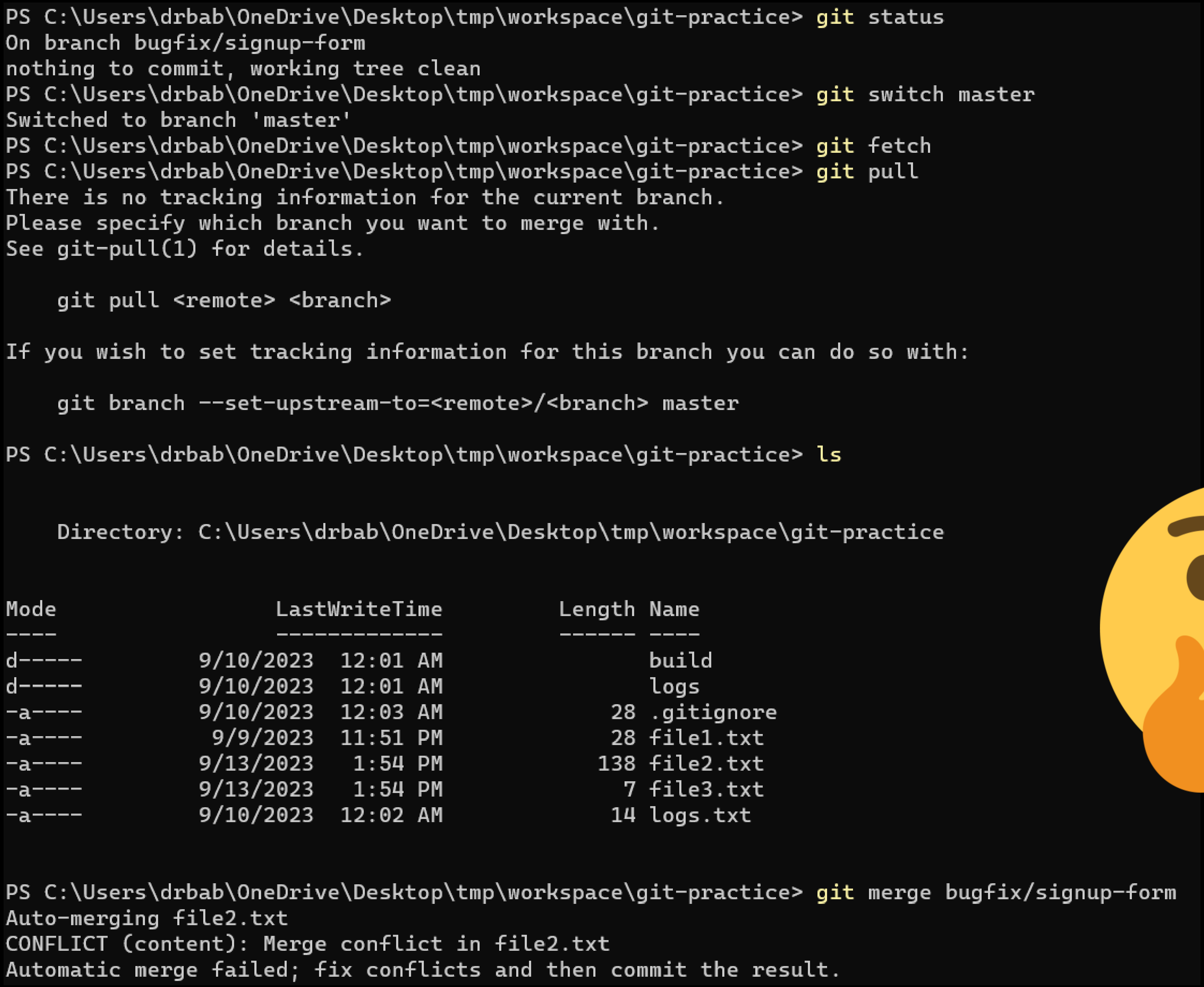
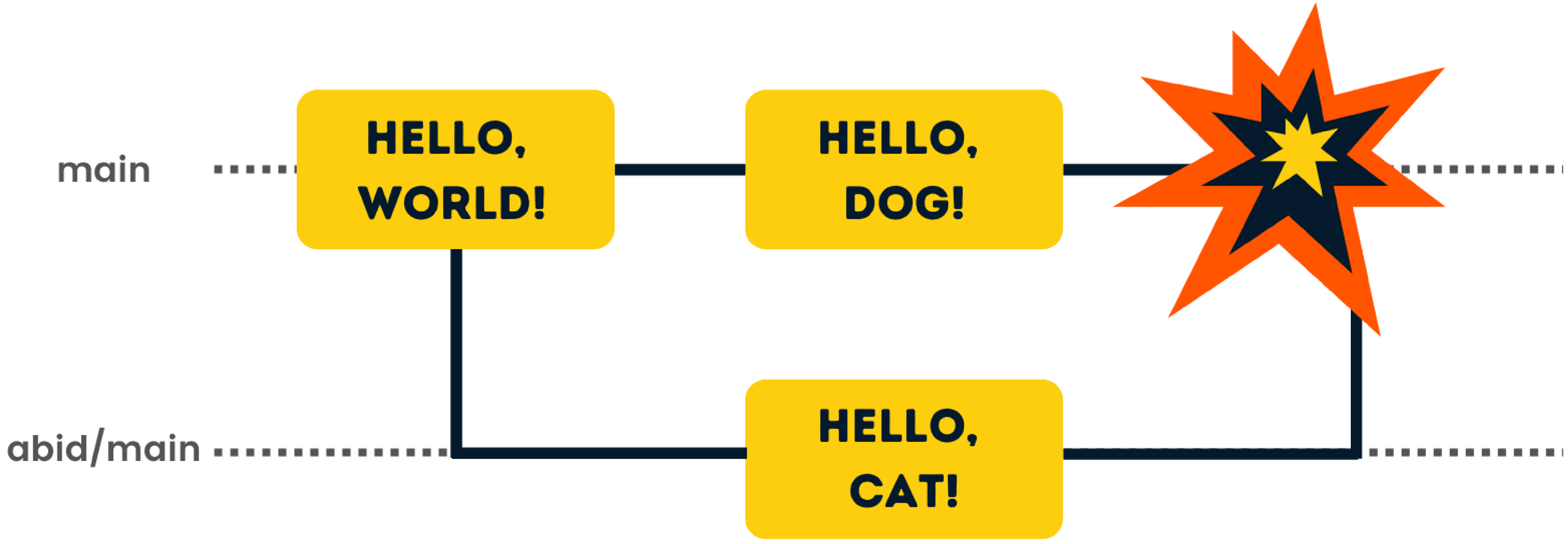
Basic git concepts #4

Majid Babaei



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch bugfix/signup-form
nothing to commit, working tree clean
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git switch master
Switched to branch 'master'
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git fetch
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git pull
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=<remote>/<branch> master

PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/10/2023  12:01 AM                build
d-----         9/10/2023  12:01 AM                logs
-a----         9/10/2023  12:03 AM             28 .gitignore
-a----          9/9/2023  11:51 PM             28 file1.txt
-a----         9/13/2023   1:54 PM            138 file2.txt
-a----         9/13/2023   1:54 PM              7 file3.txt
-a----         9/10/2023  12:02 AM             14 logs.txt


PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git merge bugfix/signup-form
Auto-merging file2.txt
CONFLICT (content): Merge conflict in file2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

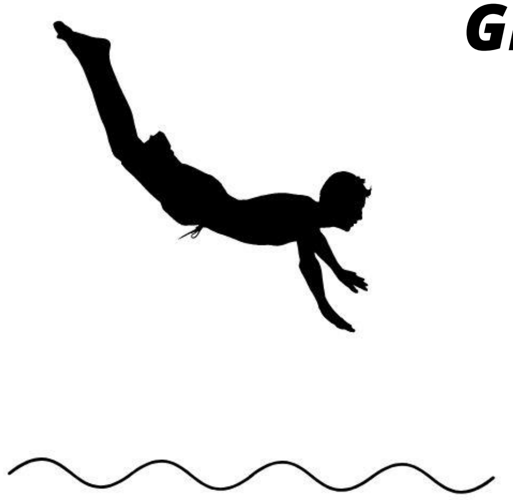

main
abid/main
HELLO, WORLD!
HELLO, DOG!
HELLO, CAT!

Merge conflicts

- In the real-world often we run into conflicts when merge branches

- It happens when:
  - Same *code has been changed in different ways* in different branches
  - The lines that have been *changed in one branch, deleted in another branch*
  - *Same file added* in two different branches with different content

Git cannot figure out how to merge the changes!

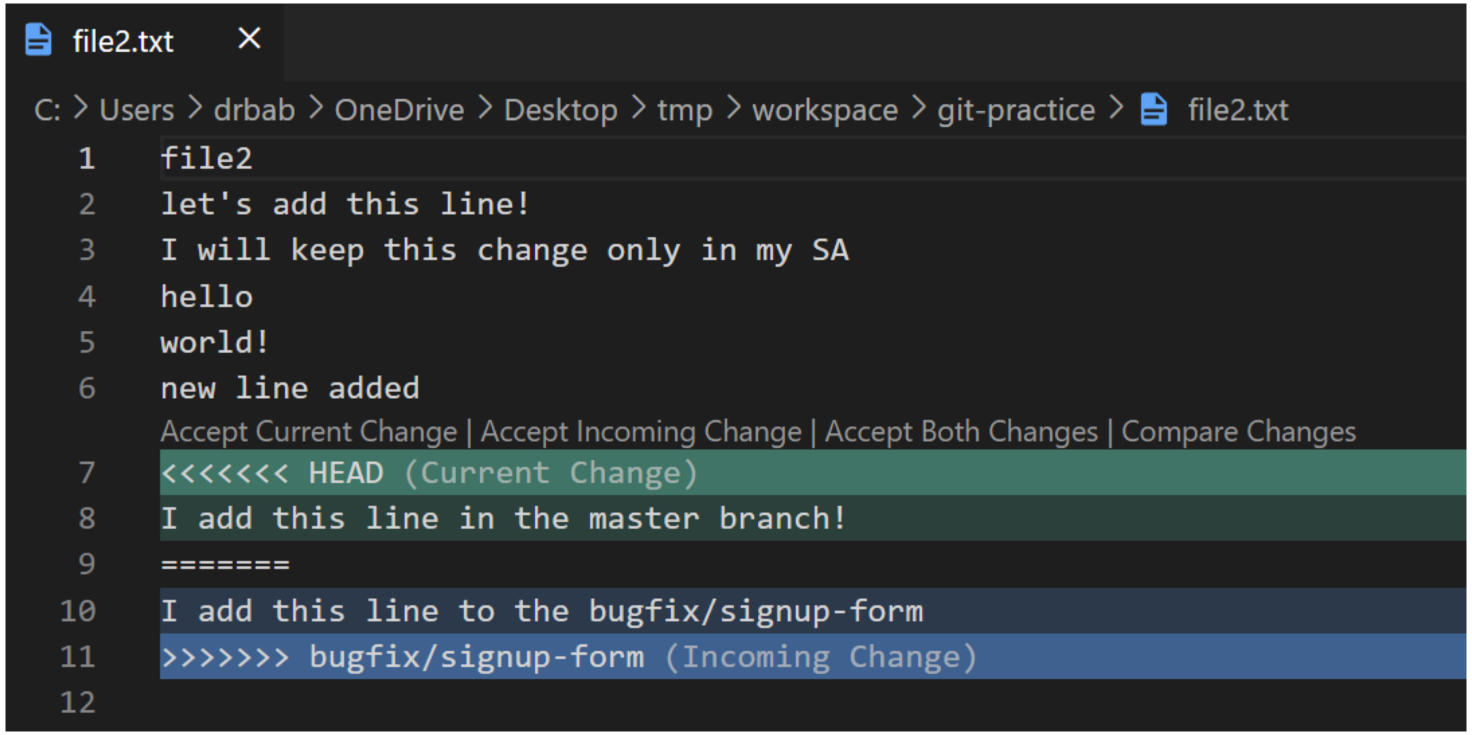


We need to jump in and tell git how we want to proceed!

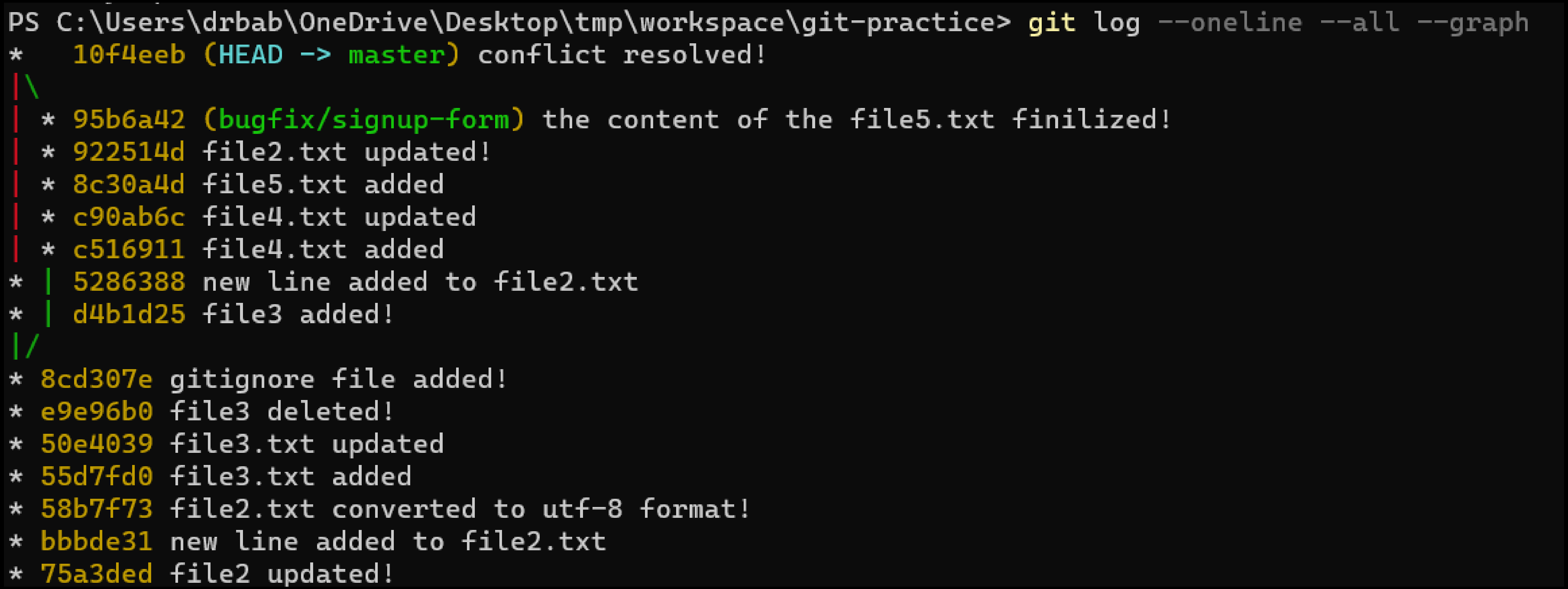
How does a Merge conflict look like?

- For example:
  - It shows changes in the current branch and in the other branch

code file2.txt


file2.txt
C: > Users > drbab > OneDrive > Desktop > tmp > workspace > git-practice > file2.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
file2
let's add this line!
I will keep this change only in my SA
hello
world!
new line added
<<<<<<< HEAD (Current Change)
I add this line in the master branch!
=======
I add this line to the bugfix/signup-form
>>>>>>> bugfix/signup-form (Incoming Change)

```

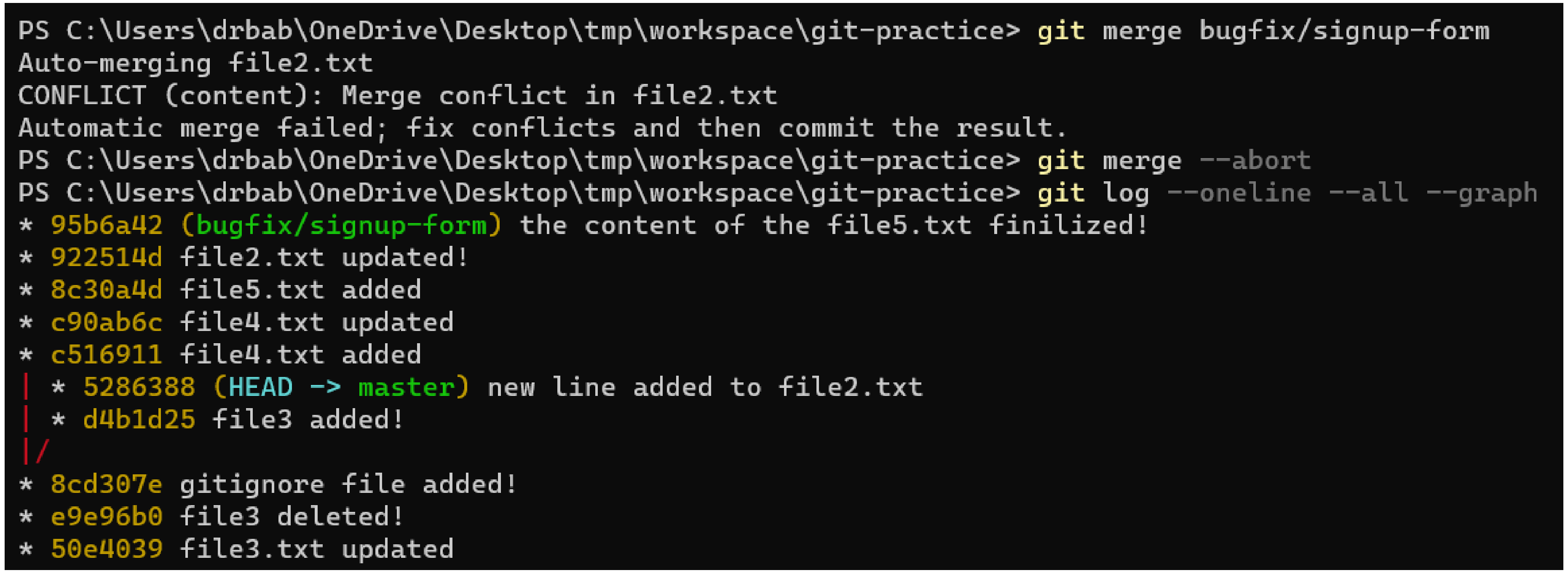


```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log --oneline --all --graph
*   10f4eeb (HEAD -> master) conflict resolved!
|\
| * 95b6a42 (bugfix/signup-form) the content of the file5.txt finilized!
| * 922514d file2.txt updated!
| * 8c30a4d file5.txt added
| * c90ab6c file4.txt updated
| * c516911 file4.txt added
* | 5286388 new line added to file2.txt
* | d4b1d25 file3 added!
|/
* 8cd307e gitignore file added!
* e9e96b0 file3 deleted!
* 50e4039 file3.txt updated
* 55d7fd0 file3.txt added
* 58b7f73 file2.txt converted to utf-8 format!
* bbbde31 new line added to file2.txt
* 75a3ded file2 updated!
```

Aborting a merge

- If you don't have time to resolve the conflict you can abort the merge and go back to the previous stage.

git merge --abort



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git merge bugfix/signup-form
Auto-merging file2.txt
CONFLICT (content): Merge conflict in file2.txt
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git merge --abort
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log --oneline --all --graph
* 95b6a42 (bugfix/signup-form) the content of the file5.txt finilized!
* 922514d file2.txt updated!
* 8c30a4d file5.txt added
* c90ab6c file4.txt updated
* c516911 file4.txt added
| * 5286388 (HEAD -> master) new line added to file2.txt
| * d4b1d25 file3 added!
|/
* 8cd307e gitignore file added!
* e9e96b0 file3 deleted!
* 50e4039 file3.txt updated
```

Undoing a faulty merge

- Sometimes we do a merge and then we will find out that the code doesn't get compiled, or it is faulty!
- That happens if you don't combine the changes properly!

- You should *undo the merge* and then *remerge*
  - Option#1: Reset the HEAD pointer!
    - You should be careful because you rewiring the history!

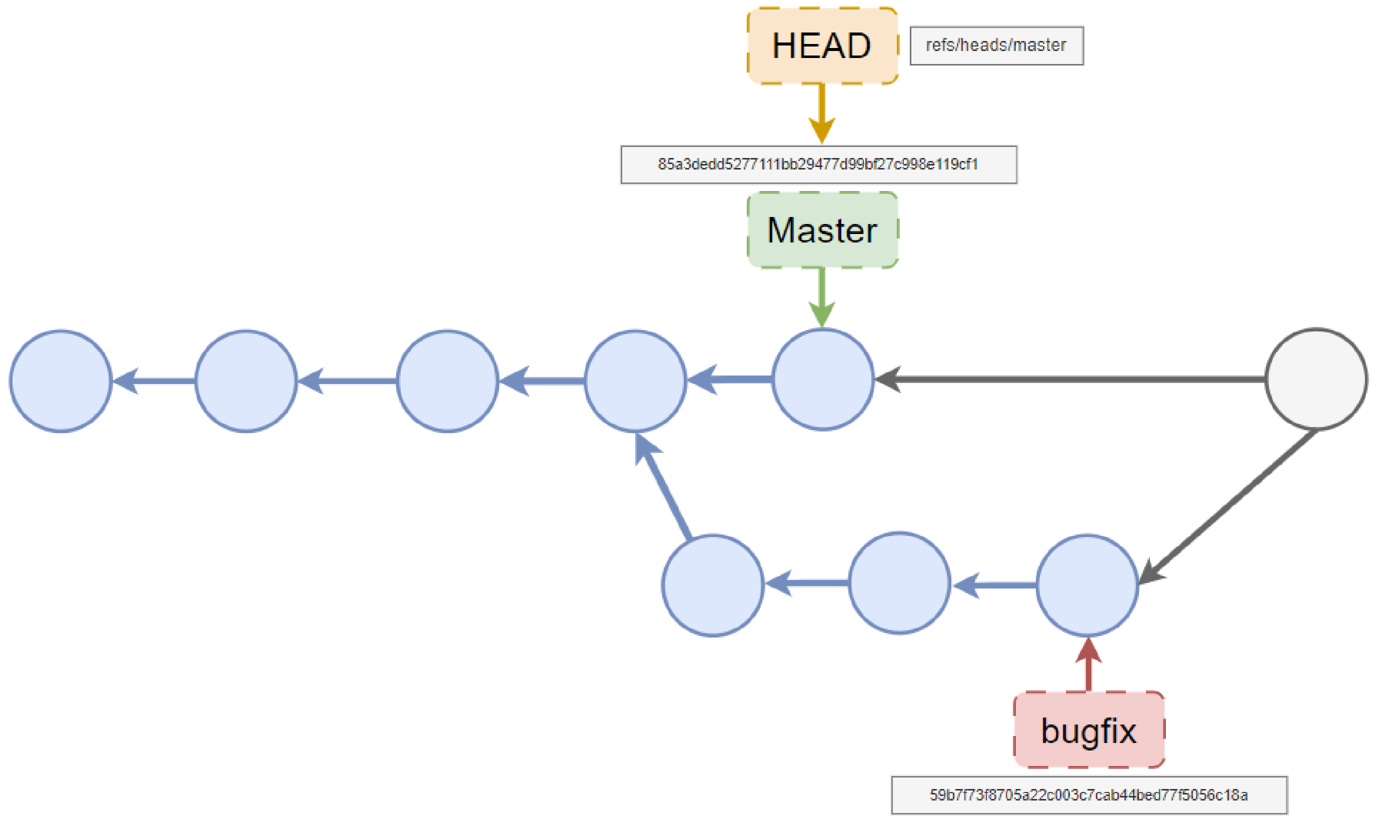
If it is local there is no problem but if it is shared with others, that might be problematic!

  - Option#2: *revert the Merge* by adding a new commit that will cancel all you have done for merging.

Removing a commit – *by resetting the HEAD*

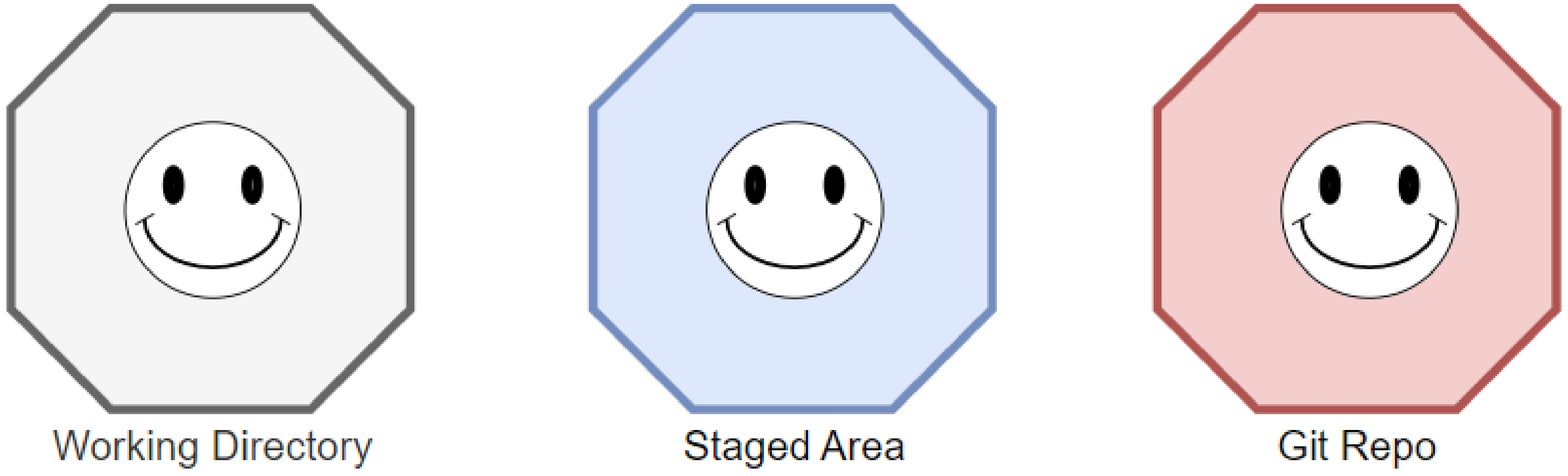
- Simply move the pointer to the last commit in the master branch
- This will create a garbage commit (*will be removed automatically by git*)

git reset --[hard|mixed|soft] HEAD~1

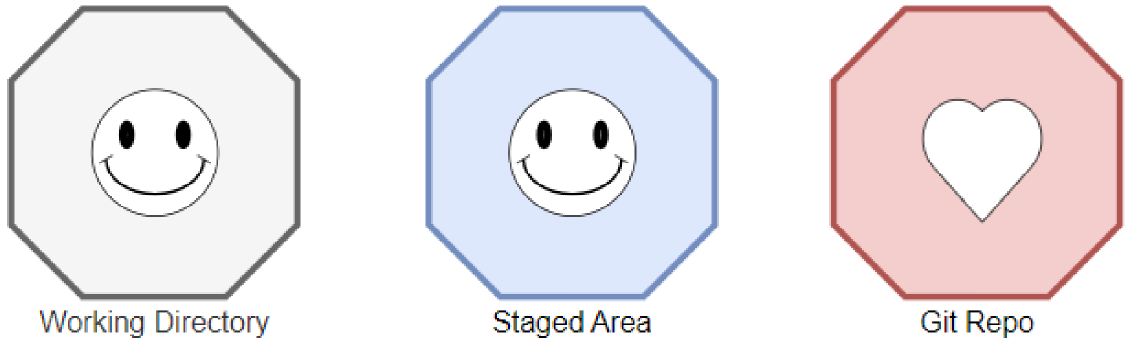

HEAD
refs/heads/master
85a3dedd5277111bb29477d99bf27c998e119cf1
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

Resetting *HEAD* pointer

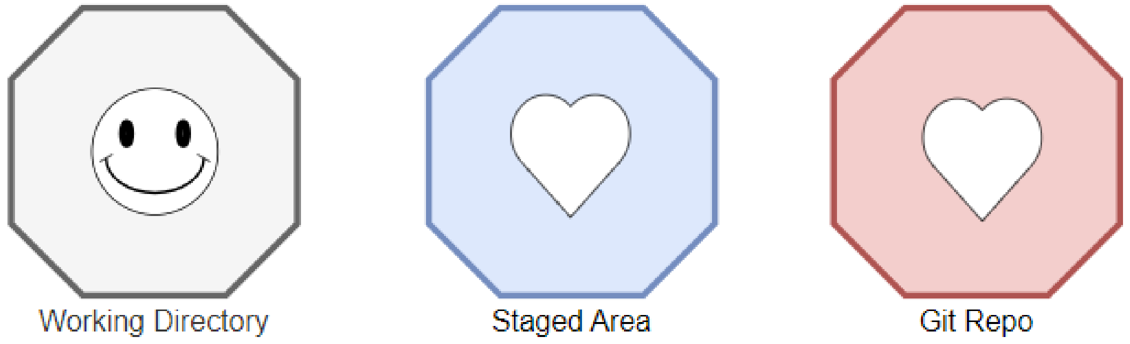
- When we are resetting the HEAD pointer we have 3 options:
    - *--soft*
    - *--mixed*
    - *--hard*



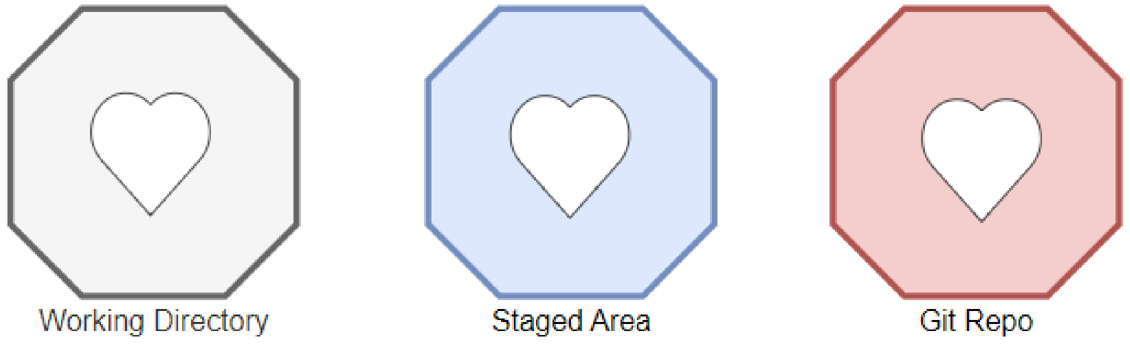
git reset --soft HEAD~1



git reset --mixed HEAD~1



git reset --hard HEAD~1



Reset HEAD does not work when …

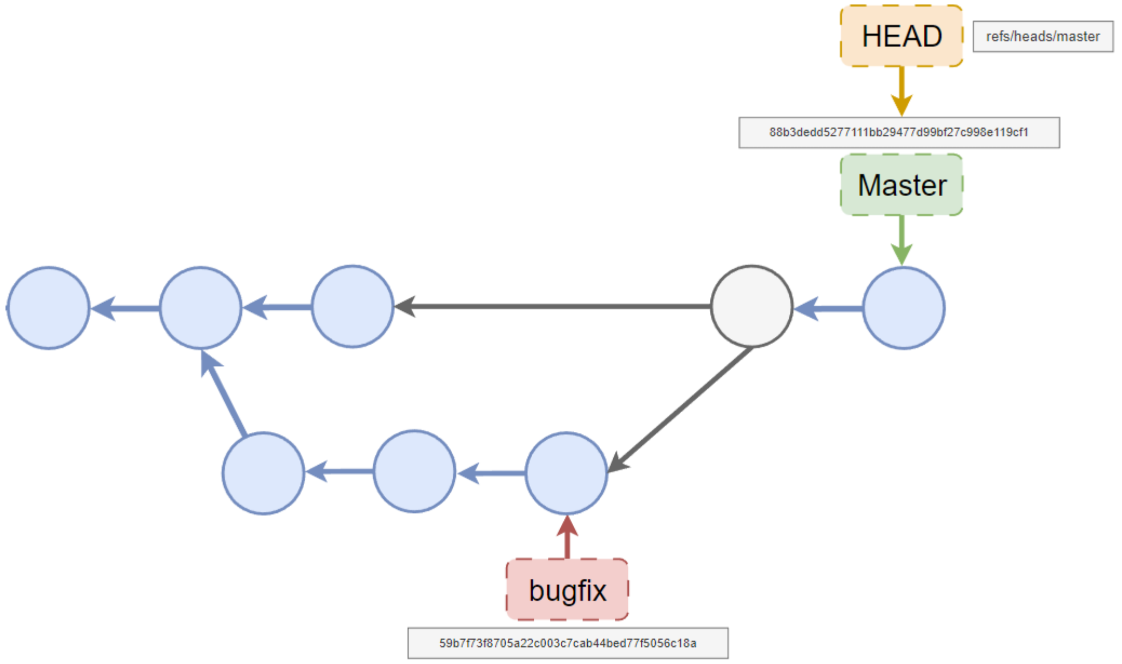
- Reset HEAD is useful when you have **NOT** already pushed the merge to a remote repository.

In such a case, when you've already shared the merge commit with your colleagues on a remote, you should consider a different solution.

Revert a merge

- Here we should specify we want to revert to what commit

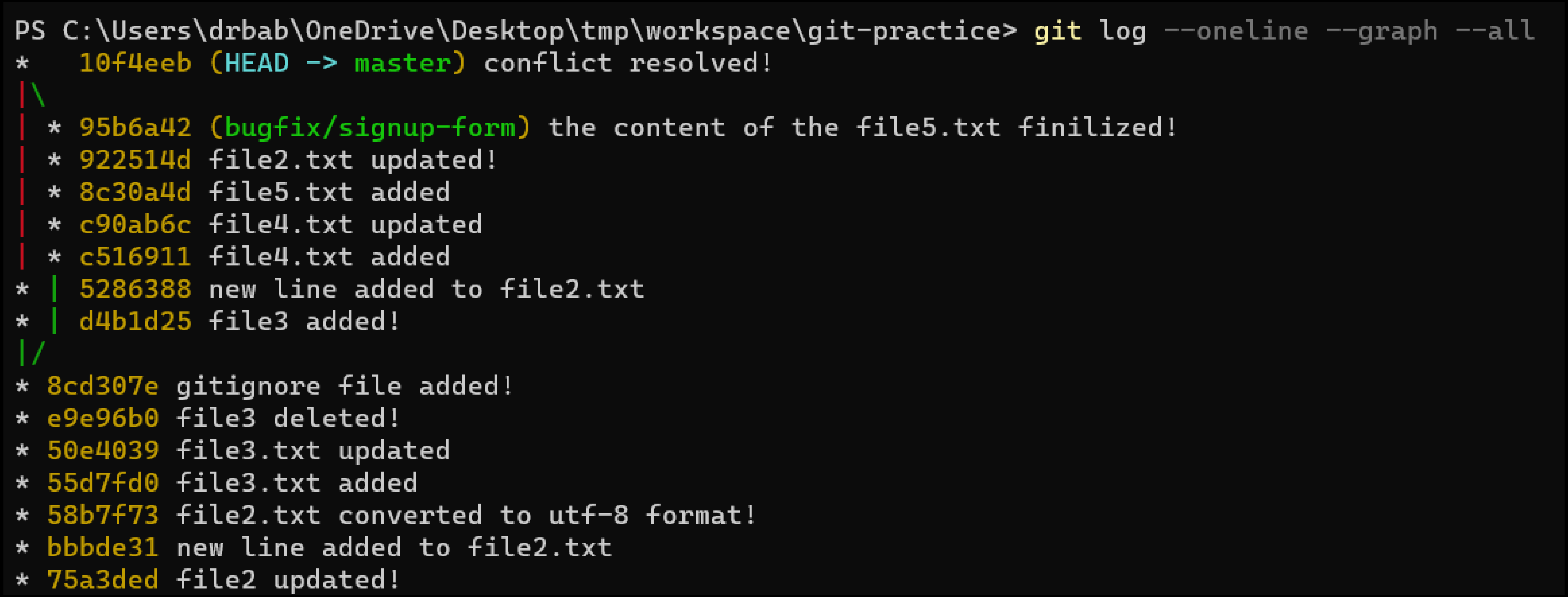
- Since the main line of development is in the master branch, we should back to the commit in the master branch. *git revert –m 1 [target commit]*


HEAD
refs/heads/master
88b3dedd5277111bb29477d99bf27c998e119cf1
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

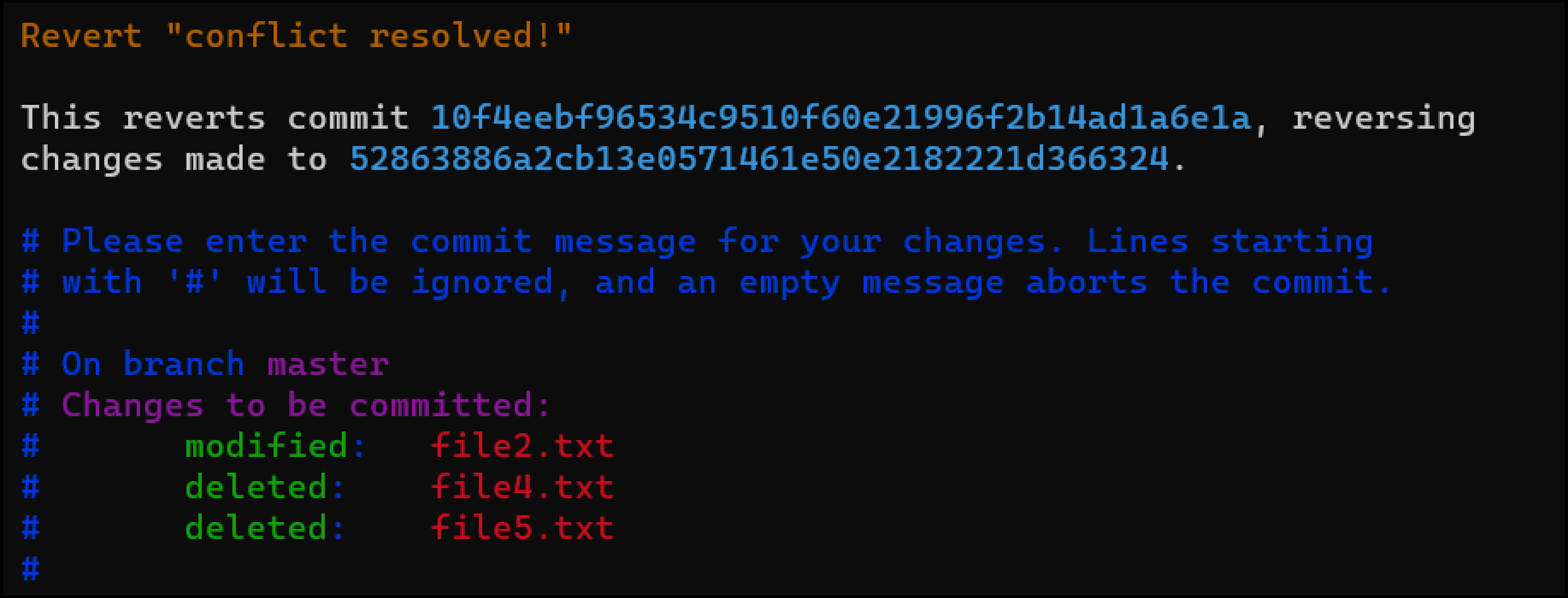
Let’s take a closer look!

- *git revert* will make sure that a new commit is created to revert the effects of that unwanted merge. *This is in contrast to git reset, where we effectively "remove" a commit from the history.*

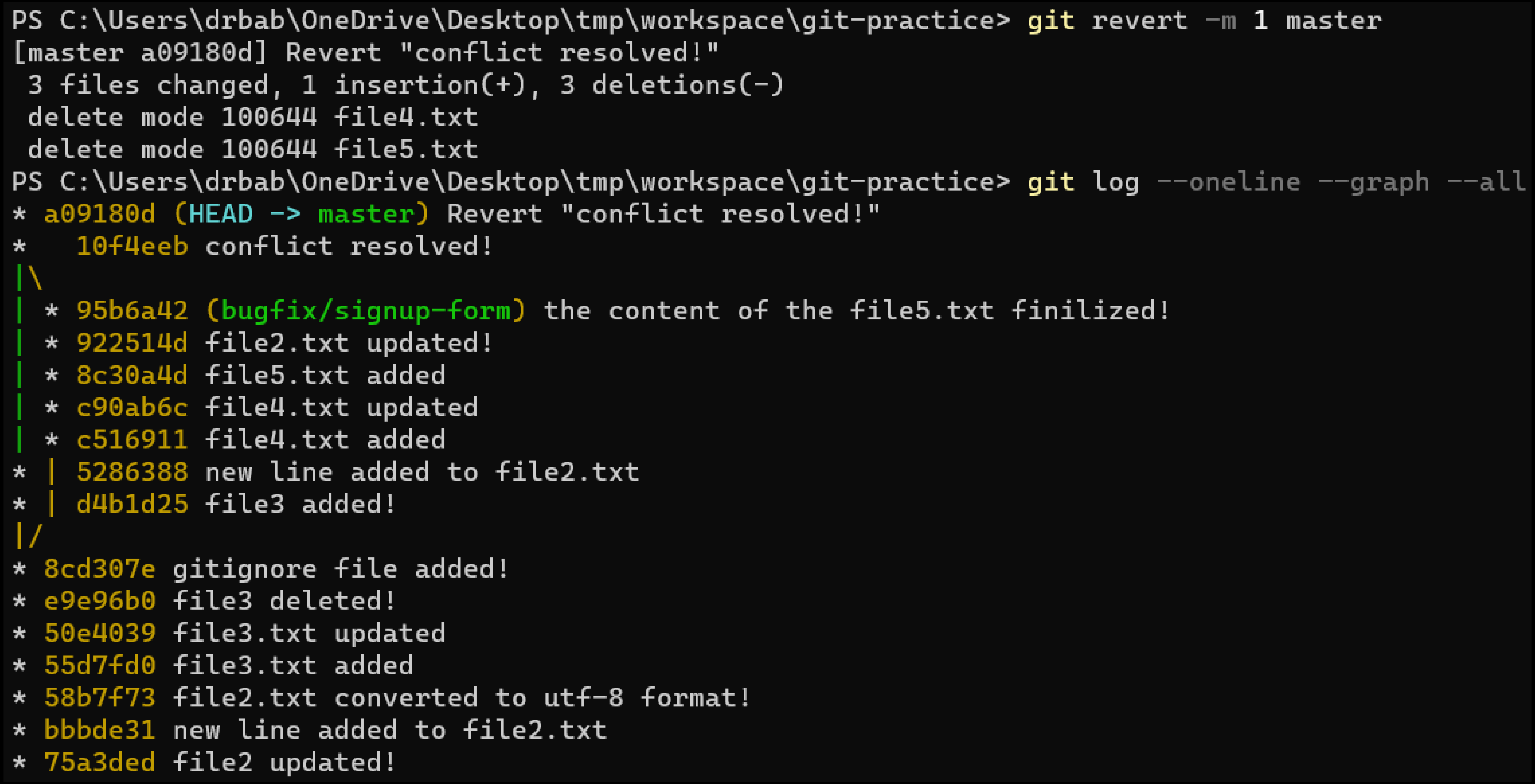
- The *-m 1* option tells Git that we want to keep the parent side of the merge (which is the branch we had merged into).



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log --oneline --graph --all
*   10f4eeb (HEAD -> master) conflict resolved!
|\
| * 95b6a42 (bugfix/signup-form) the content of the file5.txt finilized!
| * 922514d file2.txt updated!
| * 8c30a4d file5.txt added
| * c90ab6c file4.txt updated
| * c516911 file4.txt added
* | 5286388 new line added to file2.txt
* | d4b1d25 file3 added!
|/
* 8cd307e gitignore file added!
* e9e96b0 file3 deleted!
* 50e4039 file3.txt updated
* 55d7fd0 file3.txt added
* 58b7f73 file2.txt converted to utf-8 format!
* bbbde31 new line added to file2.txt
* 75a3ded file2 updated!
```



```
Revert "conflict resolved!"

This reverts commit 10f4eebf96534c9510f60e21996f2b14ad1a6e1a, reversing
changes made to 52863886a2cb13e0571461e50e2182221d366324.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   file2.txt
#       deleted:    file4.txt
#       deleted:    file5.txt
#
```

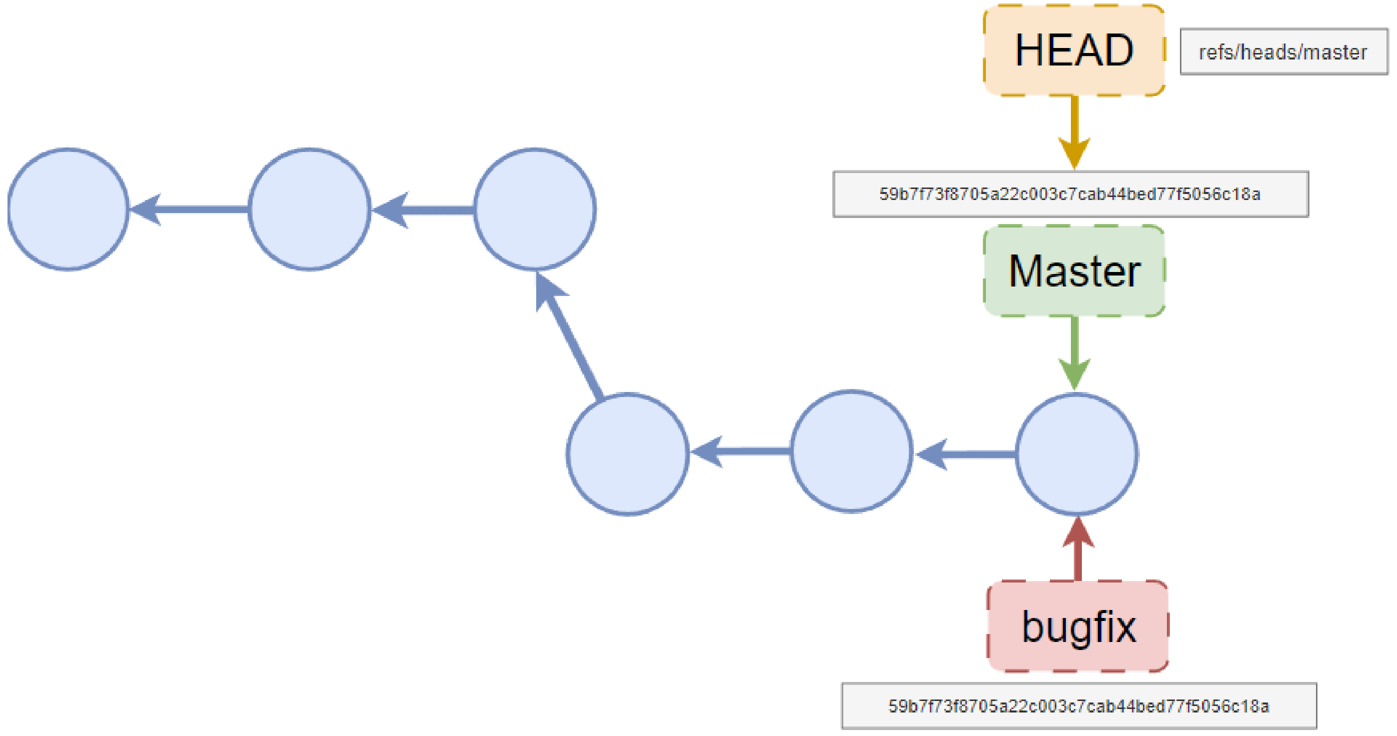


```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git revert -m 1 master
[master a09180d] Revert "conflict resolved!"
 3 files changed, 1 insertion(+), 3 deletions(-)
 delete mode 100644 file4.txt
 delete mode 100644 file5.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log --oneline --graph --all
* a09180d (HEAD -> master) Revert "conflict resolved!"
*   10f4eeb conflict resolved!
|\
| * 95b6a42 (bugfix/signup-form) the content of the file5.txt finilized!
| * 922514d file2.txt updated!
| * 8c30a4d file5.txt added
| * c90ab6c file4.txt updated
| * c516911 file4.txt added
* | 5286388 new line added to file2.txt
* | d4b1d25 file3 added!
|/
* 8cd307e gitignore file added!
* e9e96b0 file3 deleted!
* 50e4039 file3.txt updated
* 55d7fd0 file3.txt added
* 58b7f73 file2.txt converted to utf-8 format!
* bbbde31 new line added to file2.txt
* 75a3ded file2 updated!
```

Squash Merging

- Suppose we have two (low quality) commits in the bugfix branch
  - Maybe they are too fine grained
  - They don't represent single logical change set
  - We have mixed different things in each commit
  - We just add then as checkpoints along the way

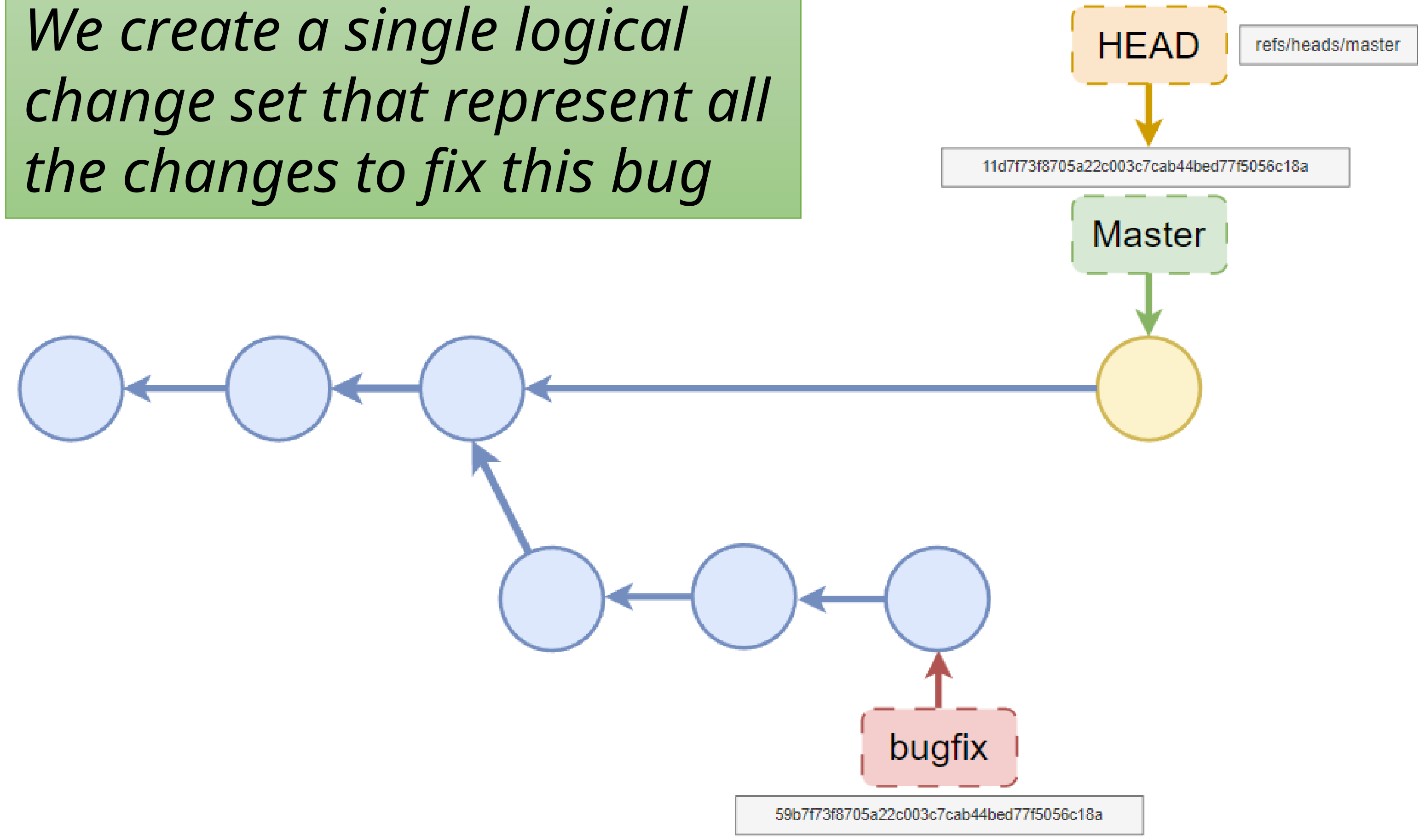
If we do a simple merge they will become part of the history of the master branch which is not good!


HEAD
refs/heads/master
59b7f73f8705a22c003c7cab44bed77f5056c18a
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

Squash Merging

- The solution is to create a new commit to combine all the changes we have made in bugfix branch

We create a single logical change set that represent all the changes to fix this bug


HEAD
refs/heads/master
11d7f73f8705a22c003c7cab44bed77f5056c18a
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

Why using *fast-forward* option is not always good?!

- A liner history that fast-forward gives when you do merging, *is not very accurate*!

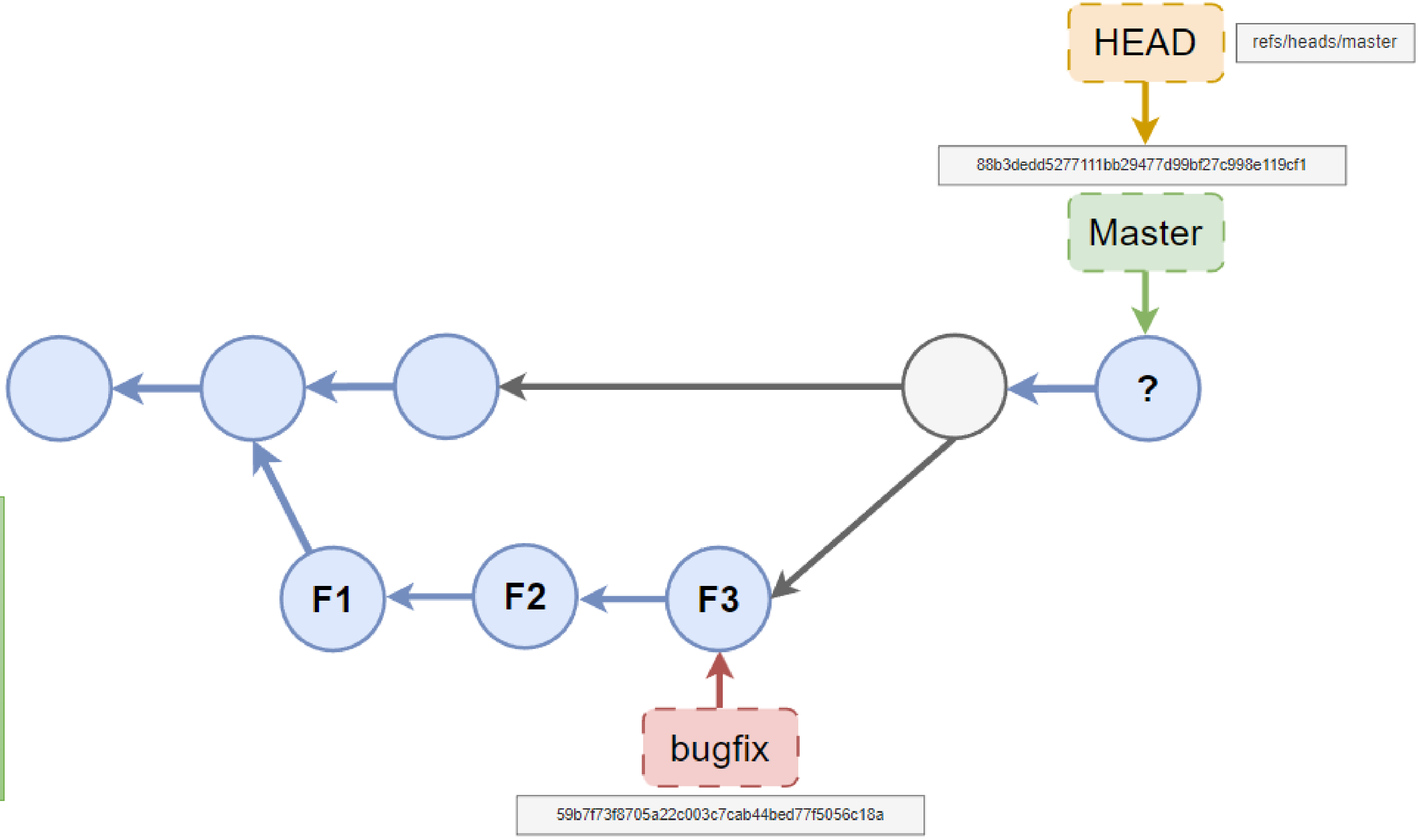
- *Reverting a feature* with *fast-forward* is not easy!

- Even if fast-forward is possible, don’t do it all the time! (*especially when you want to go back and check the history repeatedly*)
- Instead create a merge commit to combine all the changes you have made in two branches.

```
git merge --no-ff bugfix/login-form
```

Reverting a feature

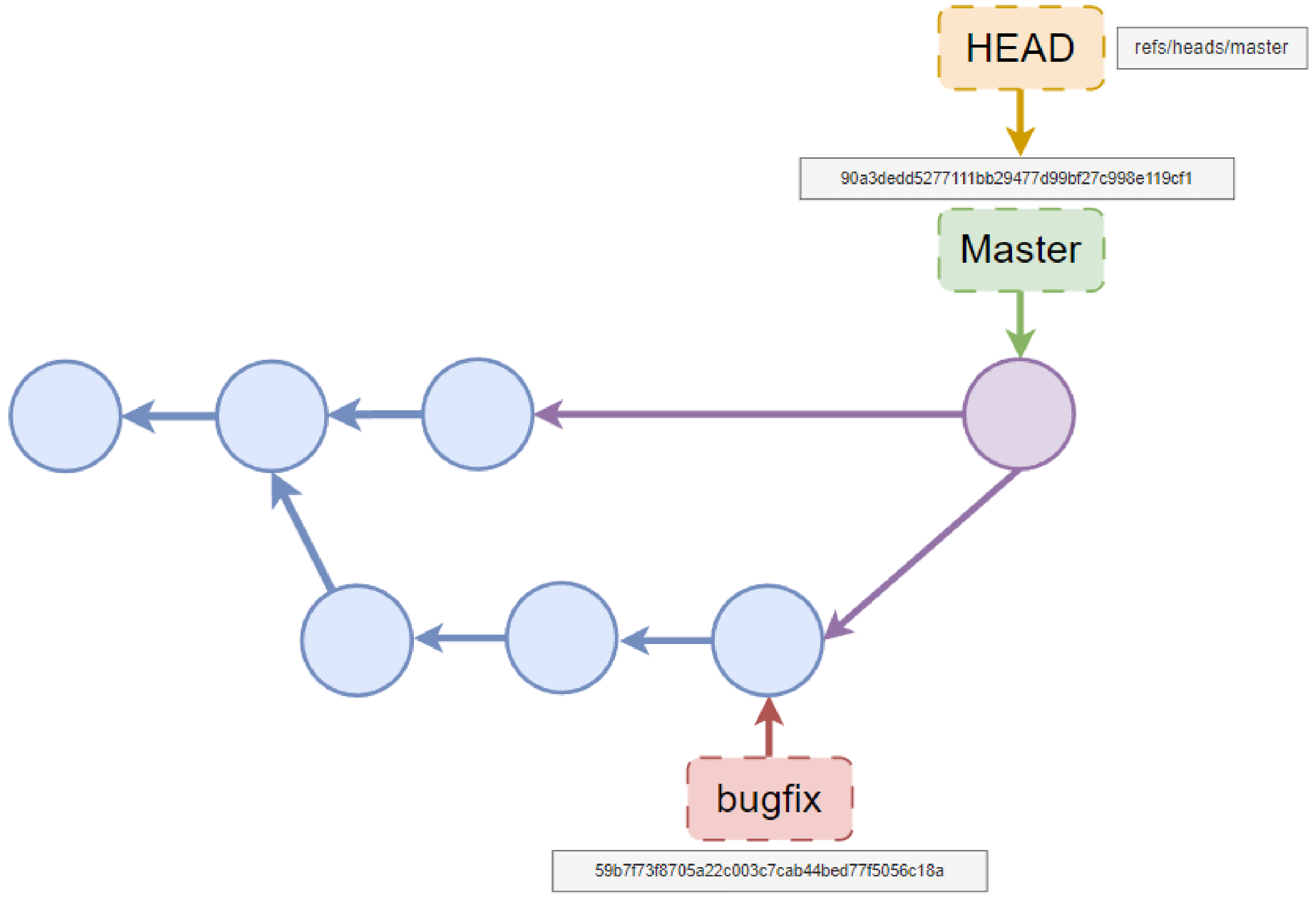
- Two features F1, F2, and F3 added in the feature branch
- You were asked to undo the changes


HEAD
refs/heads/master
88b3dedd5277111bb29477d99bf27c998e119cf1
Master
?
F1
F2
F3
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

At the end of the day! It really depends on your team culture and kind of the project that you are working on!

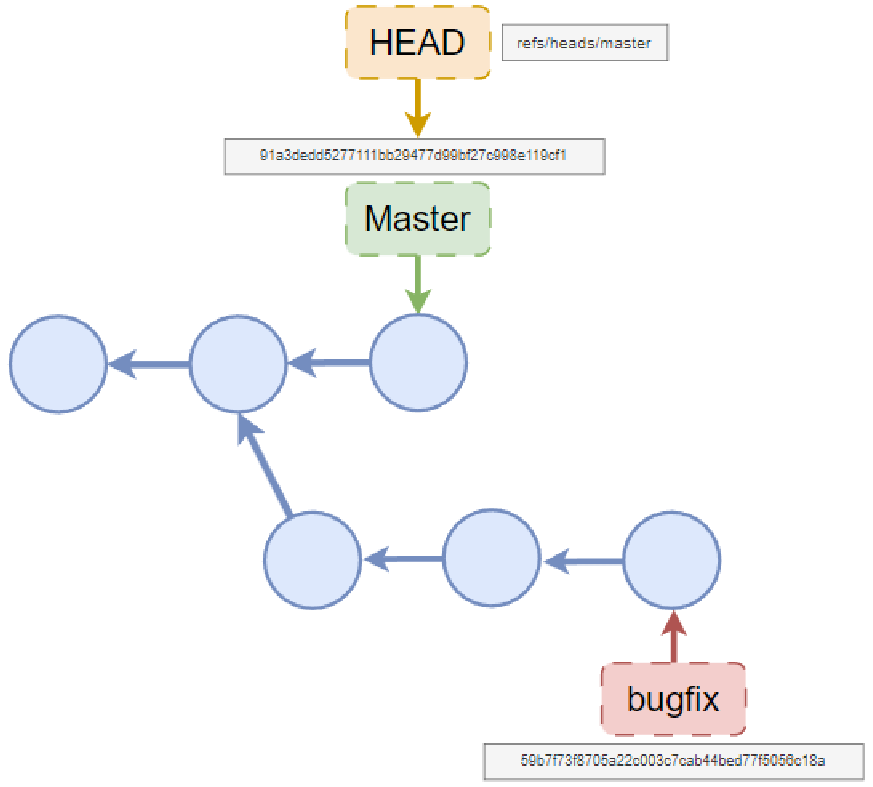
Rebasing

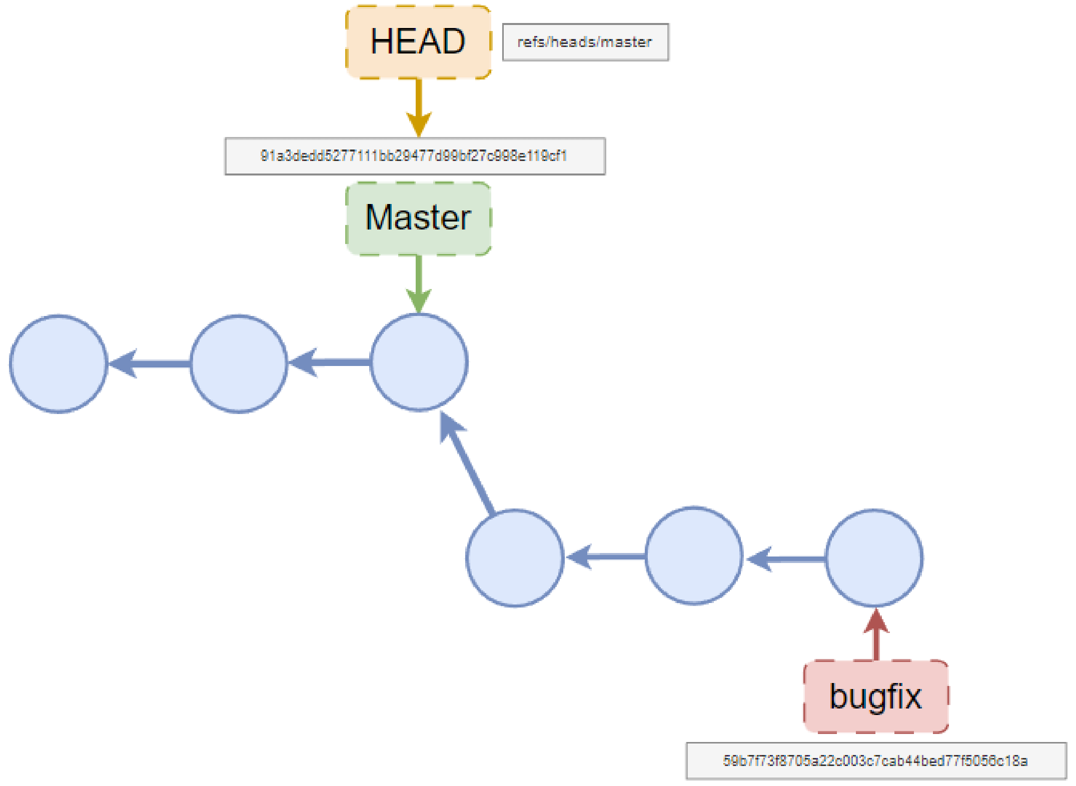
- The problem in merging is that we have no longer a linear history
- Another way to bring the changes to the master branch and maintain a liner history is rebasing.


HEAD
refs/heads/master
90a3dedd5277111bb29477d99bf27c998e119cf1
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

Rebasing

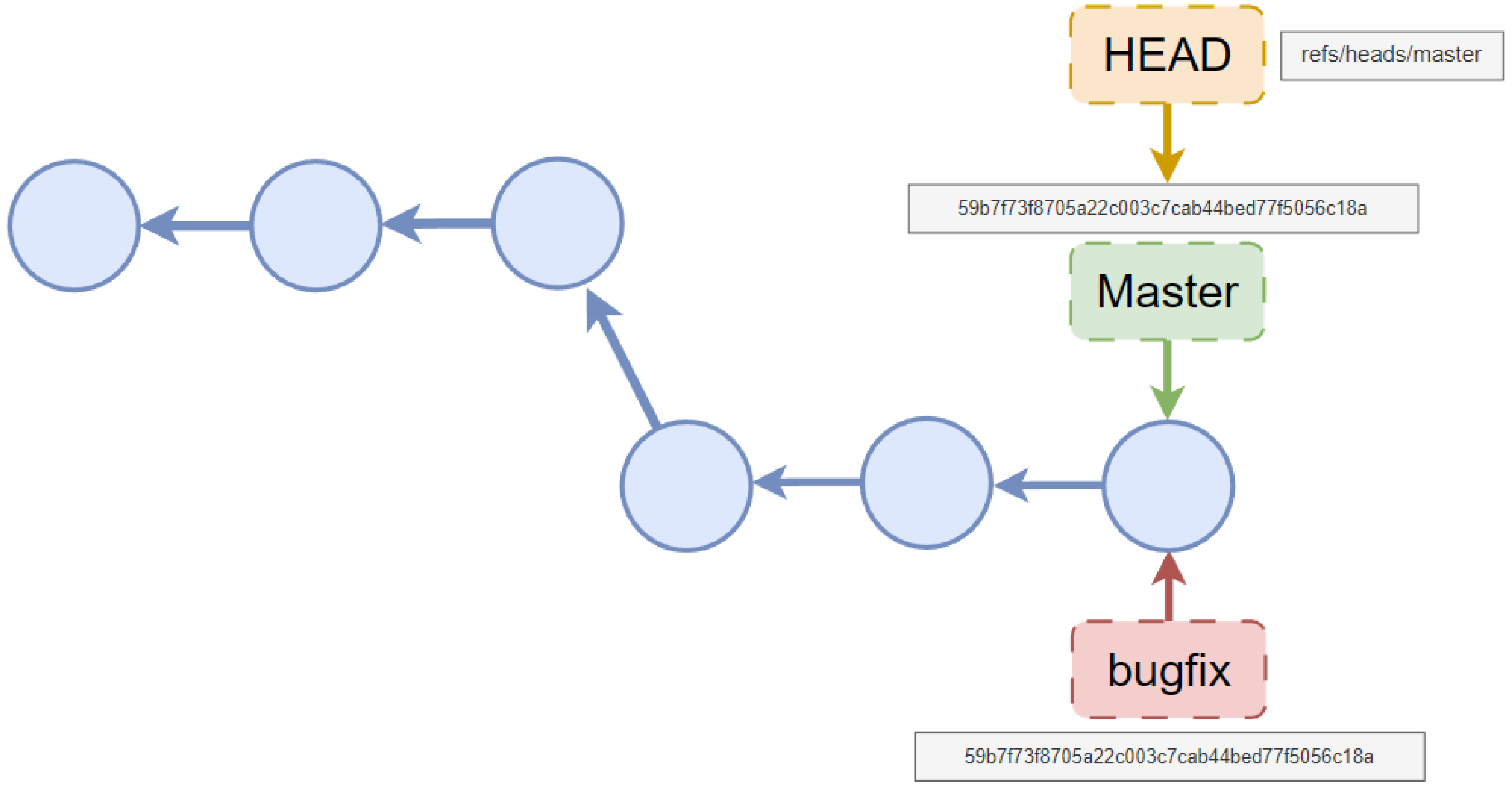
- Using rebasing you can change the base of the bugfix branch
- We will have a linear path from Master to bugfix branch.


HEAD
refs/heads/master
91a3dedd5277111bb29477d99bf27c998e119cf1
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a


HEAD
refs/heads/master
91a3dedd5277111bb29477d99bf27c998e119cf1
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

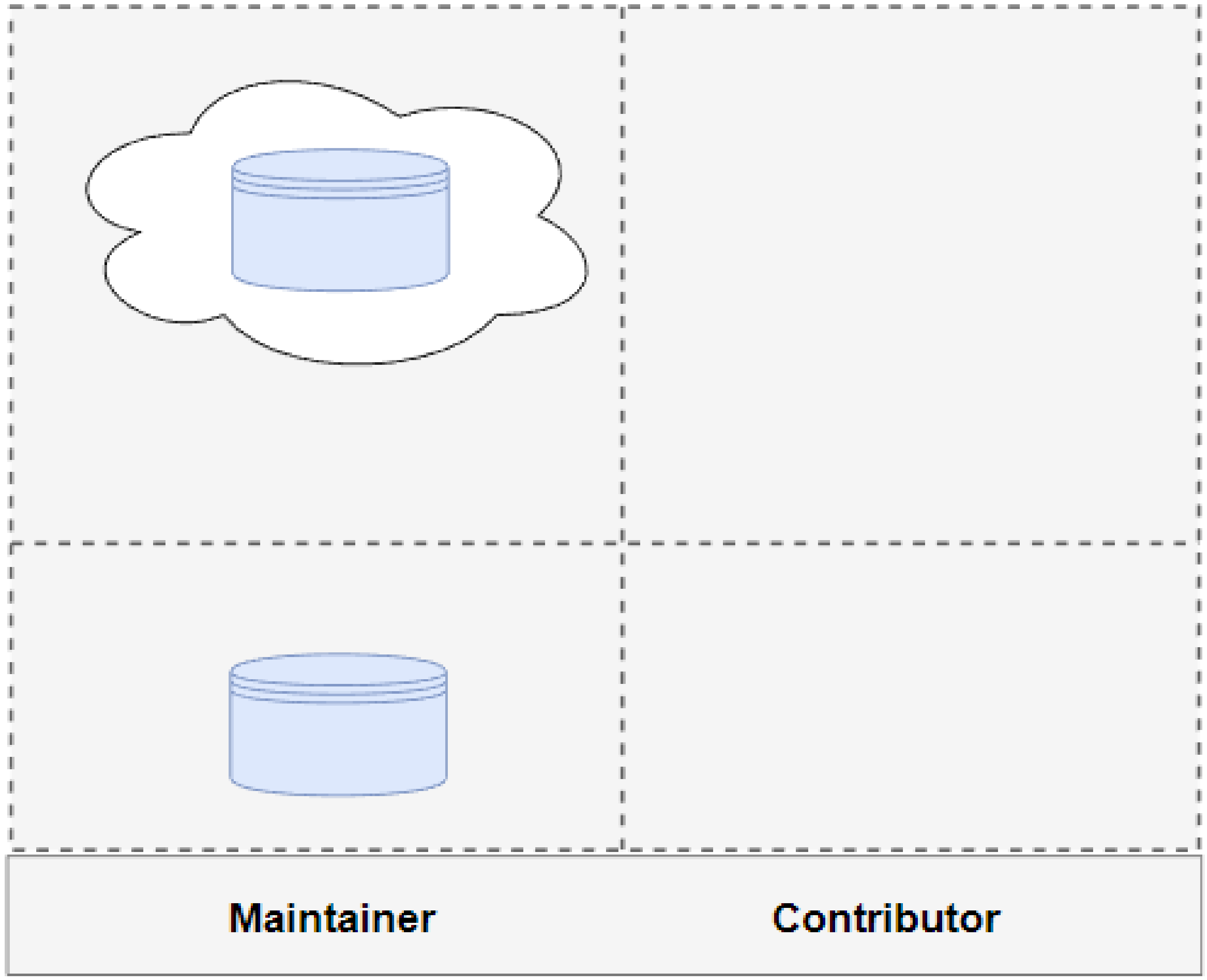
Rebasing

- Using fast-forward way to marge
- [Warning!] rebasing will rewrite the history
- You should only use it for the branches/commits that are local in your repository


HEAD
refs/heads/master
59b7f73f8705a22c003c7cab44bed77f5056c18a
Master
bugfix
59b7f73f8705a22c003c7cab44bed77f5056c18a

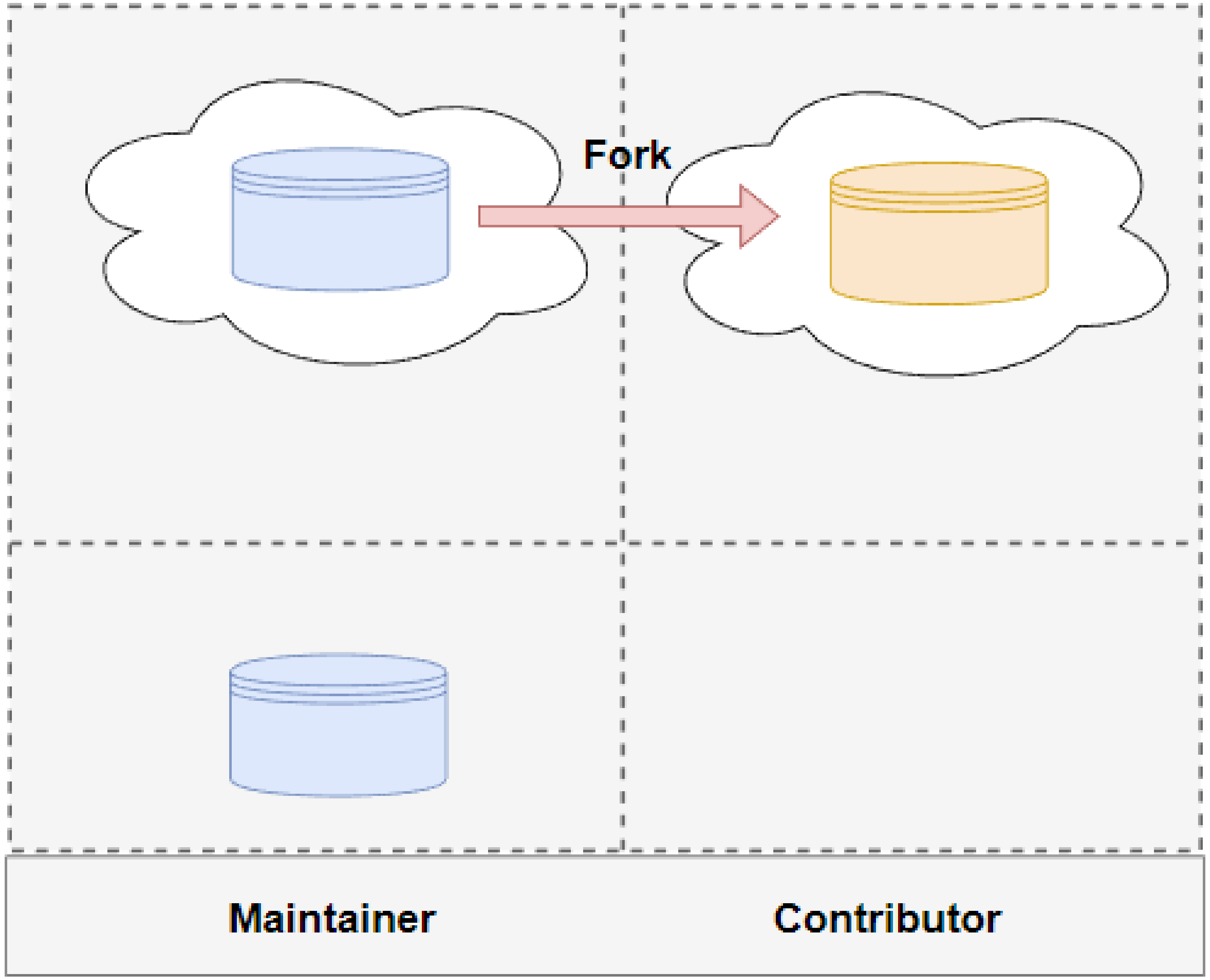
Integration Manager Workflow

- You don't know the contributors
- Only the maintainer has push access



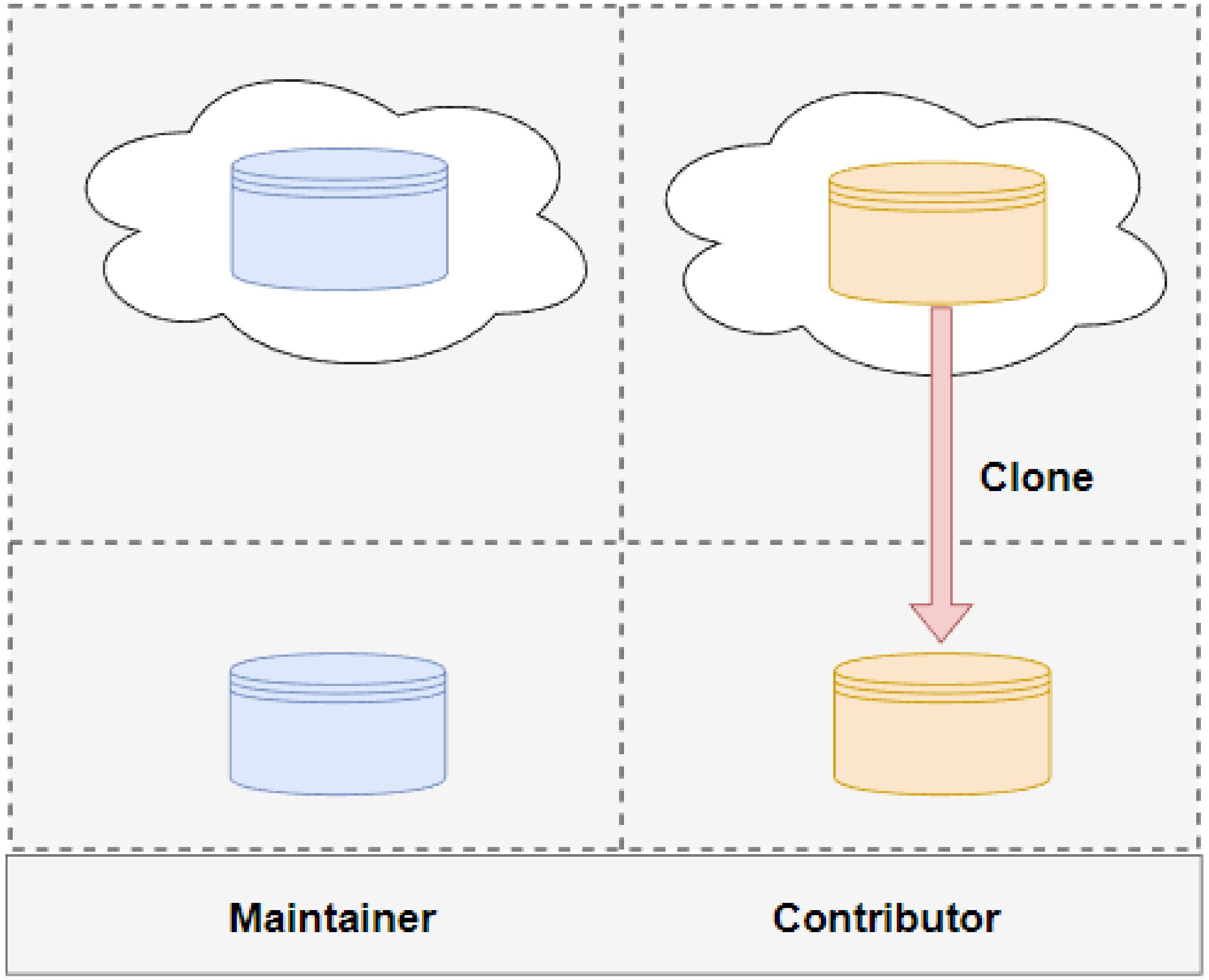
Integration Manager Workflow

- First you should fork the repository
- Then you will get a copy of this repo



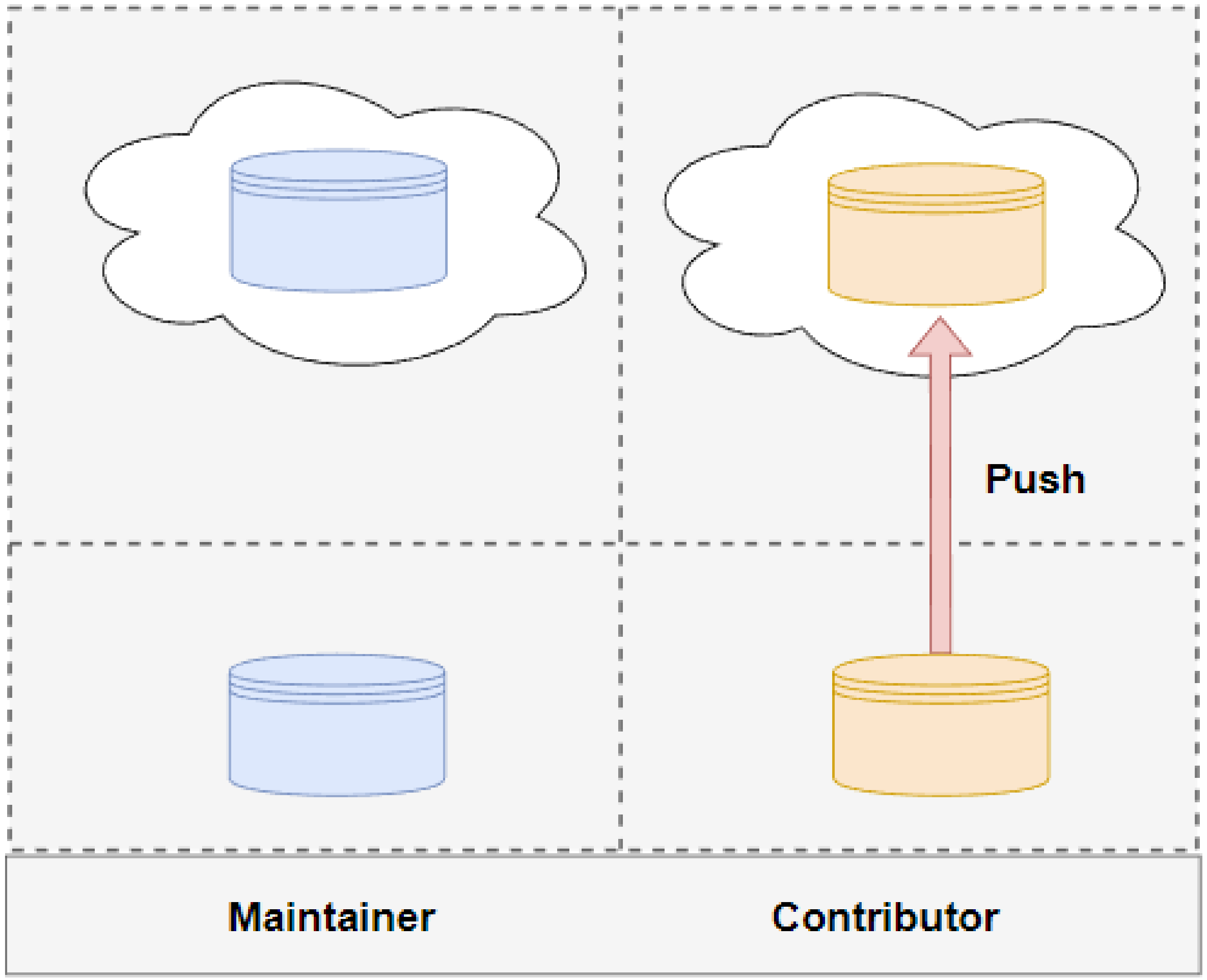
Integration Manager Workflow

- Next you clone the repository to get a local copy in your machine.



Integration Manager Workflow

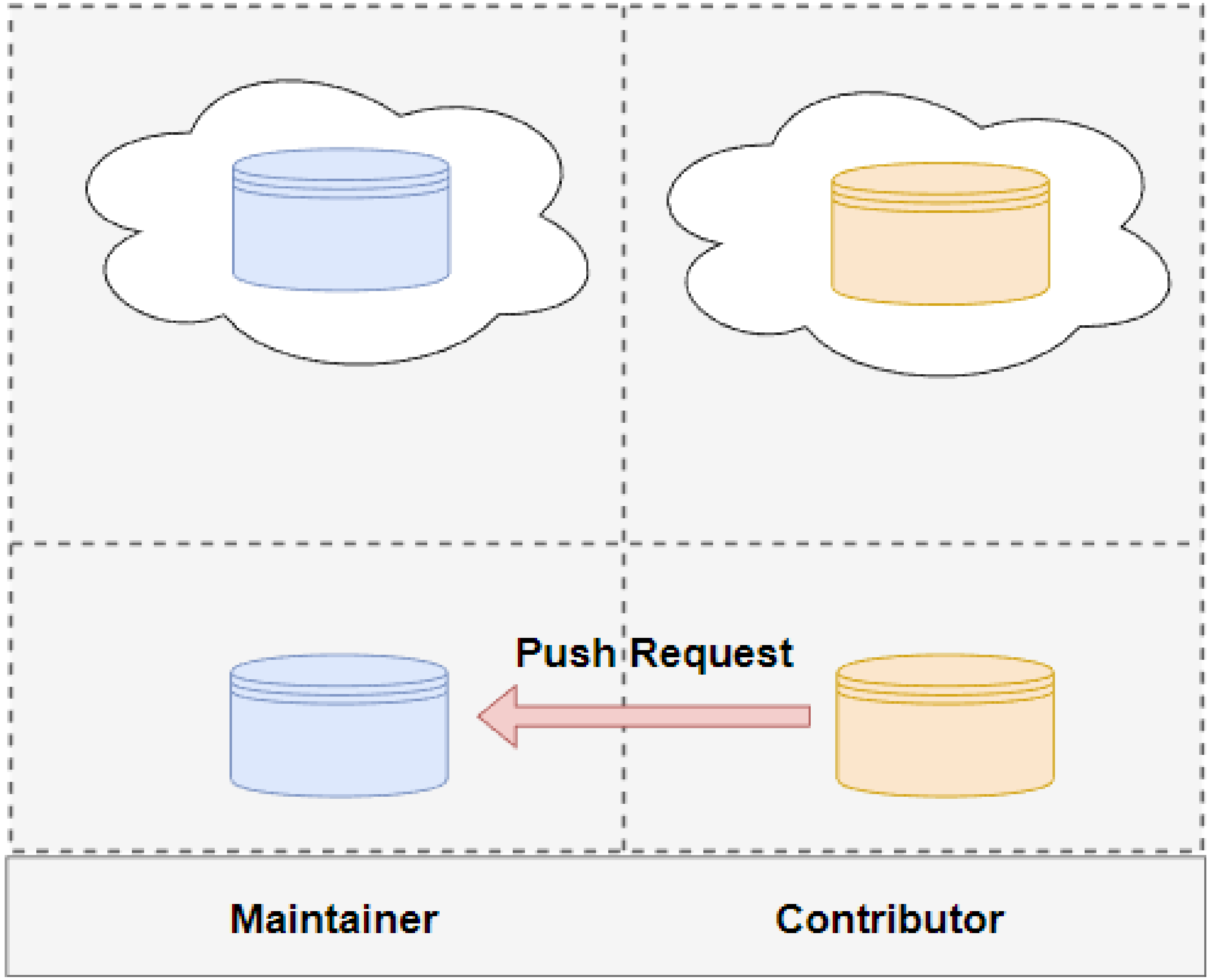
- When we are done with the changes in our local repository
- We push them to the forked repository



Integration Manager Workflow

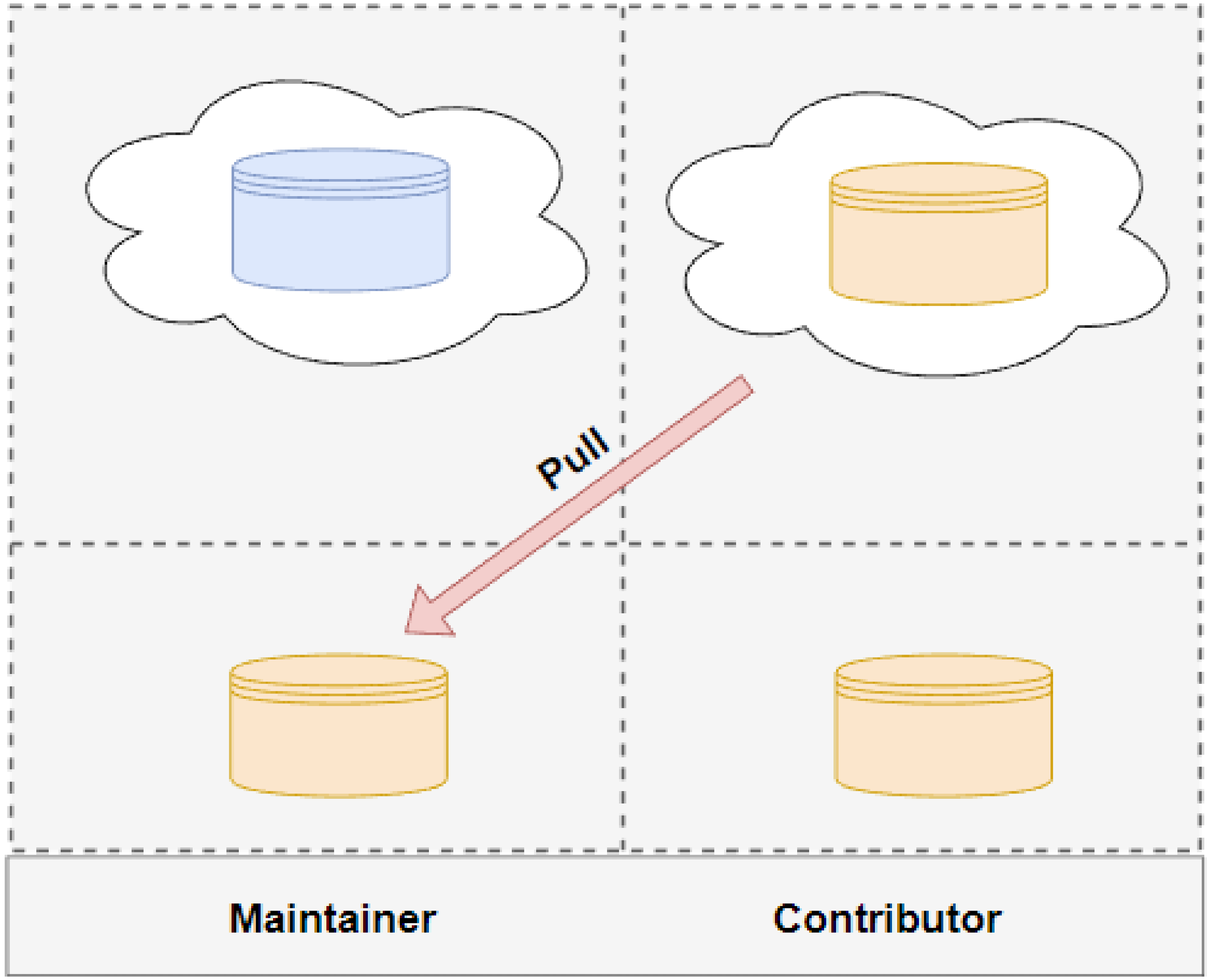
- Then we need to inform the maintainer of the project about the changes we have made.

- The maintainer gets notified



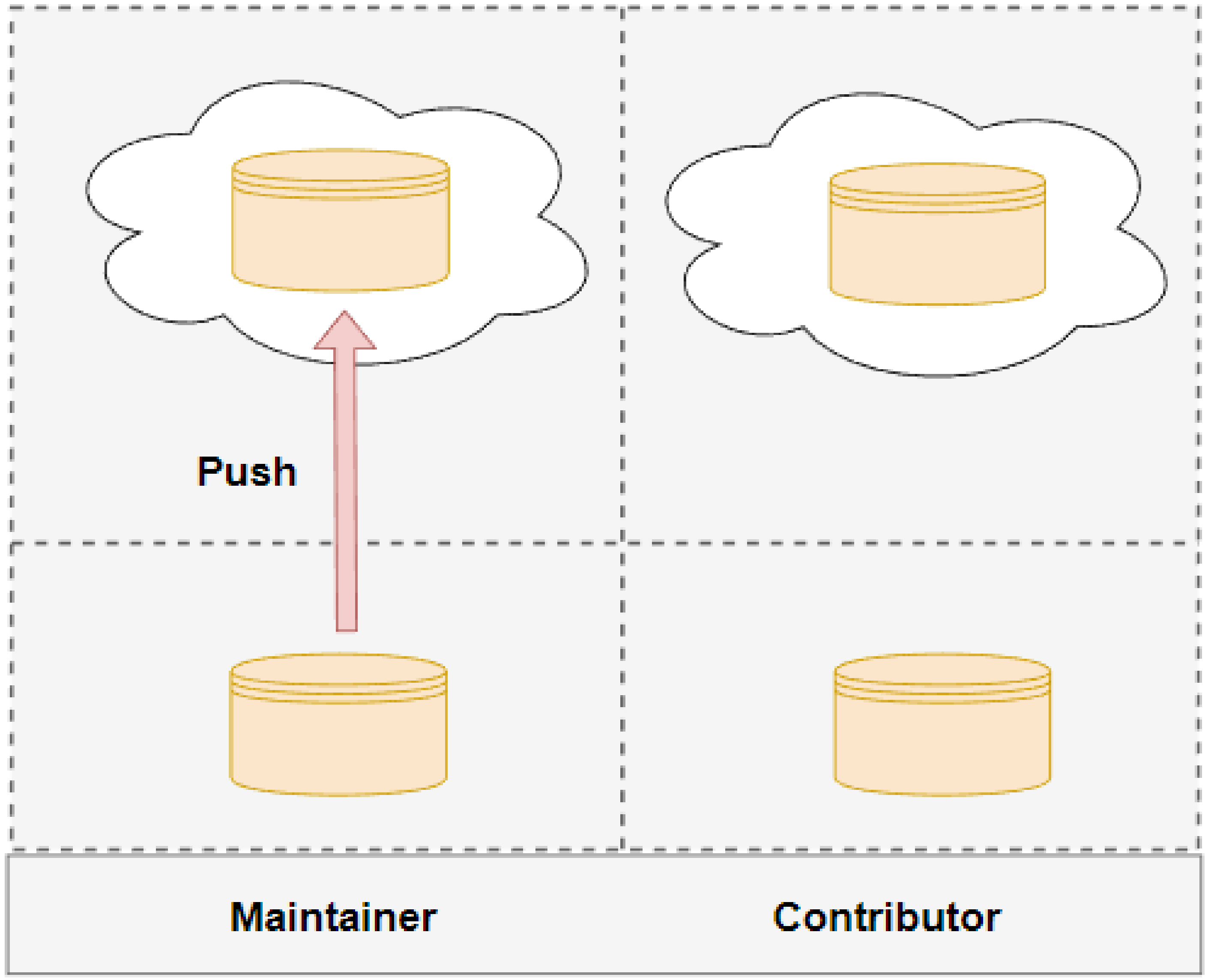
Integration Manager Workflow

- Then the maintainer will pull the changes.
- They make sure changes are OK
- If they are happy, they can merge the changes into their local repository



Integration Manager Workflow

- Finally, they can push the merged changes to the official repository of the product


Push
Maintainer
Contributor