CMSC388T

Messing Up On Git

Today's Lecture:

1 **More Git Commands**

Useful Git commands if you mess up

2 **Reverting another team's mistakes**

Demo of git reset and an introduction to git revert

3 **Advanced Git**

More useful Git commands

More Git Commands

Useful Git commands if you mess up

git alias

You can set up an alias for each command using git config.

For example:

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

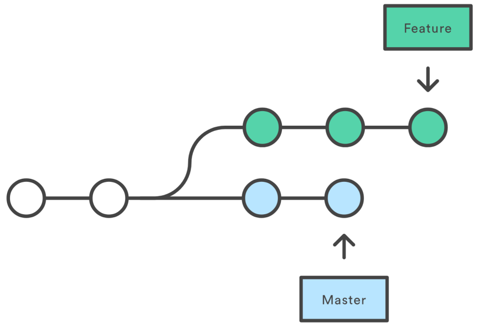
git alias

```
git config --global alias.lg1  "log --all --graph --decorate
--oneline"
```

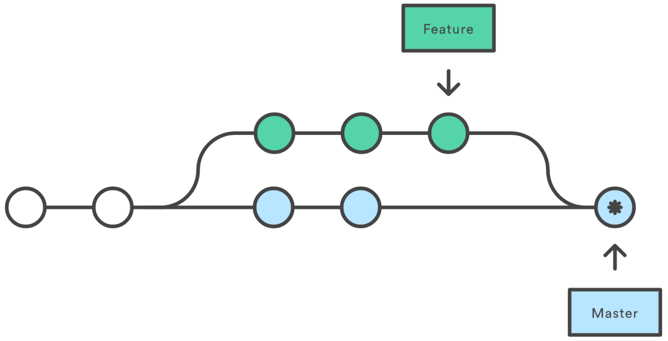
```
[ ~ ]git lg1
* 9f0dbb2 (HEAD -> main) adds main
* 60e6e78 adds 5
| * 9224164 (feature) adds feat 2
| * 283e527 adds feat 1
|/
* 19e506b adds 2
* 7d58006 adds 1
```

git rebase

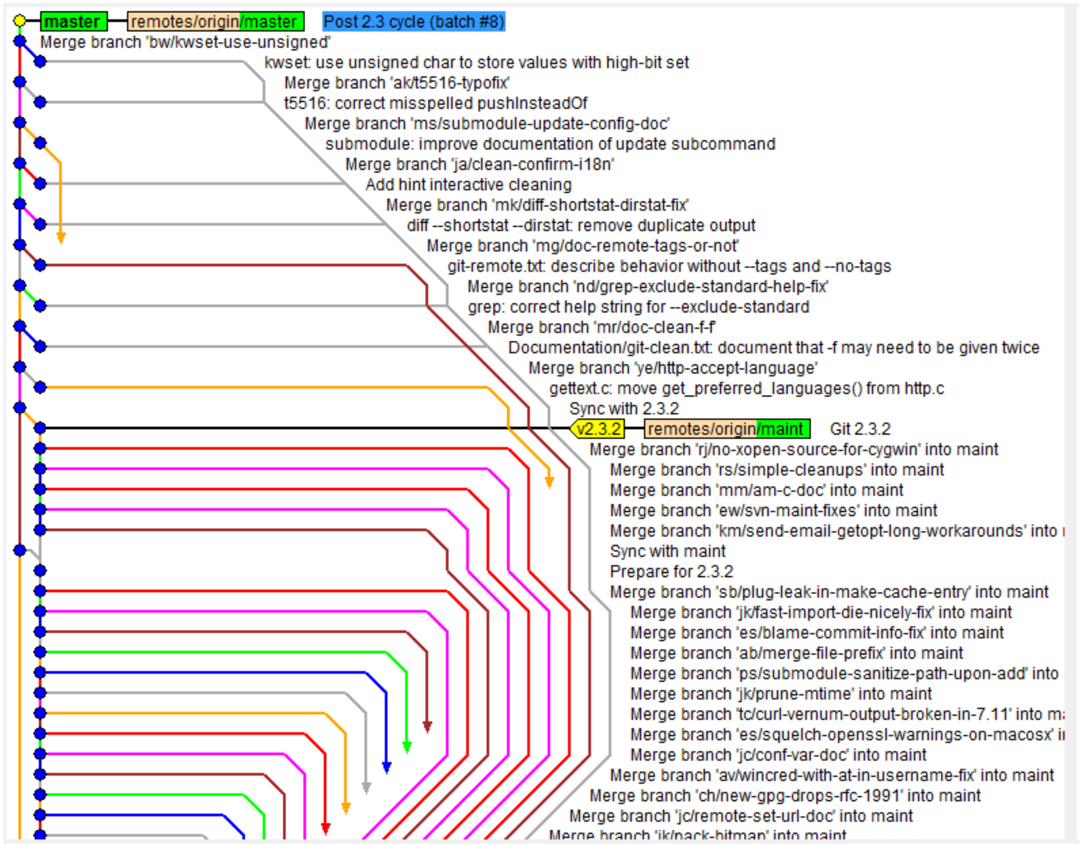
Git merge vs git rebase



```
git checkout feature
git merge master
```



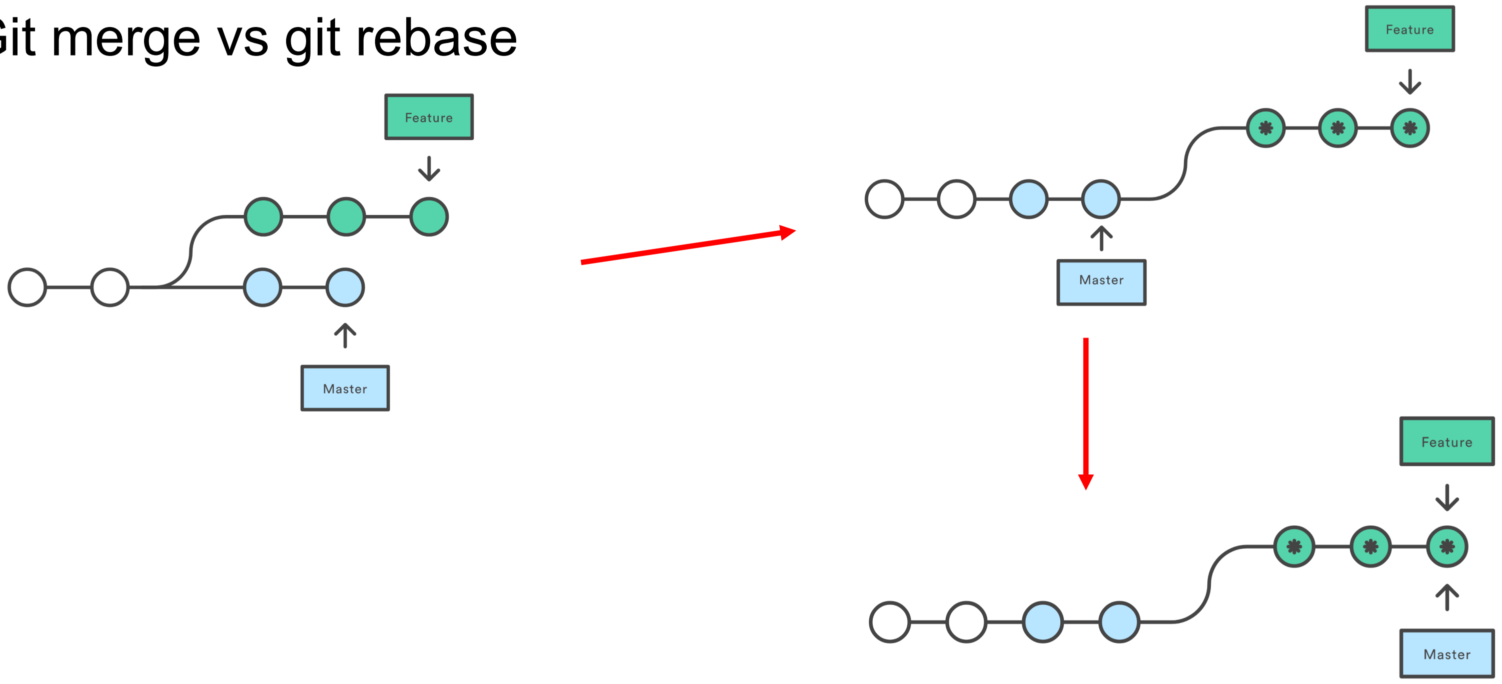
Git merge Log



```
master  remotes/origin/master  Post 2.3 cycle (batch #8)
Merge branch 'bw/kwset-use-unsigned'
kwset: use unsigned char to store values with high-bit set
Merge branch 'ak/t5516-typofix'
t5516: correct misspelled pushInsteadOf
Merge branch 'ms/submodule-update-config-doc'
submodule: improve documentation of update subcommand
Merge branch 'ja/clean-confirm-i18n'
Add hint interactive cleaning
Merge branch 'mk/diff-shortstat-dirstat-fix'
diff --shortstat --dirstat: remove duplicate output
Merge branch 'mg/doc-remote-tags-or-not'
git-remote.txt: describe behavior without --tags and --no-tags
Merge branch 'nd/grep-exclude-standard-help-fix'
grep: correct help string for --exclude-standard
Merge branch 'mr/doc-clean-f-f'
Documentation/git-clean.txt: document that -f may need to be given twice
Merge branch 'ye/http-accept-language'
gettext.c: move get_preferred_languages() from http.c
Sync with 2.3.2
v2.3.2  remotes/origin/maint  Git 2.3.2
Merge branch 'rj/no-xopen-source-for-cygwin' into maint
Merge branch 'rs/simple-cleanups' into maint
Merge branch 'mm/am-c-doc' into maint
Merge branch 'ew/svn-maint-fixes' into maint
Merge branch 'km/send-email-getopt-long-workarounds' into
Sync with maint
Prepare for 2.3.2
Merge branch 'sb/plug-leak-in-make-cache-entry' into maint
Merge branch 'jk/fast-import-die-nicely-fix' into maint
Merge branch 'es/blame-commit-info-fix' into maint
Merge branch 'ab/merge-file-prefix' into maint
Merge branch 'ps/submodule-sanitize-path-upon-add' into
Merge branch 'jk/prune-mtime' into maint
Merge branch 'tc/curl-vernum-output-broken-in-7.11' into m
Merge branch 'es/squelch-openssl-warnings-on-macosx' i
Merge branch 'jc/conf-var-doc' into maint
Merge branch 'av/wincred-with-at-in-username-fix' into maint
Merge branch 'ch/new-gpg-drops-rfc-1991' into maint
Merge branch 'jc/remote-set-url-doc' into maint
Merge branch 'jk/pack-bitmap' into maint
```

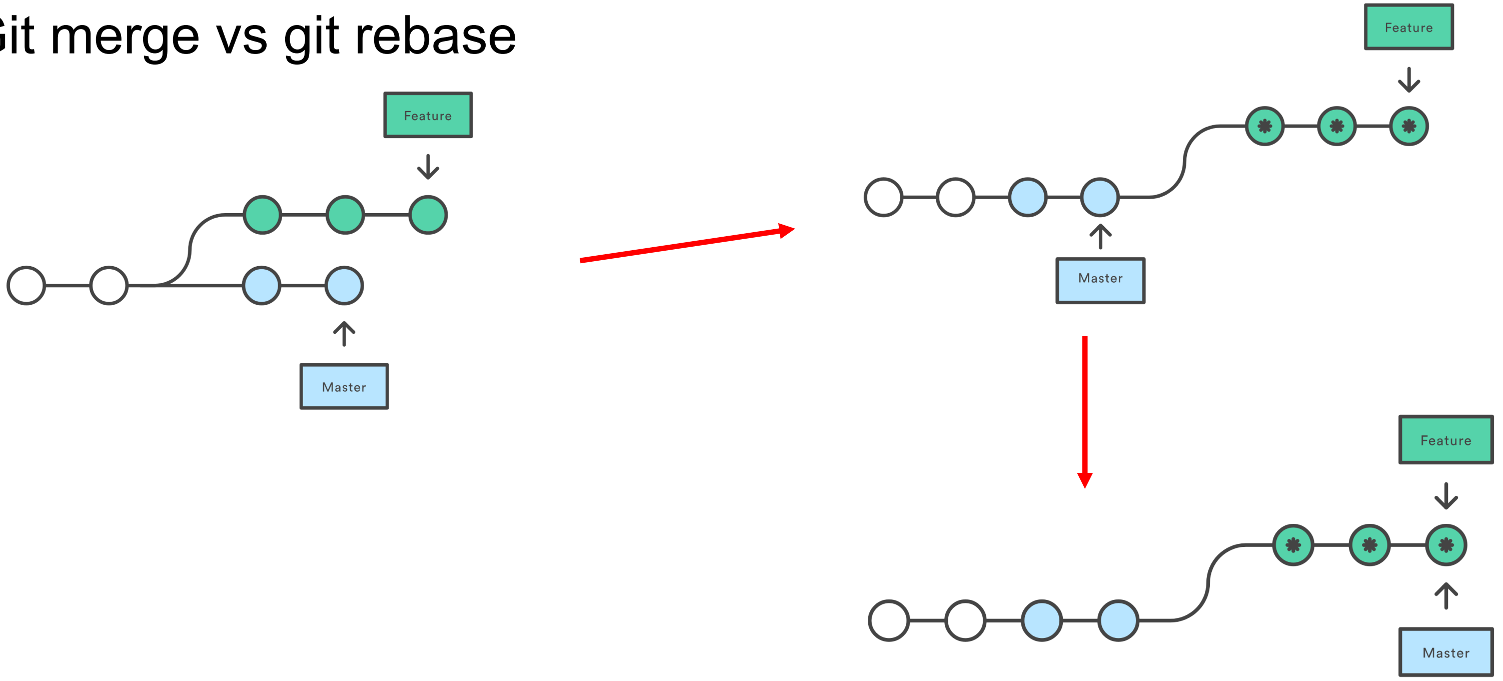
git rebase

Git merge vs git rebase



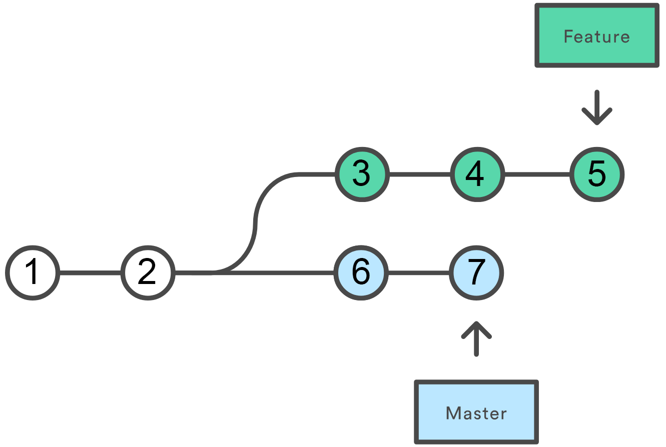
git rebase

Git merge vs git rebase



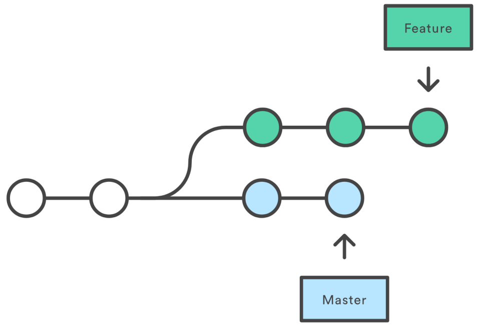
git rebase

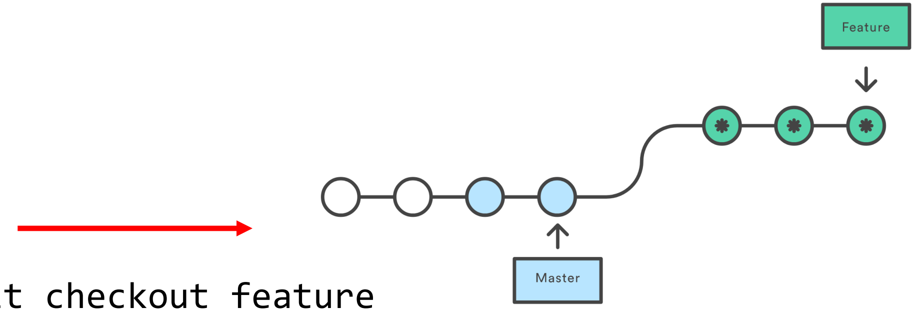
```
[ ~ ]git log --all --graph --decorate --oneline

* 44cd78c (HEAD -> master) adds 7
* d1ad5e9 adds 6
|   * 6cf4d13 (feature) adds 5
|   * b7f520f adds 4
|   * 755620b adds 3
|  /
* adea689 adds 2
* 7b66d7c adds 1
```



git rebase

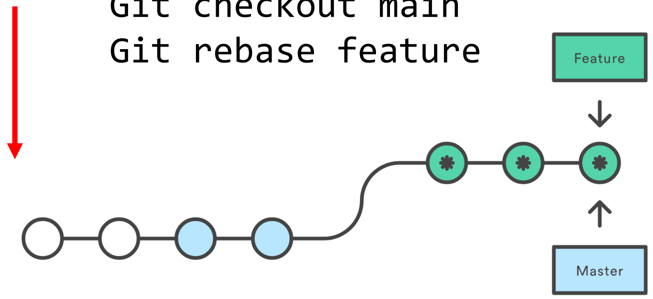
Git merge vs git rebase





```
Git checkout feature
Git rebase main
```

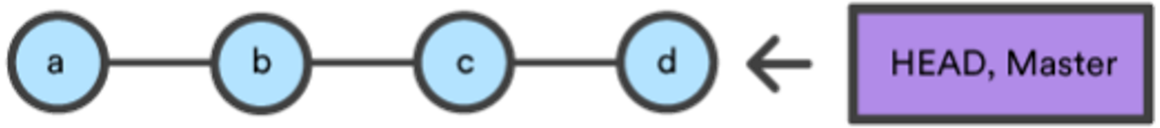
```
Git checkout main
Git rebase feature
```



git reset basics

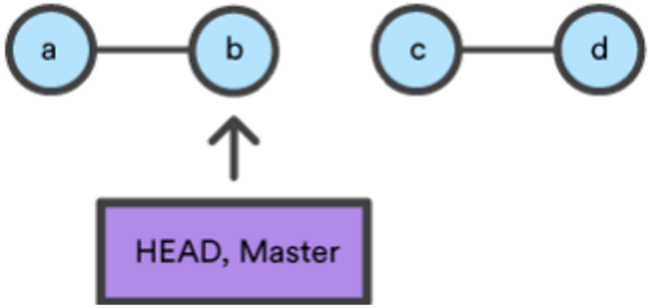
- Allows user to modify their repository history
- Helps rollback to a specific commit
- Changes back to a specific commit in a brute-force kind of way that disrupts the commit history of a repository.
- Used on your local, private repositories, especially if the repository is shared by others

We have the following sequence of commits



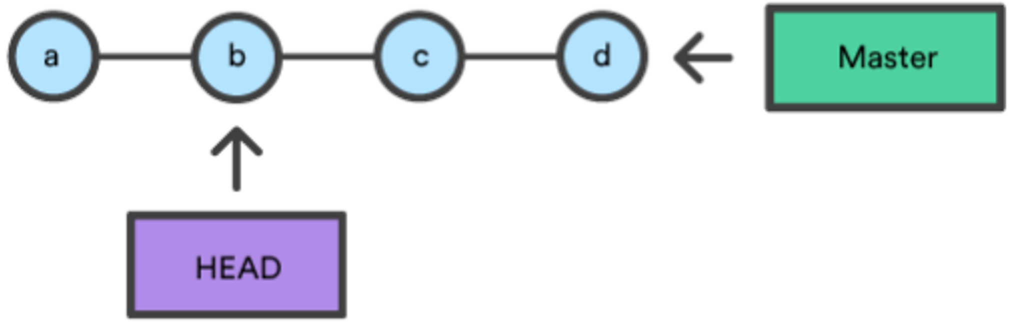
- The above diagram is a linked list of commits
- Let’s say we made 4 commits so far, A,B,C,D
- As we can see our Master and Head pointer points to our latest commit D

git reset basics continued...



Moves both the head AND branch pointers to a specific commit and the commit history is modified.

Recall *git checkout*



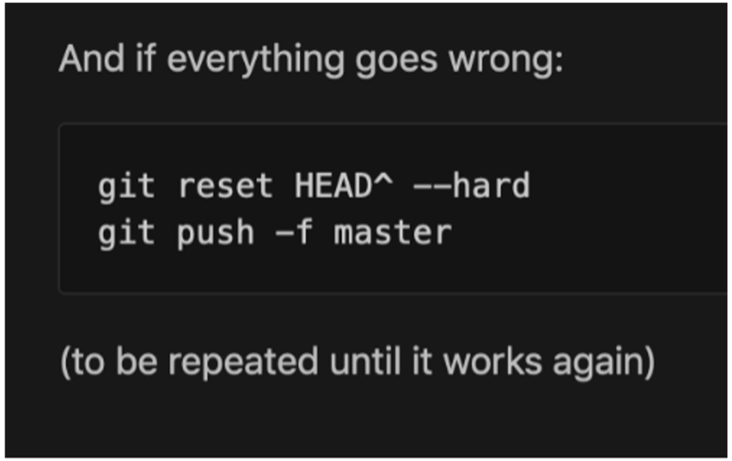
Move ONLY the HEAD pointer to a specific commit and the commit history remains untouched.

git reset --hard <hash>

- Most dangerous type of reset
- Moves the head and master pointer to the target commit
- Staging area and working directory are changed to match the specific commit
- Files in the staging area prior to running this command are discarded
    - Can cause large amounts of data loss if used incorrectly

git reset --soft <hash>

- Moves the head and master pointer to the target commit
- Staging area and working directory are left untouched
  - This is generally the safest option


And if everything goes wrong:

```
git reset HEAD^ --hard
git push -f master
```

(to be repeated until it works again)

I can hear the cries...

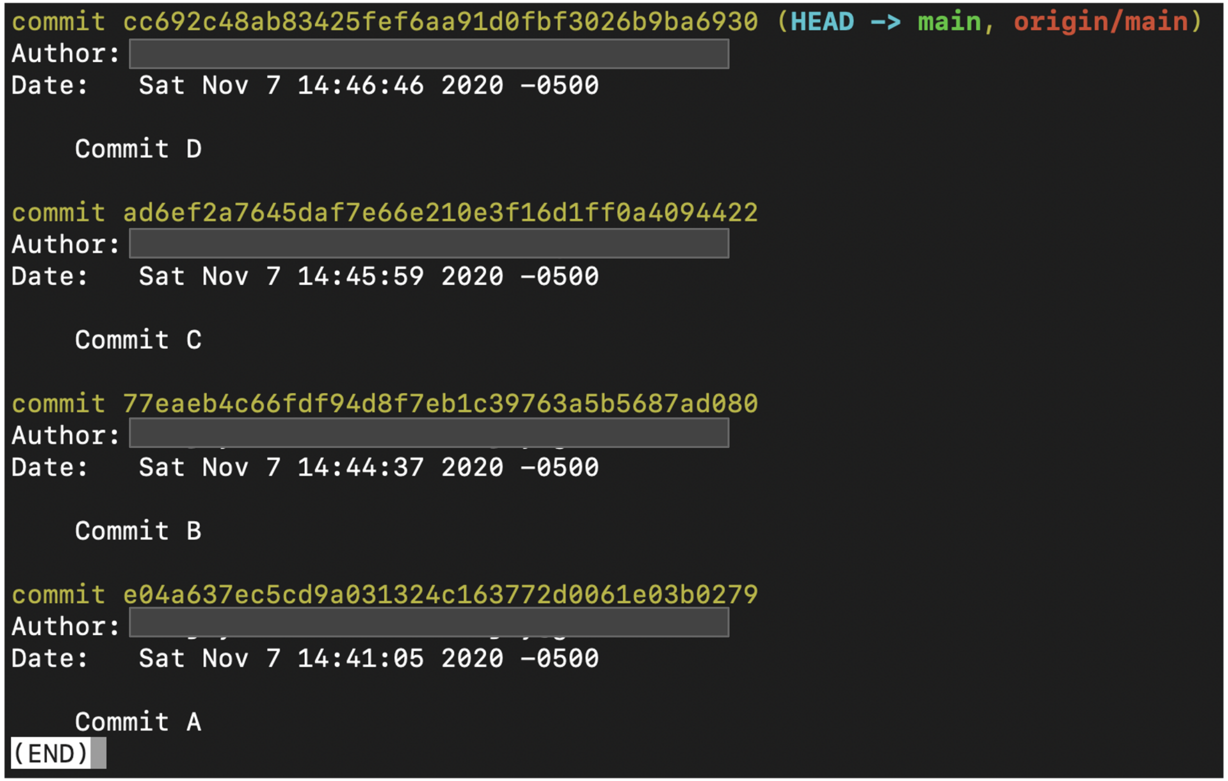
git reset --mixed <hash>

- Meant to be a median between *“--soft”* and *“--hard”,*
- The DEFAULT option if a mode for reset is not specified
- Moves the head and master pointer to the target commit
- Changes the staging area to match the specific commit
- Files in the current staging area moved back to your current working directory

Fixing a team’s mistake

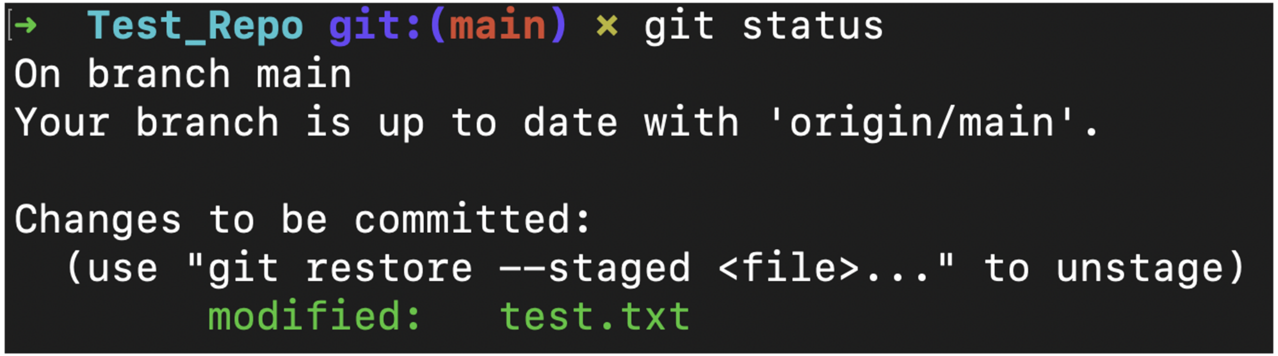
Demo of git reset and an introduction to git revert

Consider the follow Repository’s Commit History

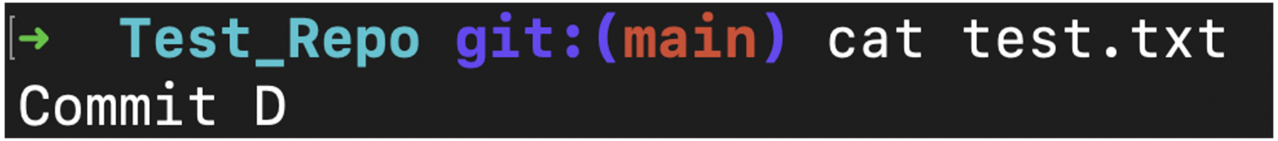


```
commit cc692c48ab83425fef6aa91d0fbf3026b9ba6930 (HEAD -> main, origin/main)
Author: [redacted]
Date:   Sat Nov 7 14:46:46 2020 -0500

    Commit D

commit ad6ef2a7645daf7e66e210e3f16d1ff0a4094422
Author: [redacted]
Date:   Sat Nov 7 14:45:59 2020 -0500

    Commit C

commit 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080
Author: [redacted]
Date:   Sat Nov 7 14:44:37 2020 -0500

    Commit B

commit e04a637ec5cd9a031324c163772d0061e03b0279
Author: [redacted]
Date:   Sat Nov 7 14:41:05 2020 -0500

    Commit A
(END)
```

Consider the same Repository's Staging Area and Working Directory



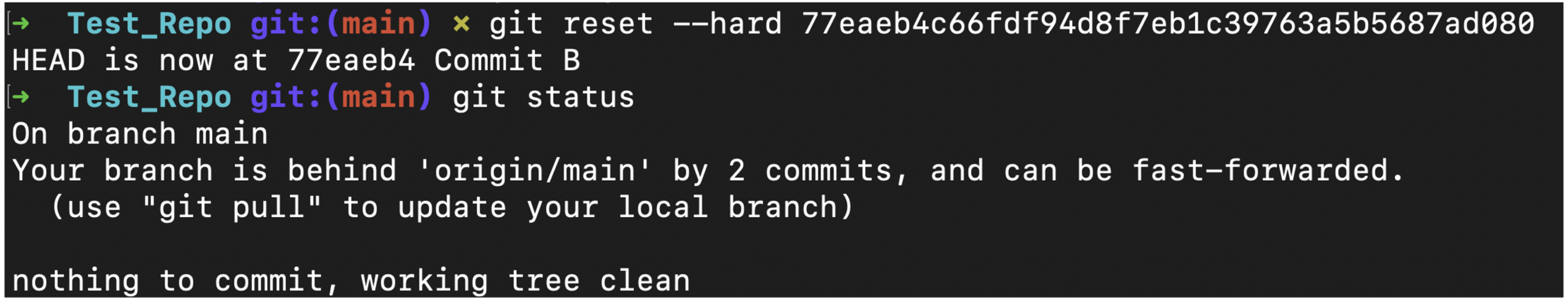
```
➜  Test_Repo git:(main) ✗ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.txt
```



```
➜  Test_Repo git:(main) cat test.txt
Commit D
```

git reset --hard example

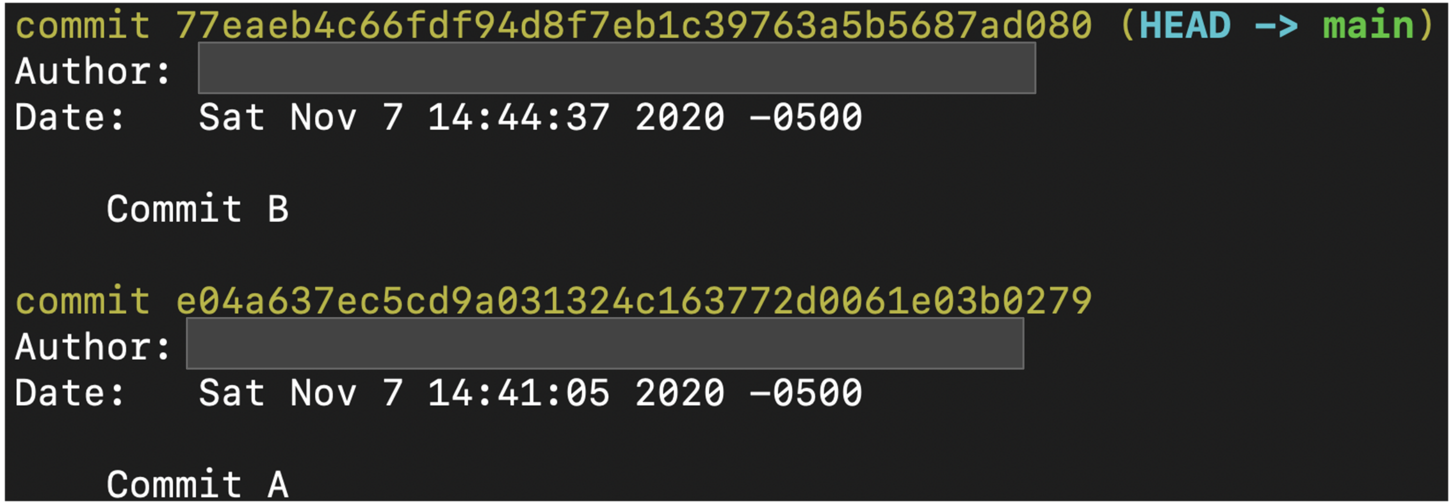
Notice how the staging area is now empty because Commit B's Staging area was empty



```
➜  Test_Repo git:(main) ✗ git reset --hard 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080
HEAD is now at 77eaeb4 Commit B
➜  Test_Repo git:(main) git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

git reset --hard example continued...

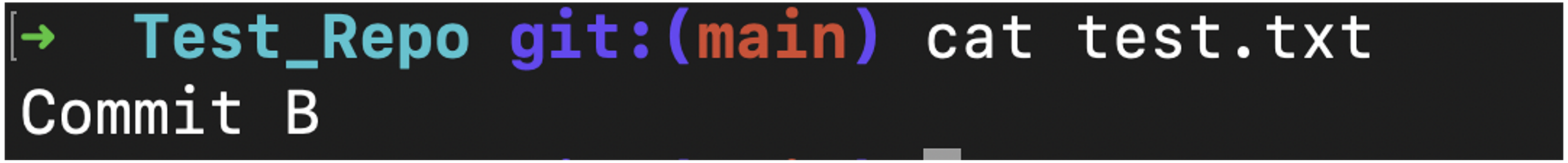
Run git log to see how the list of commits has been modified.



```
commit 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080 (HEAD -> main)
Author: [redacted]
Date:   Sat Nov 7 14:44:37 2020 -0500

    Commit B

commit e04a637ec5cd9a031324c163772d0061e03b0279
Author: [redacted]
Date:   Sat Nov 7 14:41:05 2020 -0500

    Commit A
```

git reset --hard example continued...

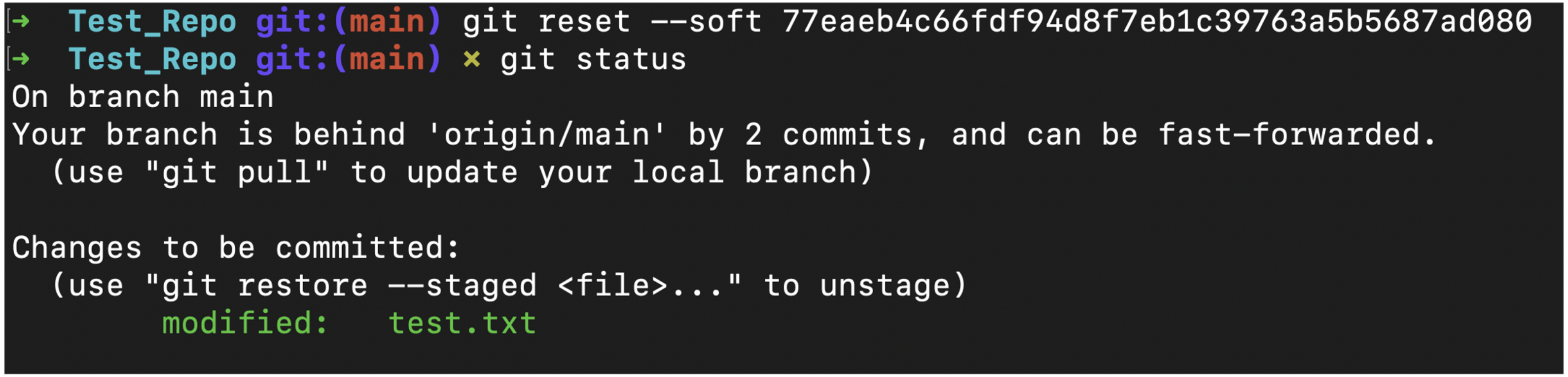
Notice how the working directory files have been ‘reverted’ and now contain a different test.txt



```
→  Test_Repo git:(main) cat test.txt
Commit B
```

git reset --soft example

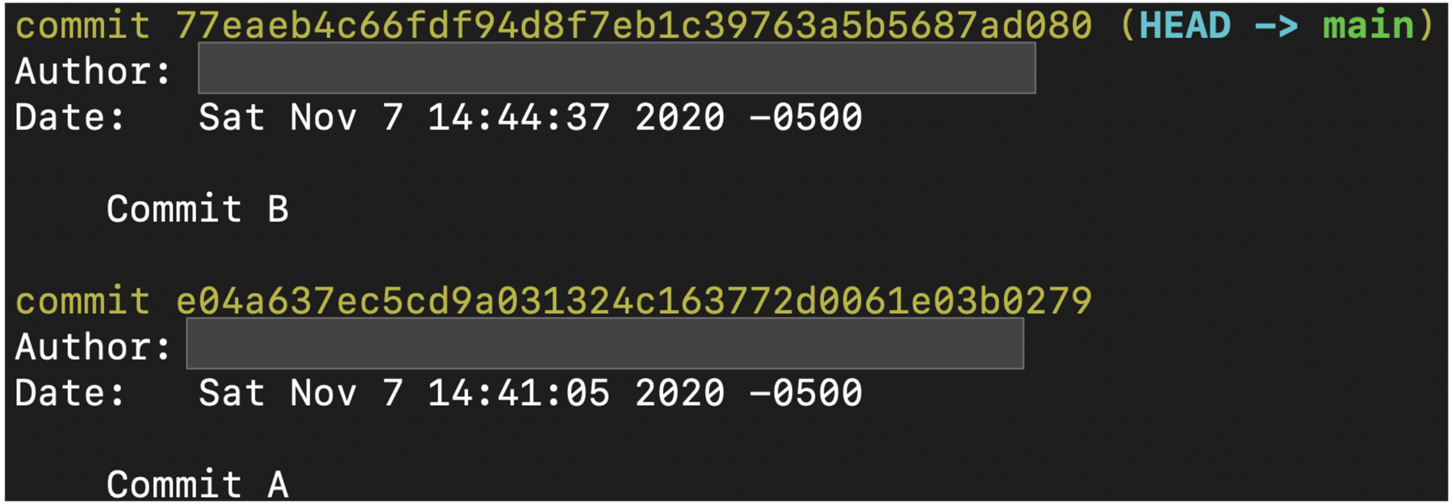
Notice how the staging area remains untouched



```
→  Test_Repo git:(main) git reset --soft 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080
→  Test_Repo git:(main) ✗ git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.txt
```

git reset --soft example continued...

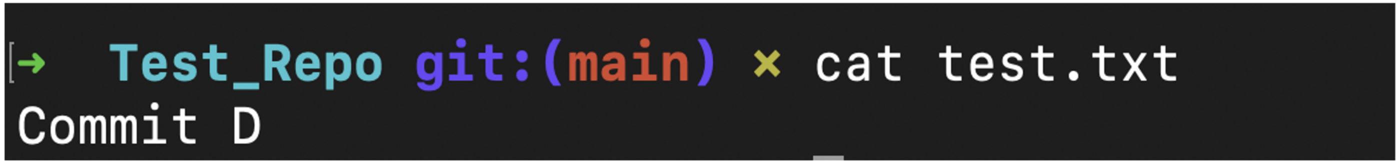
Notice how the log is the exact same as the log after we ran git reset --hard



```
commit 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080 (HEAD -> main)
Author: [redacted]
Date:   Sat Nov 7 14:44:37 2020 -0500

    Commit B

commit e04a637ec5cd9a031324c163772d0061e03b0279
Author: [redacted]
Date:   Sat Nov 7 14:41:05 2020 -0500

    Commit A
```

git reset --soft example continued...

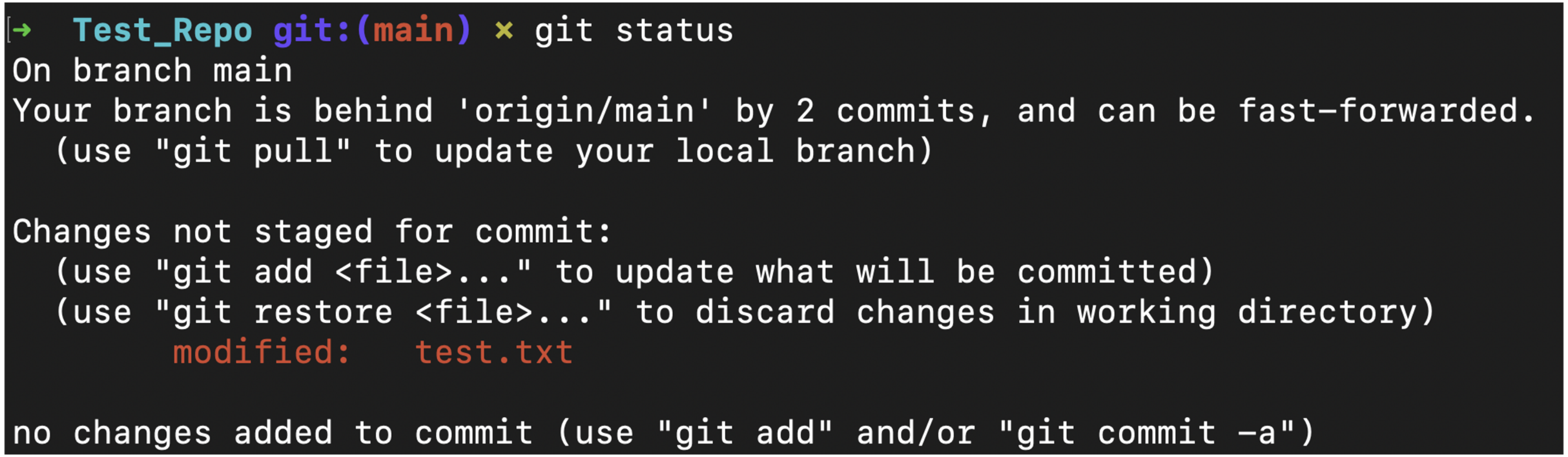
Notice how the working directory file has been left untouched



```
[➜  Test_Repo git:(main) ✗ cat test.txt
Commit D
```

git reset --mixed example

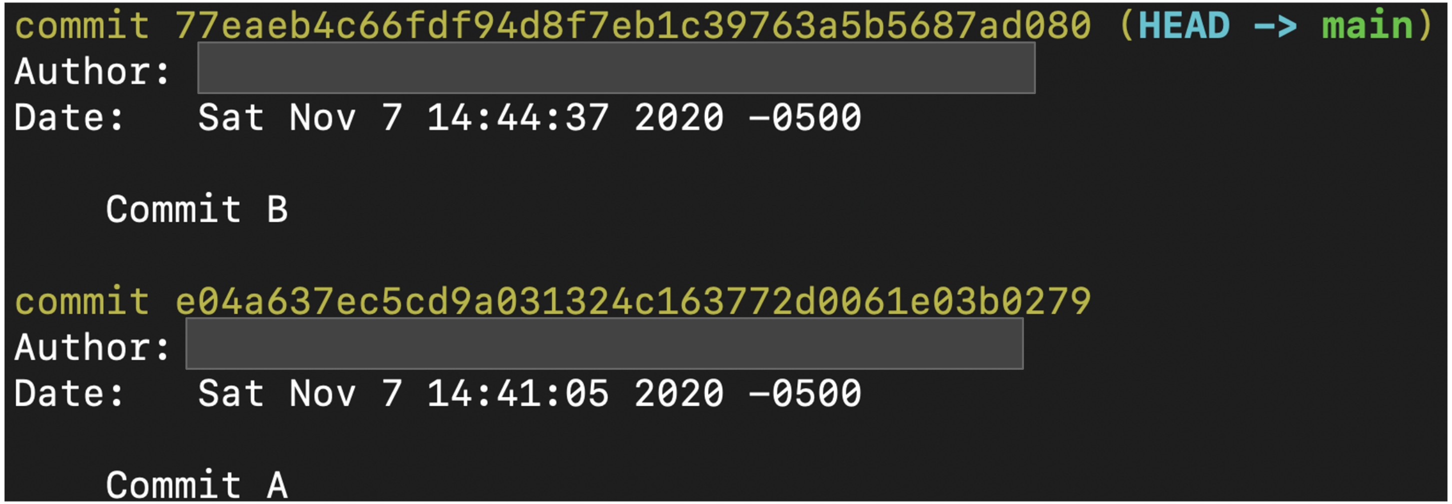
Notice how the staging area is now empty because Commit B’s Staging area was empty



```
→  Test_Repo git:(main) ✗ git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git reset --mixed example continued...

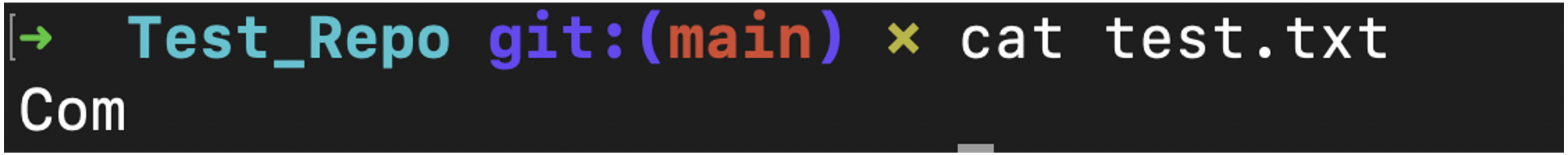
Notice how the log is the exact same as before



```
commit 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080 (HEAD -> main)
Author: [redacted]
Date:   Sat Nov 7 14:44:37 2020 -0500

    Commit B

commit e04a637ec5cd9a031324c163772d0061e03b0279
Author: [redacted]
Date:   Sat Nov 7 14:41:05 2020 -0500

    Commit A
```

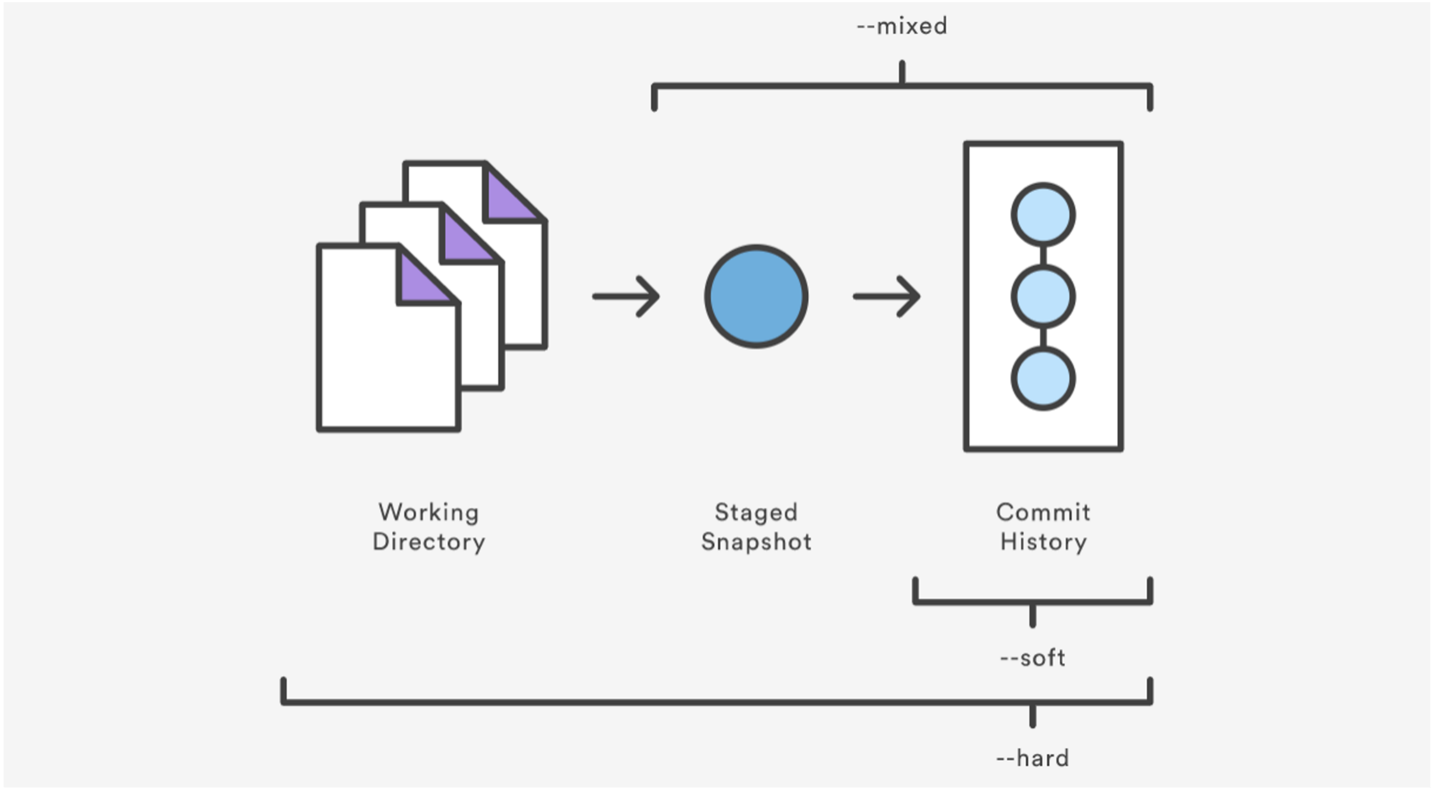
git reset --mixed example continued...

Notice how the modified file we added to the staging directory is now in our working directory.



```
➜  Test_Repo git:(main) ✗ cat test.txt
Com
```

Summarized Diagram



Popular Usage of git reset:

- If ever, you add a file to the staging area but want to remove the file from staging, we run the following command: `git reset HEAD TARGET-FILE`

- If you ever want to abandon all local changes and start fresh with a copy of your remote repository, run `git reset --hard` and then `git pull`

Clicker Quiz

Which of the following commands only modify the commit history

a) git reset --hard
b) git reset --soft
c) git reset --mixed
d) git reset

Clicker Quiz

Which of the following commands only modify the commit history

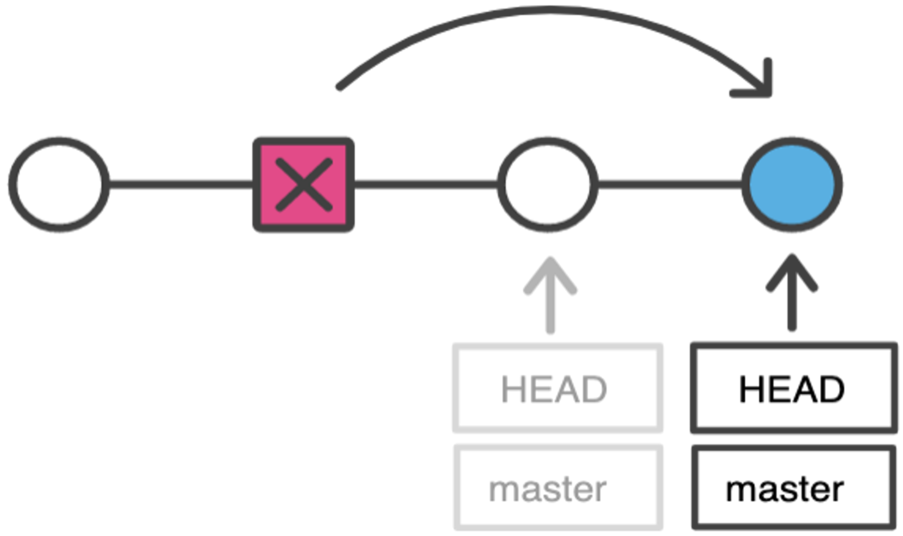
a) git reset --hard
b) git reset --soft
c) git reset --mixed
d) git reset

git revert

- Used for undoing changes to a repository.
- Revert does not modify the repository history
- Makes a new commit that that reverses any changes to achieve the state of the specified commit
- Use this kind of version control on public branches instead

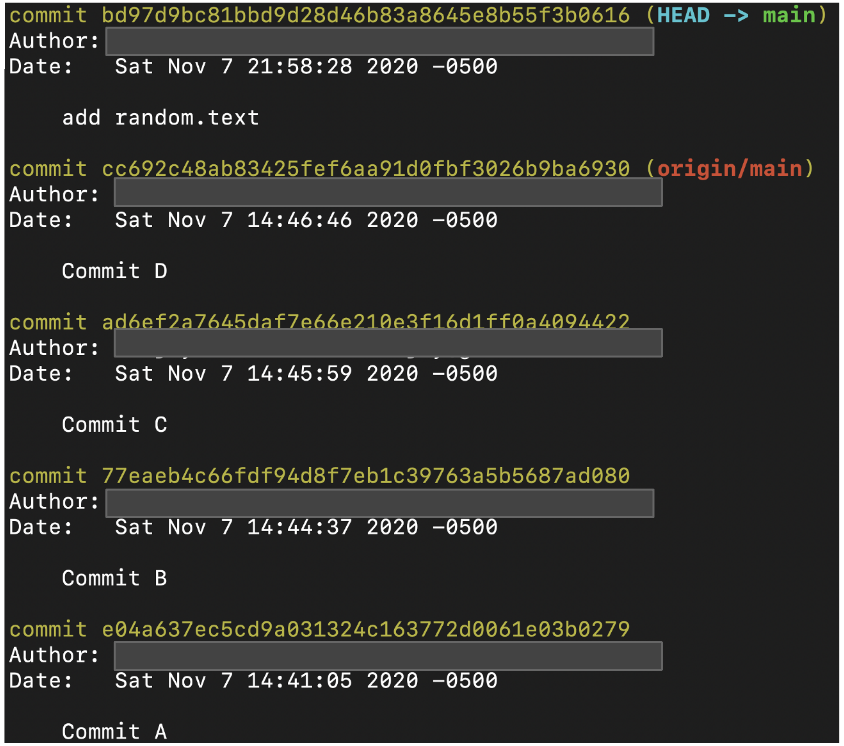
git revert continued

Notice how the new head and master are essentially just a copy of the second commit



git revert example

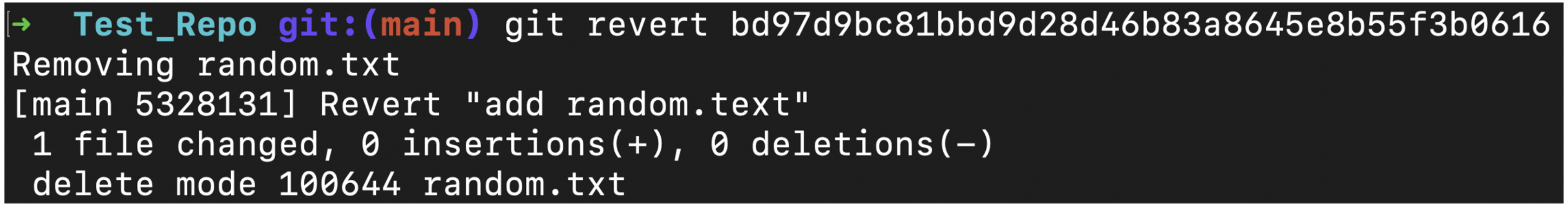
1. Consider the following situation on Test_Repo.
2. A team accidentally adds a file called random.txt
3. We want to revert the other team's change in a safe manner



```
commit bd97d9bc81bbd9d28d46b83a8645e8b55f3b0616 (HEAD -> main)
Author: [redacted]
Date:   Sat Nov 7 21:58:28 2020 -0500

    add random.text

commit cc692c48ab83425fef6aa91d0fbf3026b9ba6930 (origin/main)
Author: [redacted]
Date:   Sat Nov 7 14:46:46 2020 -0500

    Commit D

commit ad6ef2a7645daf7e66e210e3f16d1ff0a4094422
Author: [redacted]
Date:   Sat Nov 7 14:45:59 2020 -0500

    Commit C

commit 77eaeb4c66fdf94d8f7eb1c39763a5b5687ad080
Author: [redacted]
Date:   Sat Nov 7 14:44:37 2020 -0500

    Commit B

commit e04a637ec5cd9a031324c163772d0061e03b0279
Author: [redacted]
Date:   Sat Nov 7 14:41:05 2020 -0500

    Commit A
```

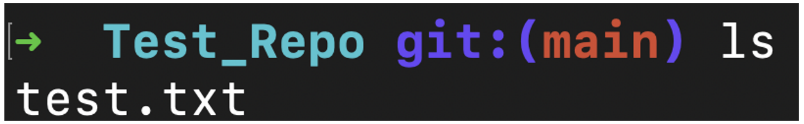
git revert example

To revert the last commit, we copy the hash and use git revert <hash>



```
➜  Test_Repo git:(main) git revert bd97d9bc81bbd9d28d46b83a8645e8b55f3b0616
Removing random.txt
[main 5328131] Revert "add random.text"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 random.txt
```

As we see below, we have reverted their addition of the file and can safely push these changes to the remote repository



```
➜  Test_Repo git:(main) ls
test.txt
```

When to use what?

Local	Remote
● git revert ● git reset ● git cherry-pick ● git checkout	● git revert ● git cherry-pick ● git checkout

Clicker Quiz

Fill in the blank:
“git revert is __________ , compared to git reset”

a) safer to use locally
b) brute force
c) safer to use remotely
d) more dangerous to use remotely

Clicker Quiz

Fill in the blank:
“git revert is __________ , compared to git reset”

a) safer to use locally
b) brute force
c) safer to use remotely
d) more dangerous to use remotely

Advanced Git

Advanced Git commands

More Git Commands

git commit --amend

Modifies your most recent commit by combining changes in your staging area with your previous commit

git reflog

Lists the history of updates to ref pointers in your local repository

git clean

Removes up untracked changes files in your repository. Keep in mind that the -n or -f flag is require

More Git Commands

git ls-files -s

Can be used with the "*--deleted*", "*--modified*", or "*--others* AND *--exclude-standard*" flag to list the files of each type

git reset --soft HEAD~N

Removes last N by moving the current HEAD to the specified commit

git diff --cached

Shows specific changes in files that are currently in the staging area

Clicker Quiz

Which of the following flags combine changes in your staging area with your previous commit?

a) --add
b) --readd
c) --revert
d) --prevamend
e) --amend

Clicker Quiz

Which of the following flags combine changes in your staging area with your previous commit?

a) --add
b) --readd
c) --revert
d) --prevamend
e) --amend

Git Hooks

Git Hooks

Git can trigger custom scripts that perform certain operations. These scripts are referred to as **hooks**.

```
[ ~ ]ls .git/hooks

pre-commit.sample
applypatch-msg.sample                pre-merge-commit.sample
commit-msg pre-push.sample           commit-msg.sample pre-rebase.sample
fsmonitor-watchman.sample            pre-receive.sample
post-update.sample                   prepare-commit-msg.sample
pre-applypatch.sample                update.sample
```

Creating a commit-msg Hook

```
[ ~ ] cd .git/hooks
[ ~ ] cp commit-msg.sample commit-msg
[ ~ ] chmod +x  commit-msg
```

commit-msg Hook

```
#!/usr/bin/env ruby
message_file = ARGV[0]
message = File.read(message_file)
$format  = /\[(\w+)\]:/
if !$format.match(message)
  puts "[POLICY] Your message is not formatted correctly"
  puts "[STANDARD] Your message should be in the format: ‘[module]:
commit message’ "
  exit 1
end
```

Test commit-msg Hook

```
[ ~ ]git commit -m 'test'
[POLICY] Your message is not formatted correctly
[STANDARD] Your message should be in the format: ‘[module]:
commit message’

[ ~ ]git commit -m ”[test]: testing tests”

[main 3457535] [test]: testing tests
 1 file changed, 1 insertion(+)
```