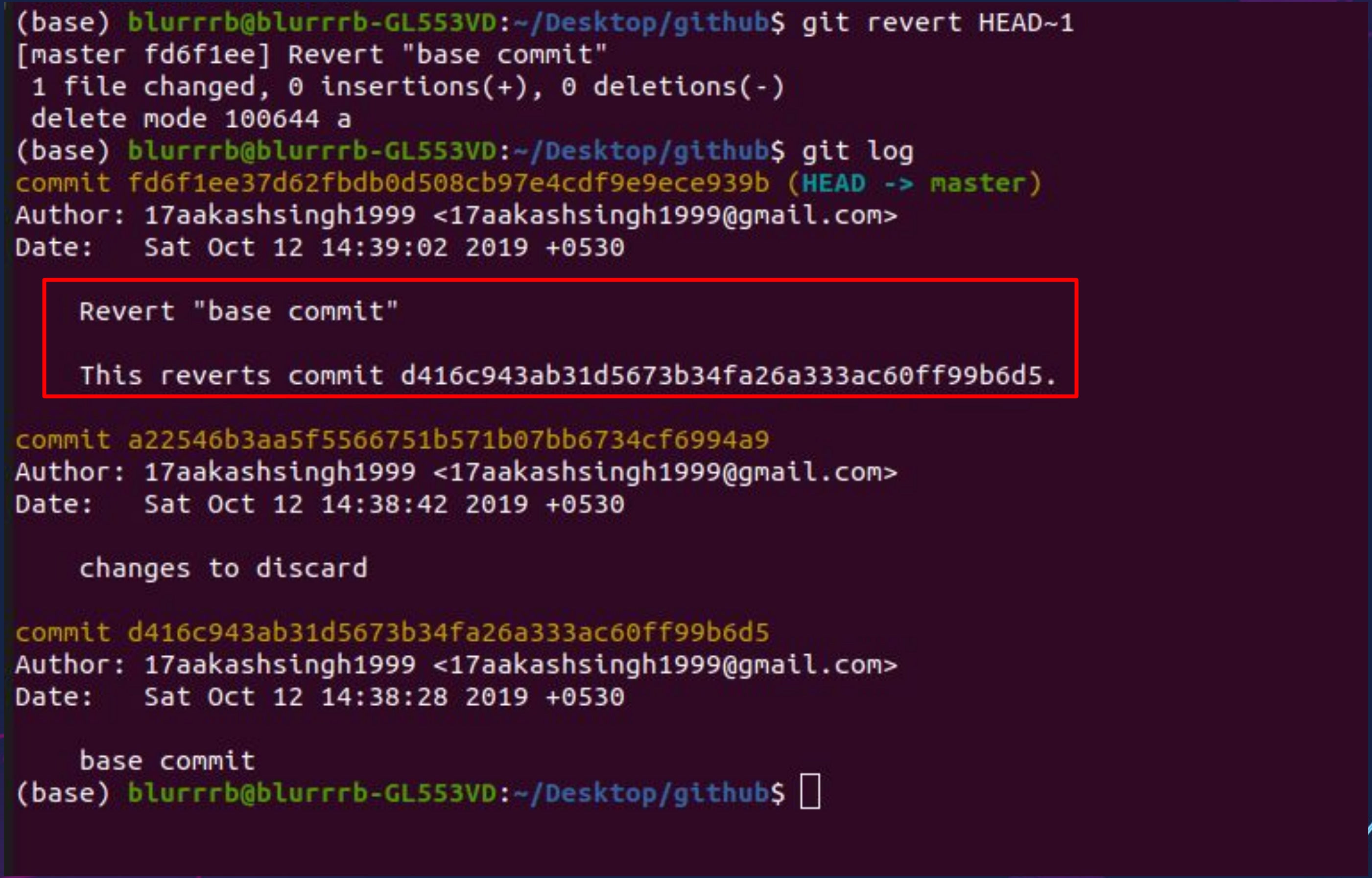
WARNING: Use the above commands carefully. Changes made by them cannot be reverted.

Safer way to reset to a previous commit

Instead of git reset, use git revert

Syntax: git revert <hash or HEAD~n>

This command works by creating a new commit to reverse the changes rather than deleting the commits.



```
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git revert HEAD~1
[master fd6f1ee] Revert "base commit"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 a
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$ git log
commit fd6f1ee37d62fbdb0d508cb97e4cdf9e9ece939b (HEAD -> master)
Author: 17aakashsingh1999 <17aakashsingh1999@gmail.com>
Date:   Sat Oct 12 14:39:02 2019 +0530

    Revert "base commit"

    This reverts commit d416c943ab31d5673b34fa26a333ac60ff99b6d5.

commit a22546b3aa5f5566751b571b07bb6734cf6994a9
Author: 17aakashsingh1999 <17aakashsingh1999@gmail.com>
Date:   Sat Oct 12 14:38:42 2019 +0530

    changes to discard

commit d416c943ab31d5673b34fa26a333ac60ff99b6d5
Author: 17aakashsingh1999 <17aakashsingh1999@gmail.com>
Date:   Sat Oct 12 14:38:28 2019 +0530

    base commit
(base) blurrrb@blurrrb-GL553VD:~/Desktop/github$
```

Thank You

Presented by:

- **Anumeha Agrawal**
- **Aakash Singh**