IT Club | uxcam

GIT WORKSHOP

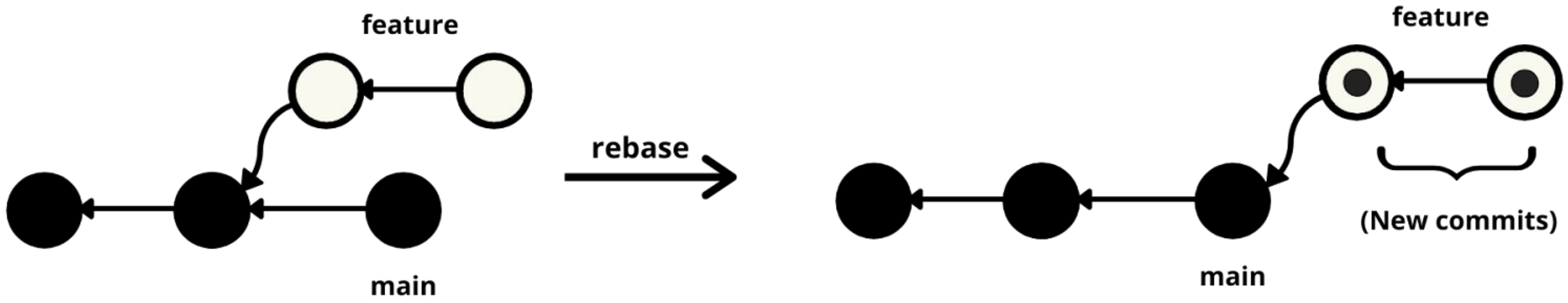
Day 3: Merge Conflicts

Branching

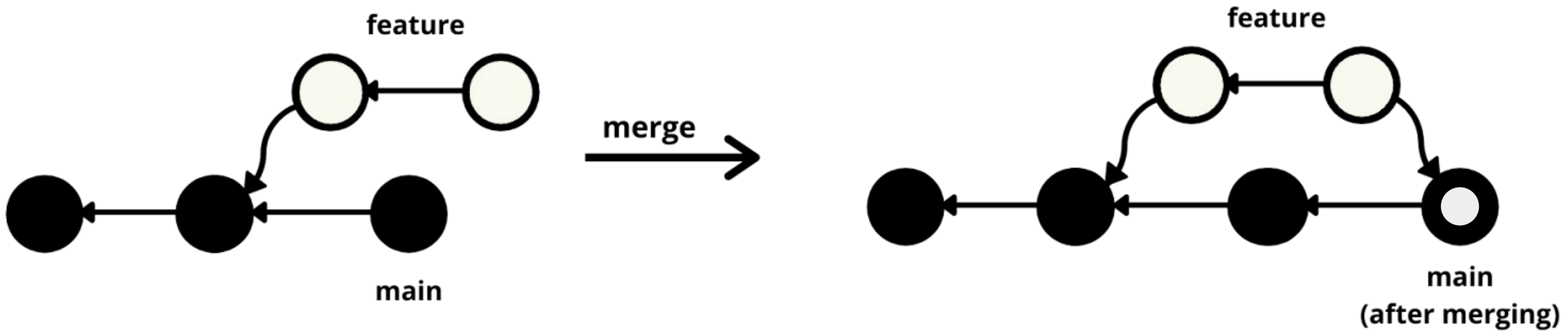
Some Leftover Topics

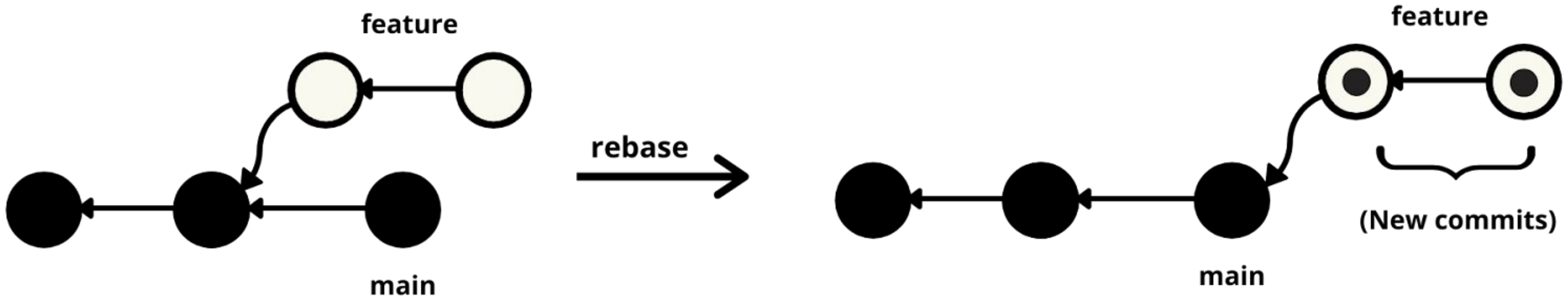
Rebasing

- Another way to integrate changes from one branch to another.
- Rewinds the head to replay your work on top of it.
- Rewrites history by creating new commits



Merge vs. Rebase





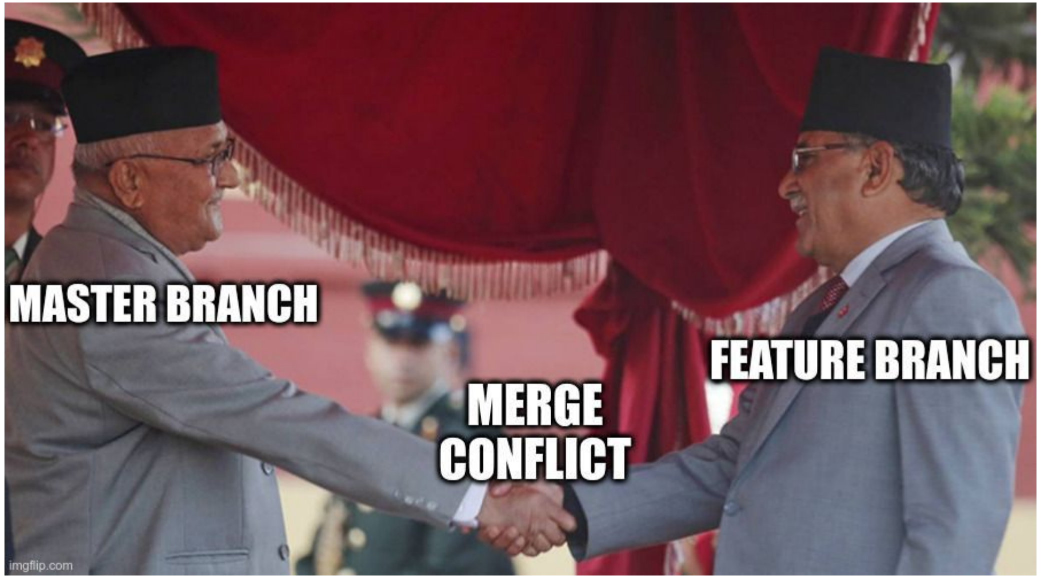
Merge Conflicts

Get Ready!

What is Merge Conflict?

What is Merge Conflict?

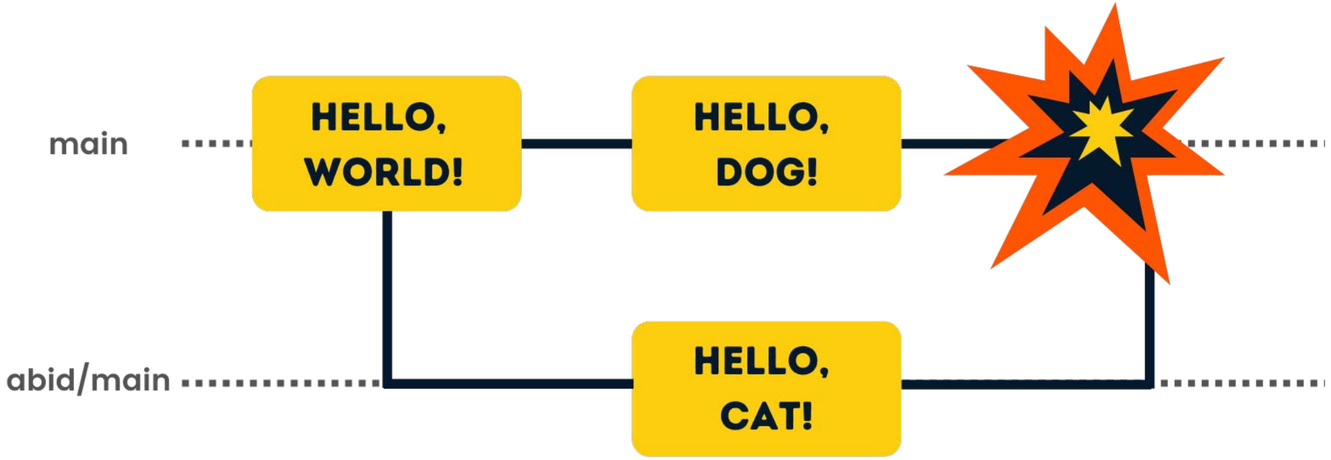
- When you are merging one branch to the other, sometimes git cannot resolve the code differences and aborts the process of auto merging.
- This is called merge conflict.



How Merge Conflict?

IT Club | uxcam

- When Performing a 3-Way Merge.
- If you change the same part of the same file differently in the two branches you're merging.

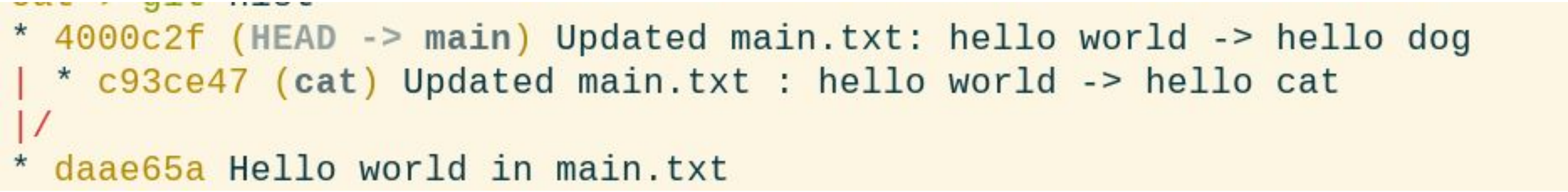


Let us Emulate a Merge Conflict

Emulating Merge Conflict

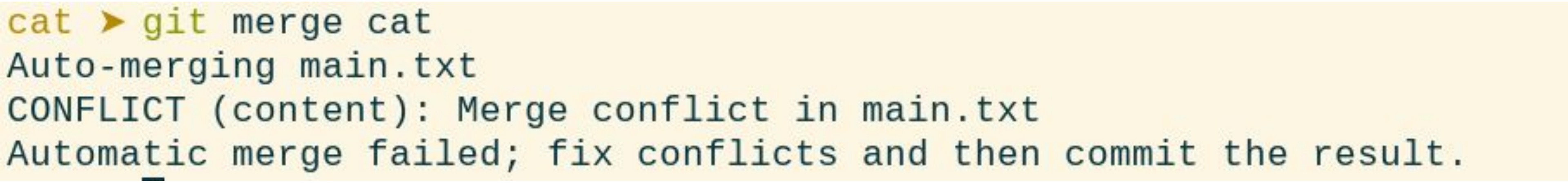
IT Club | uxcam

- Create a branched structure as below:



```
* 4000c2f (HEAD -> main) Updated main.txt: hello world -> hello dog
| * c93ce47 (cat) Updated main.txt : hello world -> hello cat
|/
* daae65a Hello world in main.txt
```

- Try to merge **cat** to **main** and Boom! You have a **Merge Conflict**.



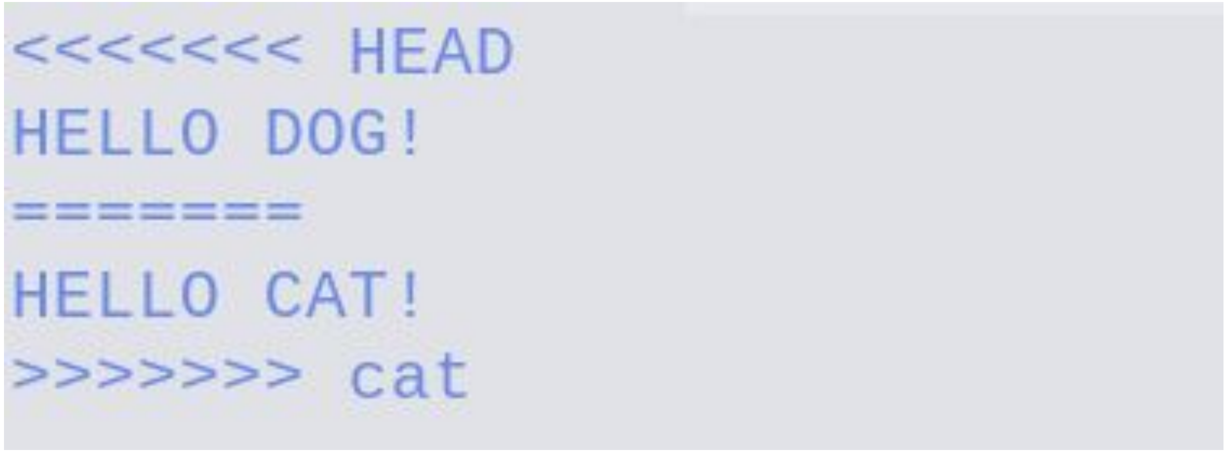
```
cat ➤ git merge cat
Auto-merging main.txt
CONFLICT (content): Merge conflict in main.txt
Automatic merge failed; fix conflicts and then commit the result.
```

How to resolve Conflicts?

Resolving Merge Conflict

IT Club | uxcam

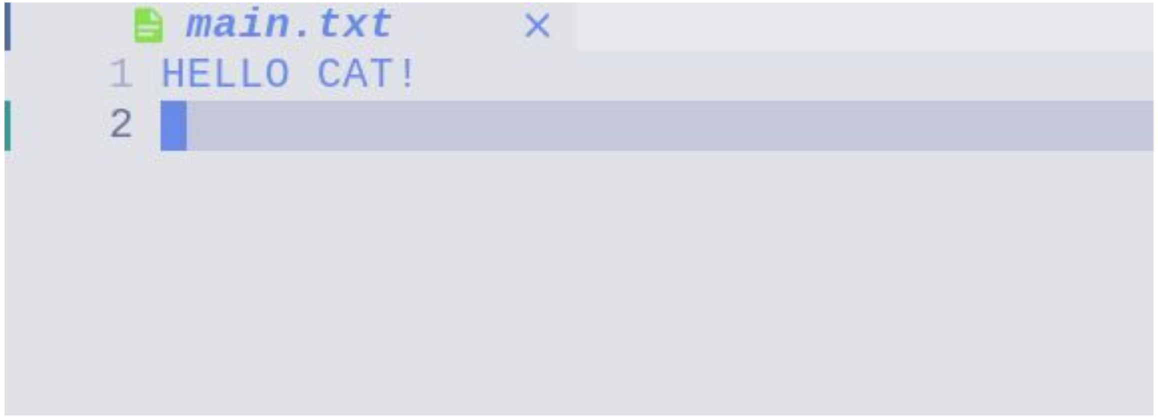
- Git adds standard conflict-resolution markers to the files that have conflicts.
- So, files with conflicts can be opened manually and conflicts can be resolved using a text editor of choice.



```
<<<<<<< HEAD
HELLO DOG!
=======
HELLO CAT!
>>>>>>> cat
```

Resolving Merge Conflict

- To resolve, simply keep the section you want and delete the conflicting section.

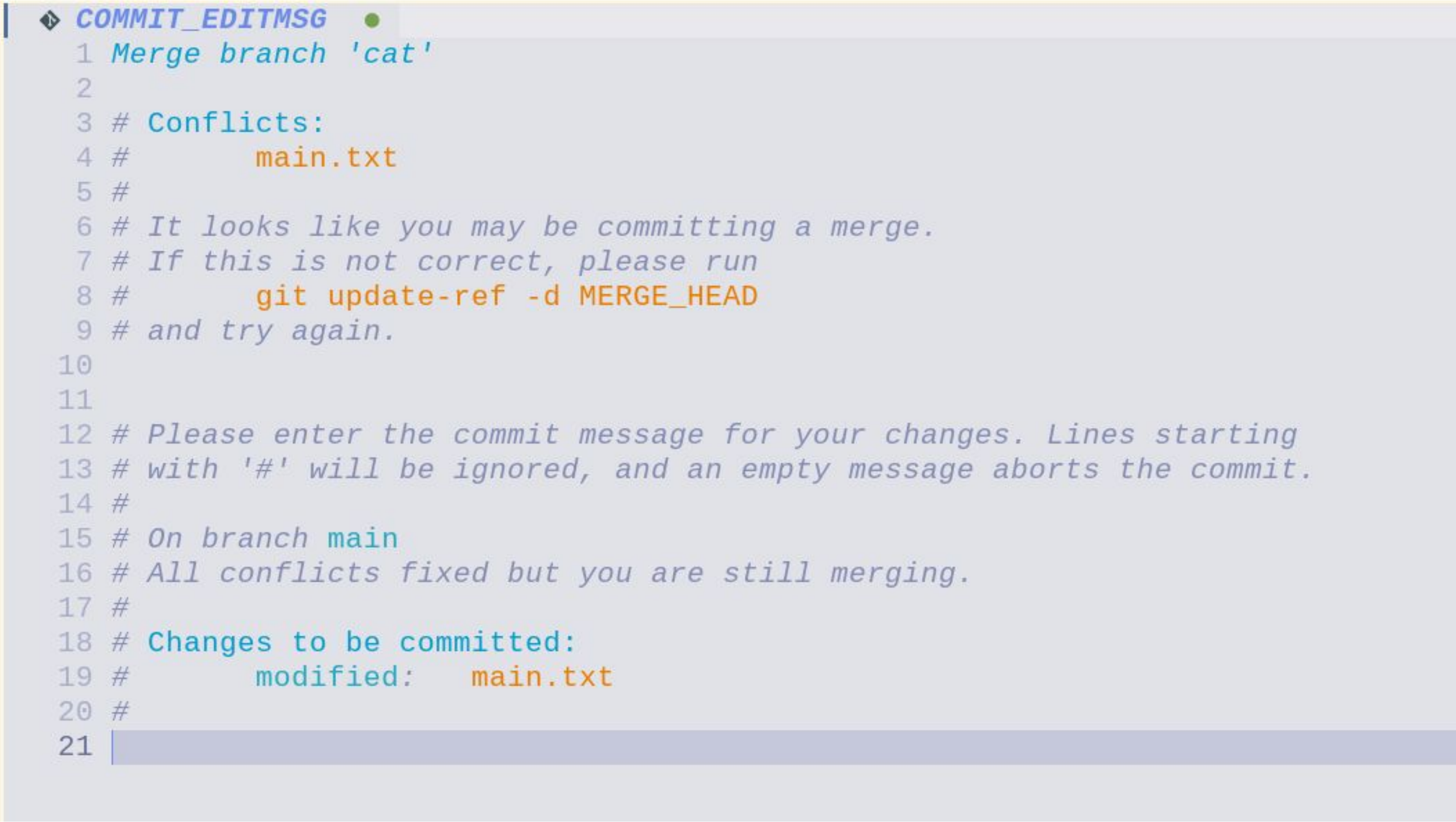


main.txt

```
HELLO CAT!

```

- Then just commit your changes using:

git commit -a


COMMIT_EDITMSG

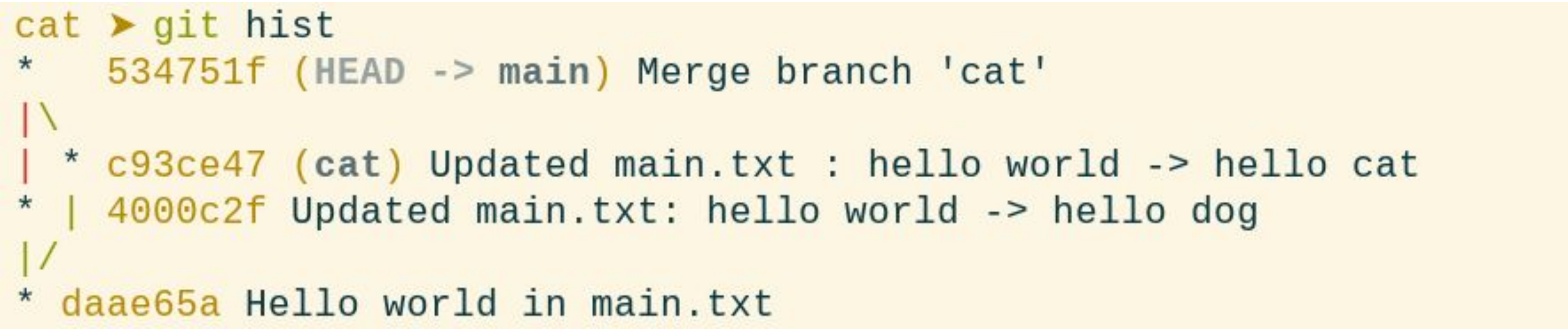
```
Merge branch 'cat'

# Conflicts:
#	main.txt
#
# It looks like you may be committing a merge.
# If this is not correct, please run
#	git update-ref -d MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#	modified:   main.txt
#

```

Let’s look at the tree.

Resolving Merge Conflict

IT Club | uxcam

- The tree looks like the following after resolving the conflict.



```
cat ➤ git hist
*   534751f (HEAD -> main) Merge branch 'cat'
|\
| * c93ce47 (cat) Updated main.txt : hello world -> hello cat
* | 4000c2f Updated main.txt: hello world -> hello dog
|/
* daae65a Hello world in main.txt
```

Using GUI tools to resolve merge conflicts.

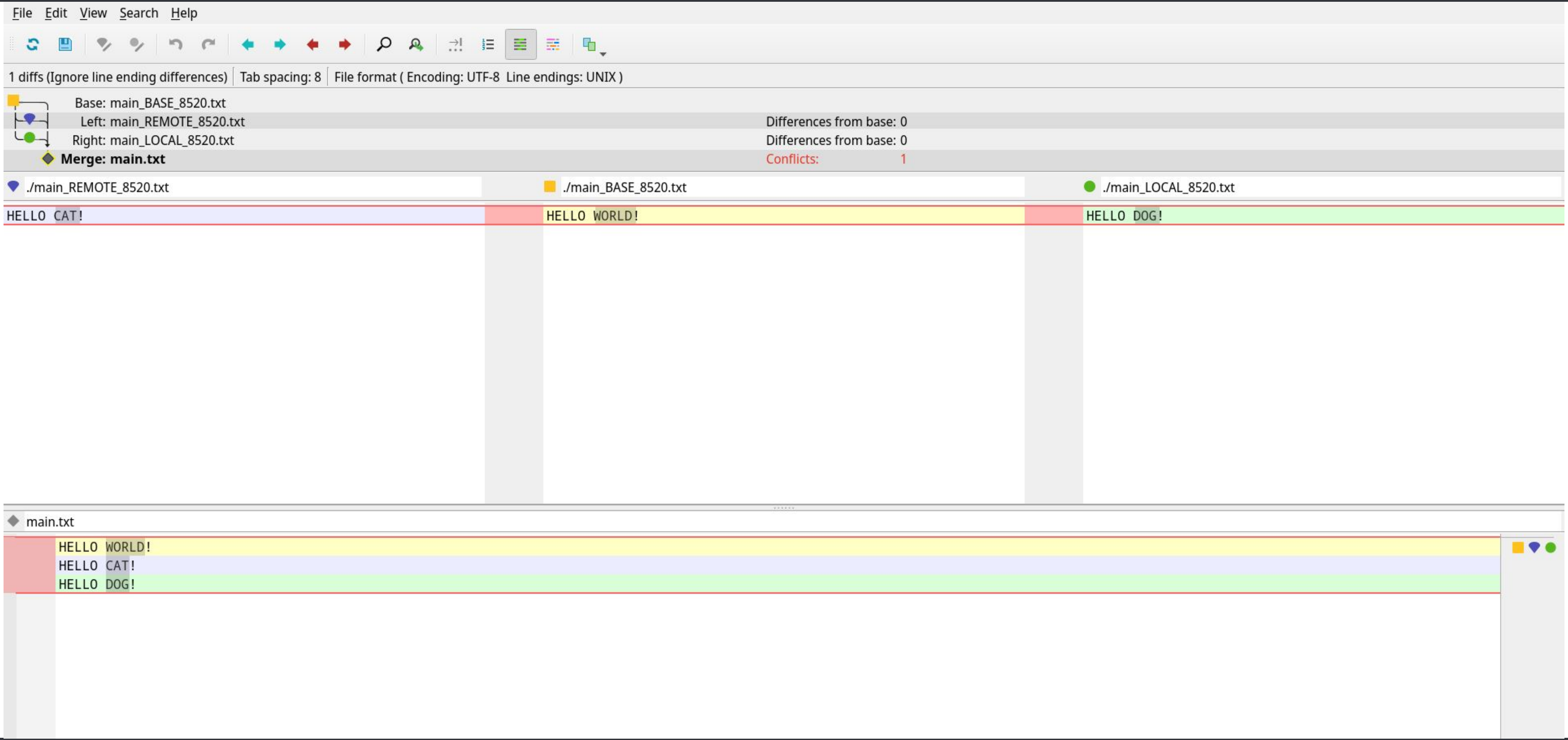
Resolving using GUI Tools

- To list available tools:

```
git mergetool --tool-help
```

Resolving using GUI Tools

IT Club | uxcam



1 diffs (Ignore line ending differences) | Tab spacing: 8 | File format (Encoding: UTF-8 Line endings: UNIX)

Base: main_BASE_8520.txt
Left: main_REMOTE_8520.txt — Differences from base: 0
Right: main_LOCAL_8520.txt — Differences from base: 0
Merge: main.txt — Conflicts: 1

./main_REMOTE_8520.txt	./main_BASE_8520.txt	./main_LOCAL_8520.txt
HELLO CAT!	HELLO WORLD!	HELLO DOG!

main.txt

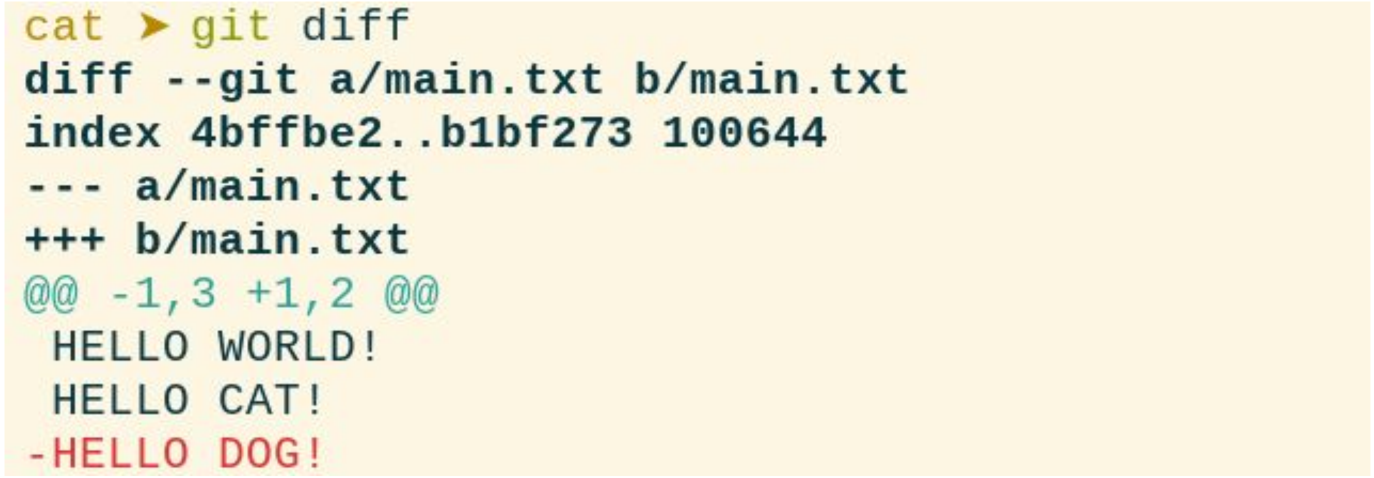
```
HELLO WORLD!
HELLO CAT!
HELLO DOG!
```

Revisiting some Commands

Remember ‘git diff’ ?

git diff

- This command is used to show the difference between different data sources.
- Different data sources includes: commits, branches, files and more.

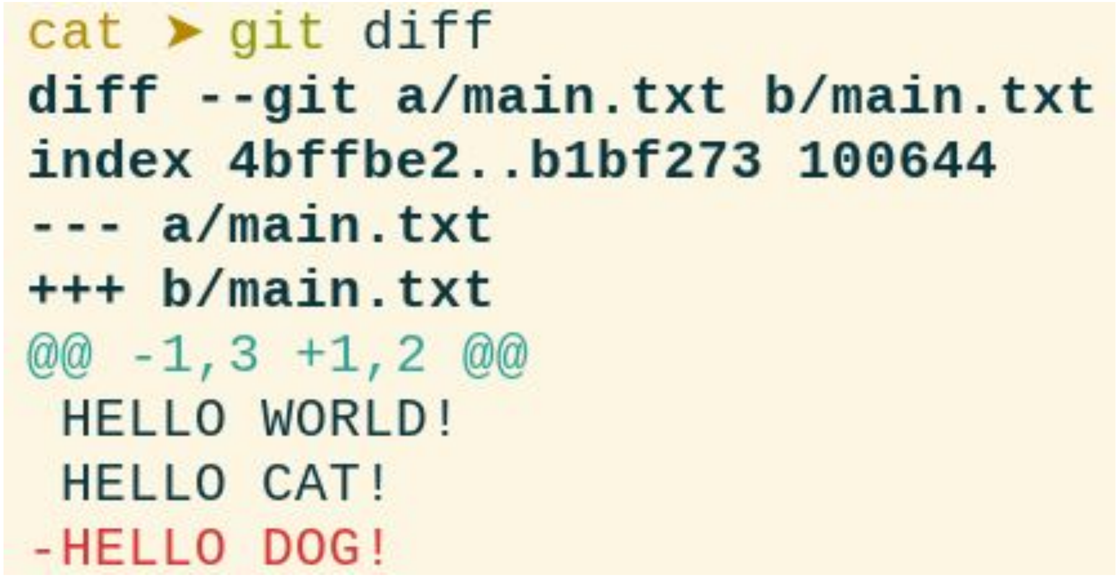


```
cat ➤ git diff
diff --git a/main.txt b/main.txt
index 4bffbe2..b1bf273 100644
--- a/main.txt
+++ b/main.txt
@@ -1,3 +1,2 @@
 HELLO WORLD!
 HELLO CAT!
-HELLO DOG!
```

git diff

Some details provided by the **git diff** command

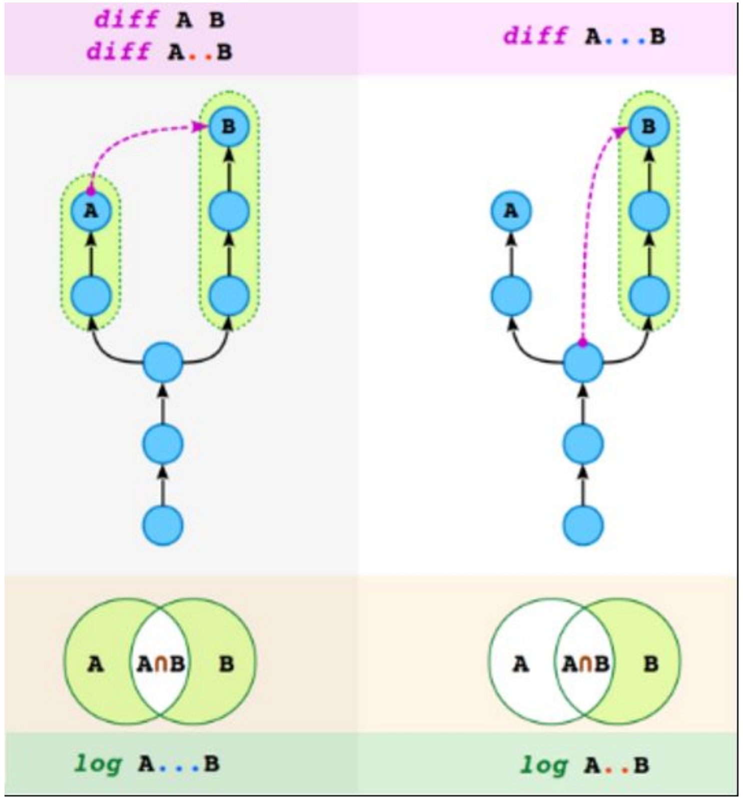
- Line 1: Compared files a/b
- Line 2: File Metadata
- Line 3: Legend
- Line 4: Chunk header.
- Line 5: Changes.



```
cat ➤ git diff
diff --git a/main.txt b/main.txt
index 4bffbe2..b1bf273 100644
--- a/main.txt
+++ b/main.txt
@@ -1,3 +1,2 @@
 HELLO WORLD!
 HELLO CAT!
-HELLO DOG!
```

git diff

Difference between the branches.



Moving around commits.

Moving around commits

Some Basic Terms:

REFS

- Is an indirect way of referring to a commit.
- User friendly alias for commit hash.

HEAD

- It points to the most recent commit reflected in the working tree.

RELATIVE REFS

- We can use relative position between branch pointer or **HEAD** to move around different commits with the help of what is known as relative references.

Moving around commits

IT Club | uxcam

Commands and Operators used:

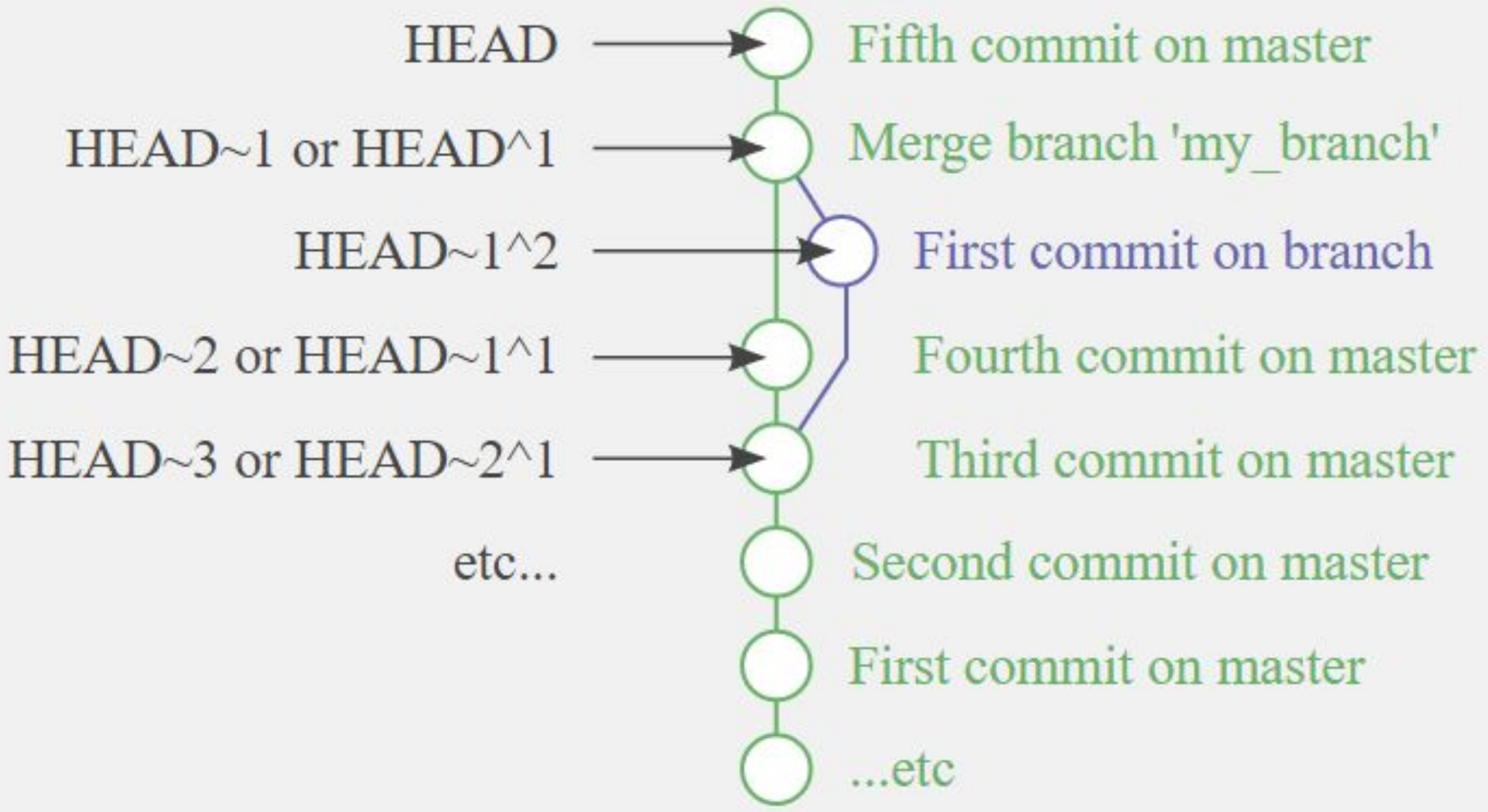
- **git checkout** : To between branches as well as commits.
- ^ **operator** specifies the **parent** of the current ref.
- ^^ **operator** specifies the **grandparent** of the current ref.
- **~<num>** specifies the ref '**num**' number of nodes above the tree.

Note:

After checking out to some commit, **HEAD** changes to the said commit instead of main(master).

This process is called **detaching HEAD**.

Referencing commits from HEAD using ~ and ^



What can we do with this ?

Moving around commits

- We can move to a different commit and then check the contents of file and activities of that commit.
- We can restore some file from the last commit.

```
git checkout HEAD~2 index.html
```

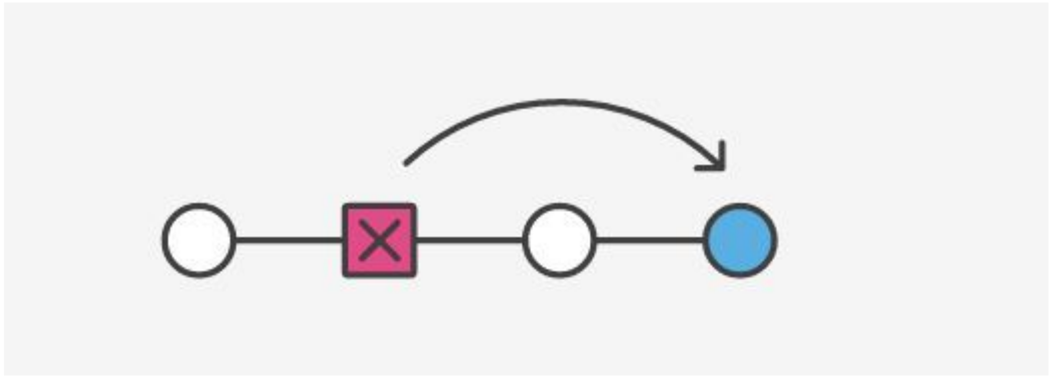
- Here, index.html file is retrieved from grandparent of HEAD. (be careful)

Reverting

Git Revert is used when you want to revert the changes of specific commit in the history without affecting the commits after it.

- Git checkout and git reset move the **HEAD** and **branch ref pointers** to a specific commit.
- But revert **does not** move ref pointers to this commit.
- History is kept intact

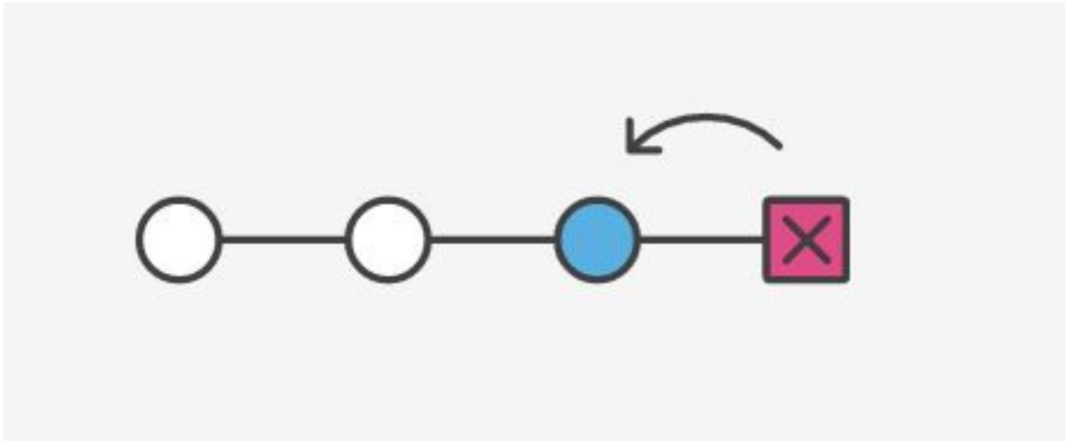
git revert <commit_hash>



Resetting

Reset is used when you want to go back to a specific snapshot and continue from there.

(Subsequent commits are lost)



Resetting

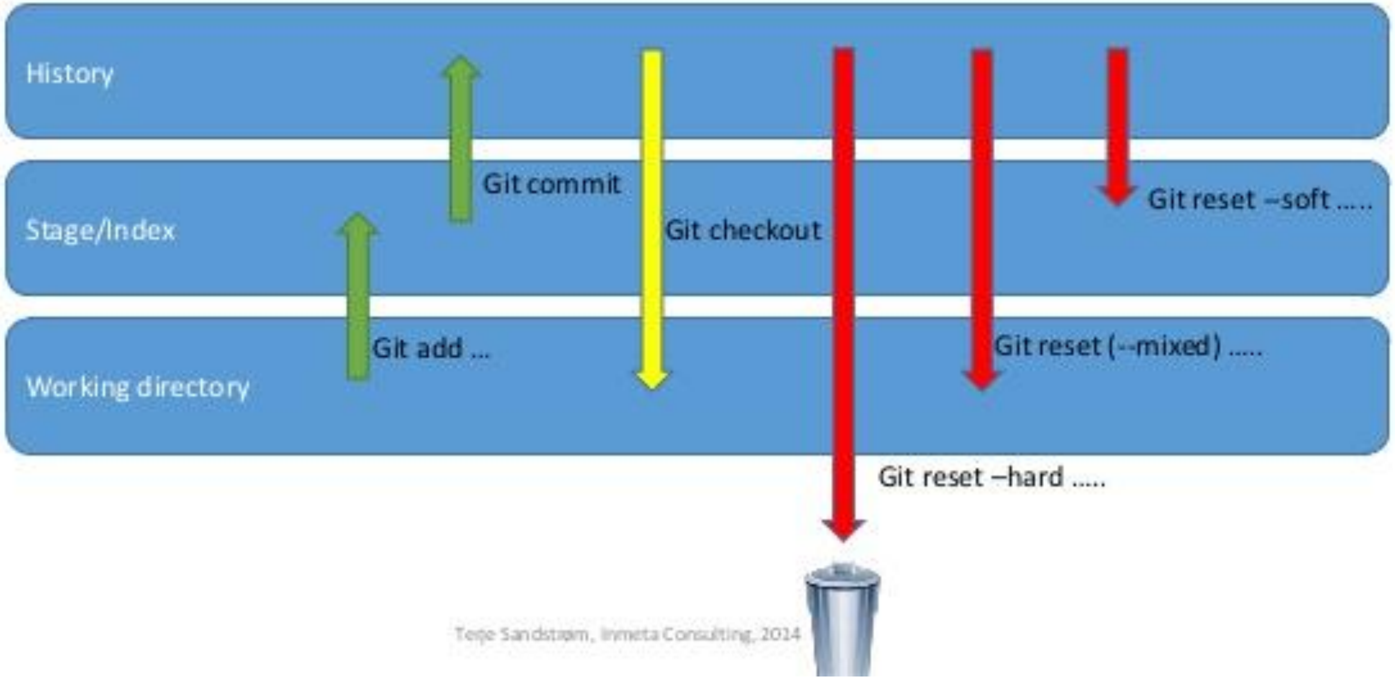
Types of resets

1. **Soft Reset**
   Removes the commit but the files are at the staging areas.
   ```
   $ git reset --soft <commit_hash>
   ```
2. **Mixed Reset**
   This is the default reset mode. Removes the staged files from staging to working area.
   ```
   $ git reset <commit_hash>
   ```
3. **Hard Reset** (Dare to use it)
   Completely resets to the previous commit. Rare example where git could actually make data unrecoverable.
   ```
   $ git reset --hard <commit_hash>
   ```

Using commands like git reflog we can get back previous commits.

Git Tree Movement Visualization

IT Club | uxcam



Amending/Undoing Things

One of the common undos takes place when you commit too early and possibly forget to add some files or you mess up your commit message.

```
$ git commit --amend
```

- This command takes your staging area and uses it for the commit.
- If no file staged then the staging area is empty ,so only commit msg is changed.

Stashing : She is a beauty!

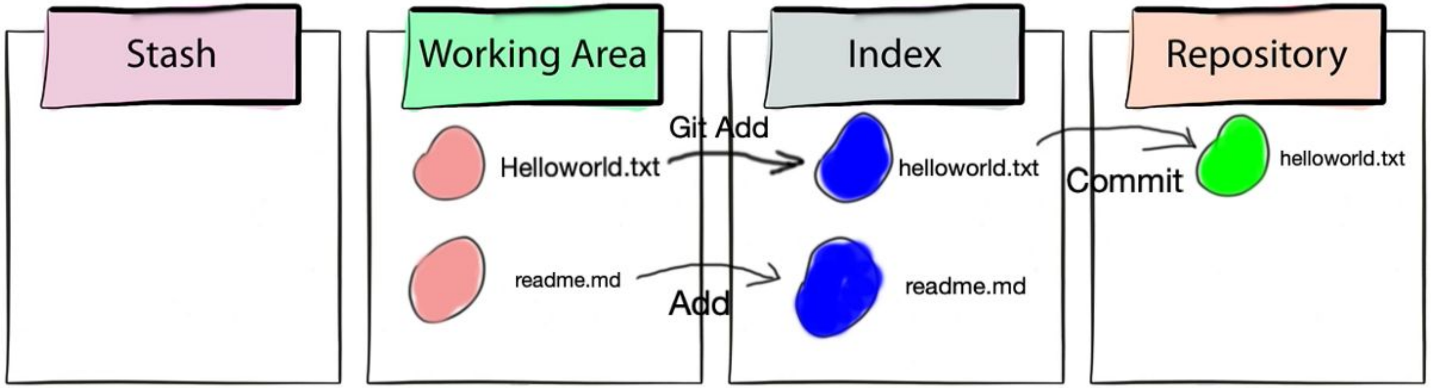
- Whenever you are working on a feature but you are not yet ready for commit as you don’t want to do a commit of half-done work just so you can get back to this point later.

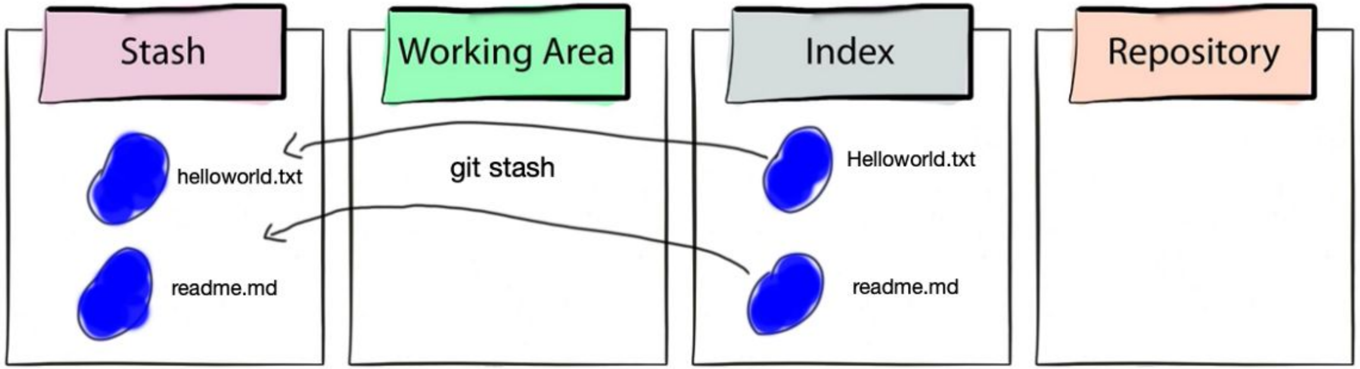
- Then you have stash to the rescue.

- Now you can switch branches for a bit to work on something else.

Stashing

IT Club | uxcam





Stashing

IT Club | uxcam

git stash [-u | -a] -> general stash command

git stash save "message" -> stash with messages

git stash list -> show the list of all stashes (it is a stack)

git stash apply -> apply the stash at the top of stack

git stash apply | pop <ref> -> apply a particular stash

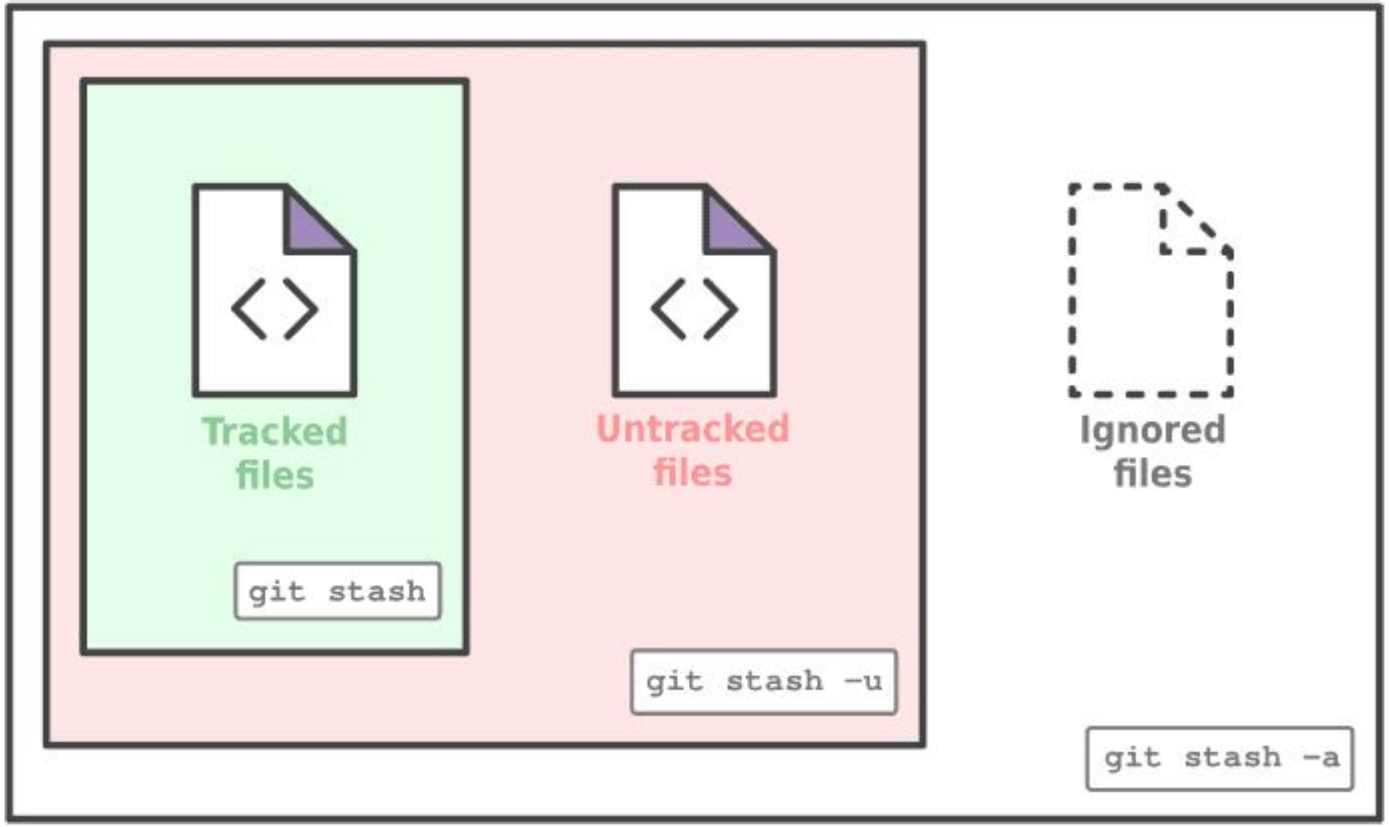
git stash pop -> apply a stash and remove it from list

git stash clear -> remove all stashes

<u>`Note: Merge conflict can occur even in stashing. Think about it.`</u>

IT Club | uxcam

Stashing



Time for some demonstrations!

Questions?