Table of Contents

Cookbook

Presentations

Git Cookbook

My Git Cookbook with notes during learning.

In this chapter I save essential commands

At the beginning of the days...

Create repo

```
$ git init
```

Details

```
$ mkdir my-repo
```

```
$ git init
Initialized empty Git repository in /my-repo/.git/
```

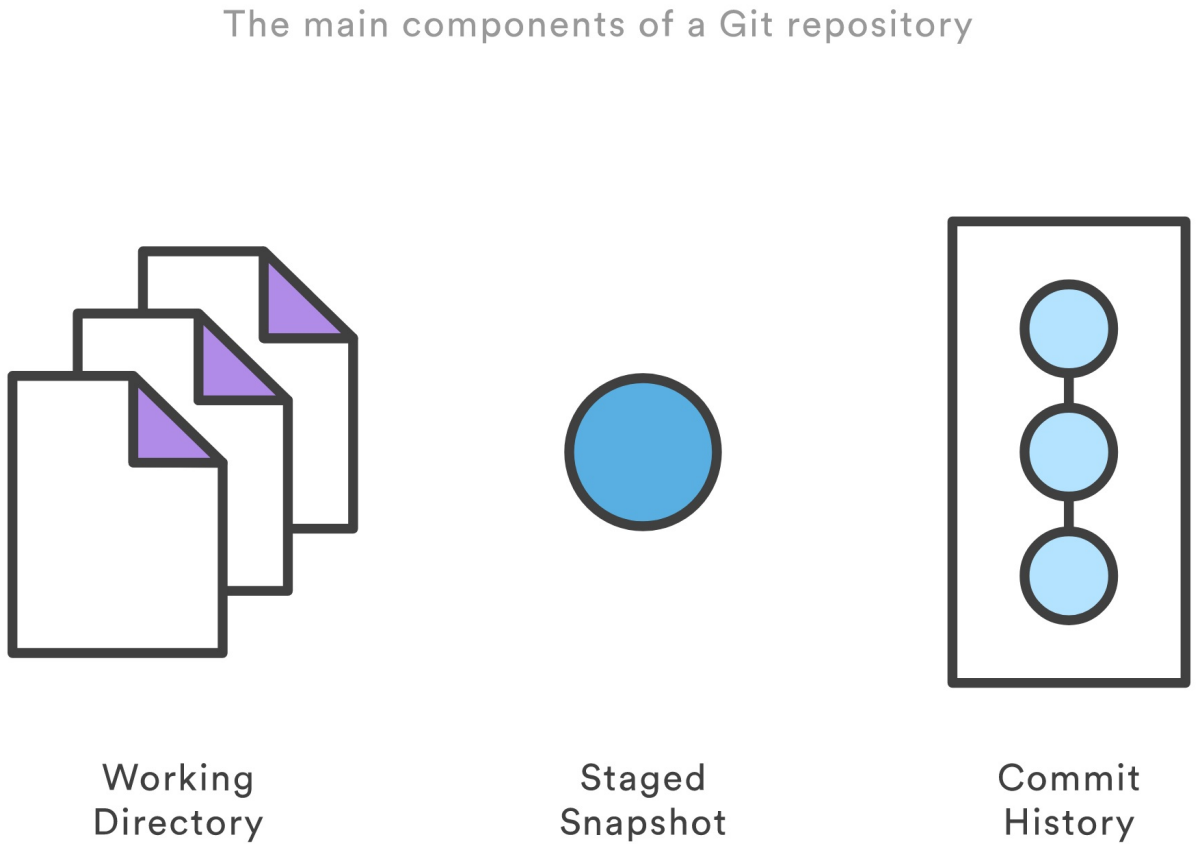
```
$ tree .git
.git
├── HEAD
├── branches
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

9 directories, 13 files
```

Further reading

- `$ git help init`

There 3 main components of a Git repository



> https://wac-cdn.atlassian.com/dam/jcr:0c5257d5-ff01-4014-af12-faf2aec53cc3/01.svg?cdnVersion=fk

Add files to staging

Let's create a file

```
$ echo "111" > 1.txt
$ ls .
1.txt
```

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    1.txt

nothing added to commit but untracked files present (use "git ad
d" to track)
```

```
$ tree .git
.git
├── HEAD
├── branches
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

9 directories, 13 files
```

As you can see nothing changed in the repo's state

And now we add the file to the staging area

```
 $ git add 1.txt
```

```
 $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   1.txt
```

```
 $ tree .git
.git
├── HEAD
├── branches
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── objects
│   ├── 58
│   │   └── c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

10 directories, 15 files
```

As you can see appeared new object `58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c`

- What is this?

  It is a blob with contents of `1.txt`

- What is `58c9bd...` ?

  It's a **sha1** hash of `blob␣<blob size>\0<content>`

```
$ git hash-object 1.txt
58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
```

> **Important** As you can see hash doesn't depend of time, OS, localization and so on. It means that if you create file with the same content on other machine you will have the same hash.

But if we add another file with the same content?

```
$ mkdir files
$ echo "111" > files/2.txt
$ tree .
.
├── 1.txt
└── files
    └── 2.txt

1 directory, 2 files
```

```
$ git add .
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   1.txt
    new file:   files/2.txt
```

So, what do we have in the **objects**?

```
$ tree .git/objects
.git/objects
├── 58
│   └── c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
├── info
└── pack

3 directories, 1 file
```

Nothing changed!

Git just **can't create** any **object** because new object will have **the same hash and content**.

Further reading

- `$ git help add`
- Saving changes

Save it!

```
$ git init
Initialized empty Git repository in /my-repo/.git/
$ echo "111" > 1.txt
$ mkdir files
$ echo "111" > files/2.txt
$ git add .
$ git commit -m "Initial commit"
[master (root-commit) ce6496b] Initial commit
 2 files changed, 2 insertions(+)
 create mode 100644 1.txt
 create mode 100644 files/2.txt
```

And what do we have in **objects**?

```
$ tree .git
.git
├── COMMIT_EDITMSG
├── HEAD
├── branches
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
```

```
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 3c
│   │   └── 1cb3829a09a57df9ea94f7bfdf76ed123161c2
│   ├── 58
│   │   └── c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
│   ├── 61
│   │   └── caec3709a1e6473b2f33bfc92bd9d138071e88
│   ├── ce
│   │   └── 6496b7a3dd69b1ee8e403c22b77a148bd38ec4
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

What creates `commit`

Let's checkout what was created...

We have these objects:

- `3c1cb3829a09a57df9ea94f7bfdf76ed123161c2`
- `58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c`
- `61caec3709a1e6473b2f33bfc92bd9d138071e88`
- `ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4`

Before start lets read about `cat-file`

`ce6496` - commit

```
$ git cat-file -t ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4
commit
```

```
$ git cat-file -p ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4
tree 3c1cb3829a09a57df9ea94f7bfdf76ed123161c2
author Konstantin Portnov <konstantin.portnov@mercadolibre.cl> 1
500482225 -0400
committer Konstantin Portnov <konstantin.portnov@mercadolibre.cl
> 1500482225 -0400

Initial commit
```

```
$ cat ./git/
```

`3c1cb3` - tree `.`

```
$ git cat-file -t ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4
tree
```

```
$ git cat-file -p 3c1cb3829a09a57df9ea94f7bfdf76ed123161c2
100644 blob 58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c    1.txt
040000 tree 61caec3709a1e6473b2f33bfc92bd9d138071e88    files
```

`61caec` - tree `./files`

```
$ git cat-file -t 61caec3709a1e6473b2f33bfc92bd9d138071e88
tree
```

```
$ git cat-file -p 61caec3709a1e6473b2f33bfc92bd9d138071e88
git cat-file -p 61caec3709a1e6473b2f33bfc92bd9d138071e88
100644 blob 58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c    2.txt
```

`58c9bd` - blob "111"

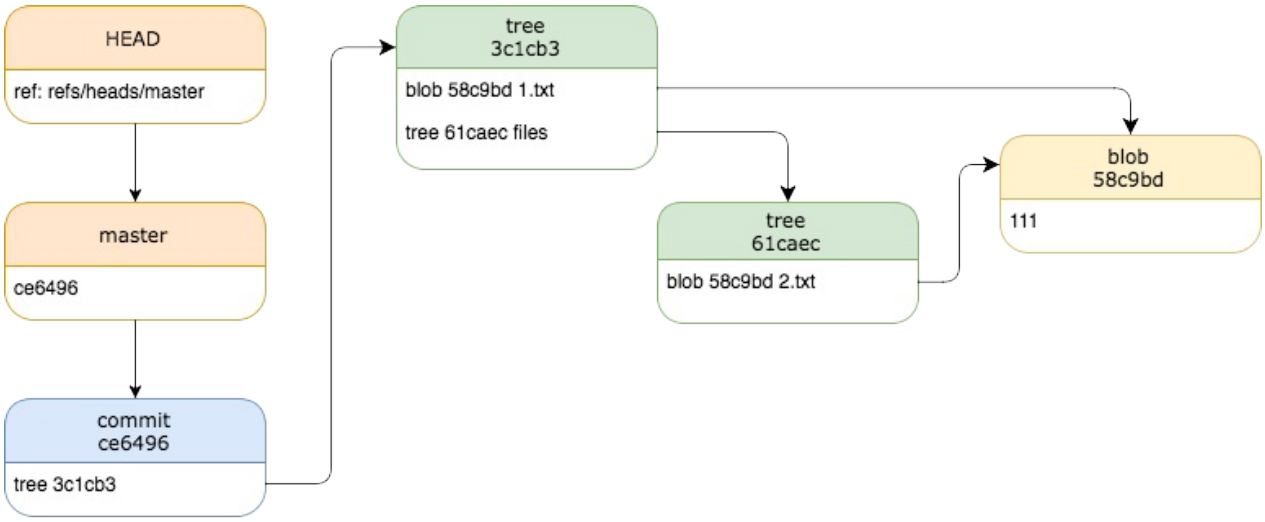
```
$ git cat-file -t 58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
blob
```

```
$ git cat-file -p 58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
111
```

Result

So, we have this situation:

Type	Hash
commit	`ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4`
tree	`3c1cb3829a09a57df9ea94f7bfdf76ed123161c2`
tree	`61caec3709a1e6473b2f33bfc92bd9d138071e88`
blob	`58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c`



Each commit contains at least 3 parts:

- Commit
- Tree
- Blob

Let's add another commit

```
$ echo "222" > 2.txt
$ git add .
$ git commit -m "Second commit"
[master e1ed469] Second commit
 1 file changed, 1 insertion(+)
 create mode 100644 2.txt
```

```
$ tree .git/objects
.git/objects
├── 3c
│   └── 1cb3829a09a57df9ea94f7bfdf76ed123161c2
├── 58
│   └── c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c
├── 5a
│   └── 2554e0627a4bd4da9ea522975b9b97f5278b46
├── 61
│   └── caec3709a1e6473b2f33bfc92bd9d138071e88
├── c2
│   └── 00906efd24ec5e783bee7f23b5d7c941b0c12c
├── ce
│   └── 6496b7a3dd69b1ee8e403c22b77a148bd38ec4
├── e1
│   └── ed469b8595e129f85af5b6d1fd70957fa5a95a
├── info
└── pack
```

And we have **3 new** files:

Type	Hash
commit	`e1ed469b8595e129f85af5b6d1fd70957fa5a95a`
tree	`5a2554e0627a4bd4da9ea522975b9b97f5278b46`
blob	`c200906efd24ec5e783bee7f23b5d7c941b0c12c`

`e1ed46` - commit

```
$ git cat-file -t e1ed469b8595e129f85af5b6d1fd70957fa5a95a
commit
```

```
$ git cat-file -p e1ed469b8595e129f85af5b6d1fd70957fa5a95a
tree 5a2554e0627a4bd4da9ea522975b9b97f5278b46
parent ce6496b7a3dd69b1ee8e403c22b77a148bd38ec4
author Konstantin Portnov <konstantin.portnov@mercadolibre.cl> 1
500488288 -0400
committer Konstantin Portnov <konstantin.portnov@mercadolibre.cl
> 1500488288 -0400

Second commit
```

`5a2554` - tree `.`

```
$ git cat-file -t 5a2554e0627a4bd4da9ea522975b9b97f5278b46
tree
```

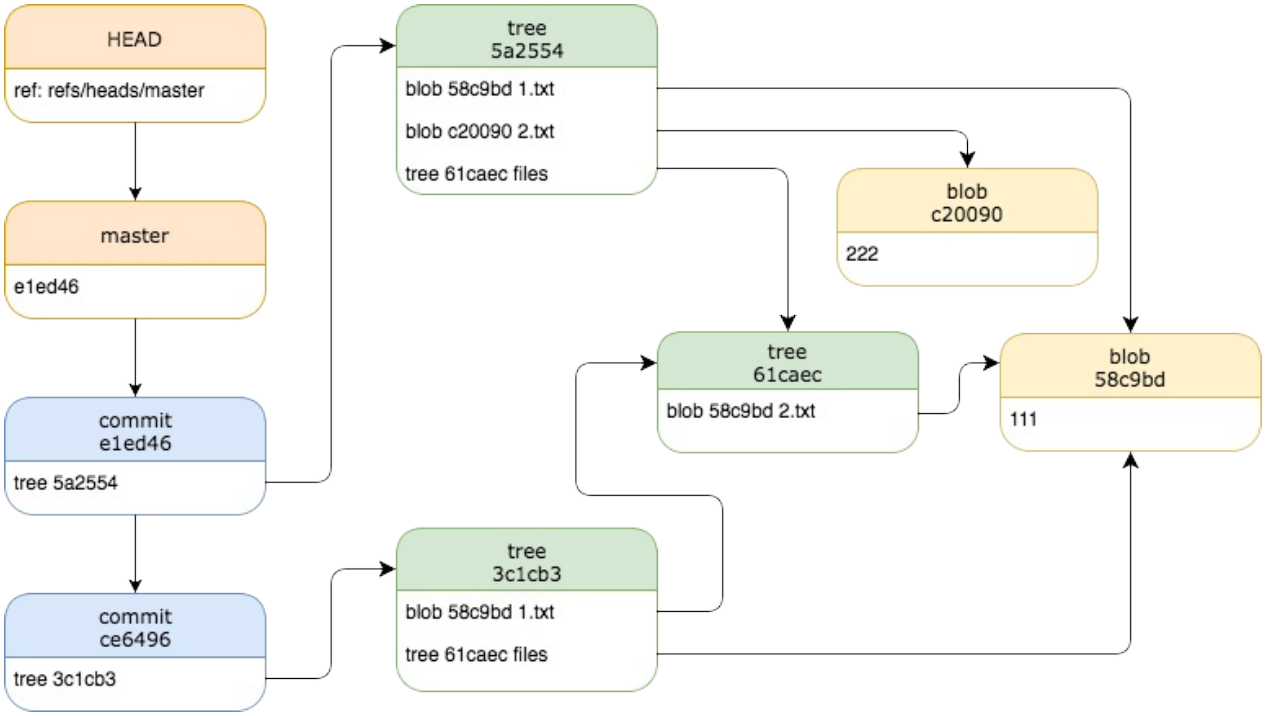
```
$ git cat-file -p 5a2554e0627a4bd4da9ea522975b9b97f5278b46
100644 blob 58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c    1.txt
100644 blob c200906efd24ec5e783bee7f23b5d7c941b0c12c    2.txt
040000 tree 61caec3709a1e6473b2f33bfc92bd9d138071e88    files
```

`c20090` - blob "222"

```
$ git cat-file -t c200906efd24ec5e783bee7f23b5d7c941b0c12c
blob
```

```
$ git cat-file -p c200906efd24ec5e783bee7f23b5d7c941b0c12c
222
```

And now we have graph as at this picture



Further reading

- `$ git help commit`
- Saving changes

Tag them all!

Tag is an object

My first tag

Add - Light first

```
$ git tag the-beginning
```

```
$ git cat-file -p the-beginning
tree 99339a532fb221a08fb0faf07175e5380bc0999d
parent 6347ba4da019ebb50a9bcdb6ced527adb83c78ee
author Konstantin Portnov <konstantin.portnov@mercadolibre.cl> 1
500360967 -0400
committer Konstantin Portnov <konstantin.portnov@mercadolibre.cl
> 1500360967 -0400

Second
```

```
$ tree .git/refs
.git/refs
├── heads
│   ├── feature
│   └── master
└── tags
    └── the-beginning

2 directories, 3 files
```

```
$ cat .git/refs/tags/the-beginning
1dc39d54f58d34ce093e3894d26ff736ab65fd25
```

```
$ git cat-file -t the-beginning
commit
```

```
$ git cat-file -p the-beginning
tree 99339a532fb221a08fb0faf07175e5380bc0999d
parent 6347ba4da019ebb50a9bcdb6ced527adb83c78ee
author Konstantin Portnov <konstantin.portnov@mercadolibre.cl> 1
500360967 -0400
committer Konstantin Portnov <konstantin.portnov@mercadolibre.cl
> 1500360967 -0400

Second
```

Adding tag you can pass `<commit hash>` or `<object>`

```
$ git tag first HEAD~2
```

> `HEAD~2` second son of the HEAD of the current branch

```
$ git tag seconds develop
```

> Here `develop` is name of branch

List

```
$ git tag
the-beginning
```

Show

Will show the commit at which points tag

```
$ git tag show the-beginning
```

Delete

```
$ git tag -d the-beginning
Deleted tag 'the-beginning' (was 885bb6c)
```

Add - Annotated

```
$ git tag -a -m "My annotation" another-feature
```

What's the difference?

```
$ git cat-file -p another-feature
object 242a260315700452bd8a28b6d2eb8c7f086a8830
type commit
tag another-feature
tagger Konstantin Portnov <konstantin.portnov@mercadolibre.cl> 1
500493476 -0400

My annotation
```

`242a26...` it's a commit tag points to.

```
$ tree .git/refs
.git/refs
├── heads
│   ├── feature
│   └── master
└── tags
    ├── another-feature
    └── the-beginning

2 directories, 4 files
```

```
$ cat .git/refs/tags/another-feature
ecde83ebacd53baf96c26b421a6888c55bae6587
```

Further reading

- `$ git help tag`
- 2.6 Git Basics - Tagging

Next step...

Opps... I mistaked with the last commit

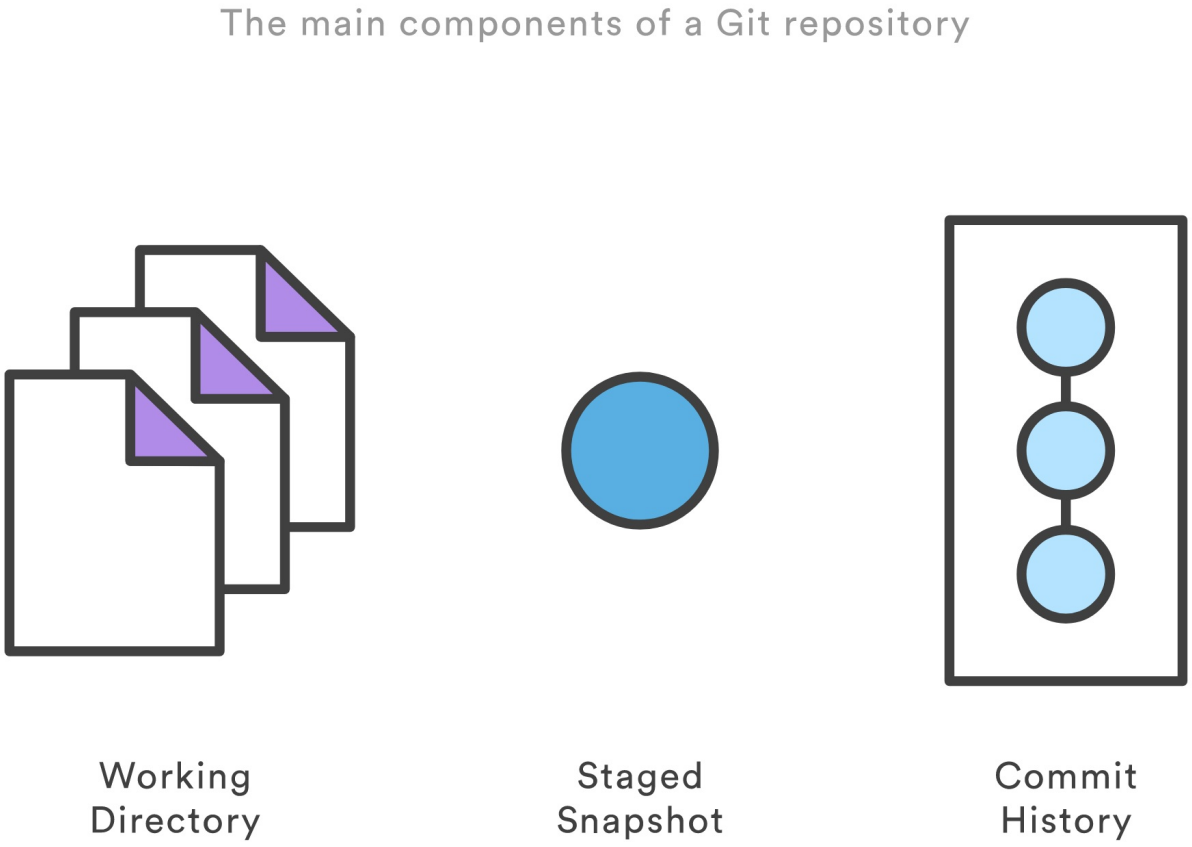
> Alternatively, you can edit the working directory and update the index to fix your mistake, just **as if you were going to create a new commit**, then run

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

Fixing a mistake by rewriting history

Make ~~America~~ Our Project Great Again!

There 3 main components of a Git repository

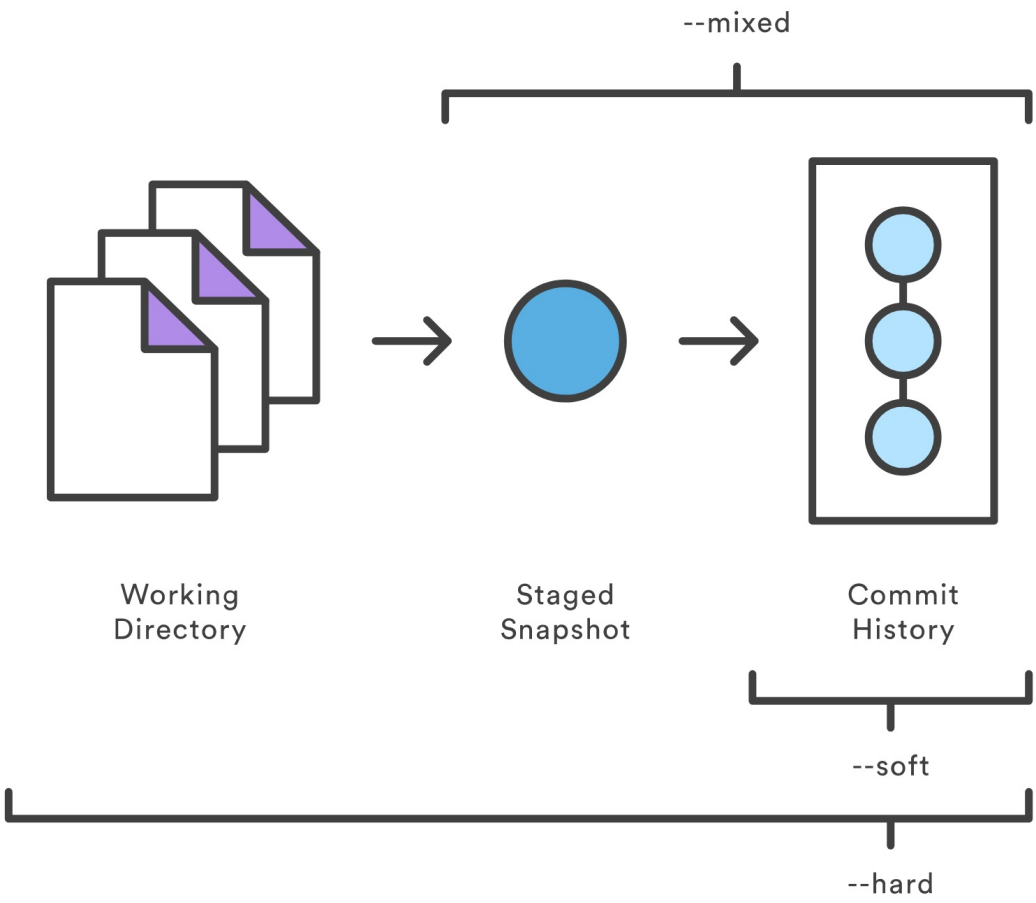


> https://wac-cdn.atlassian.com/dam/jcr:0c5257d5-ff01-4014-af12-faf2aec53cc3/01.svg?cdnVersion=fk

- `--soft` The staged snapshot and working directory are not altered in any way.
- `--mixed` The staged snapshot is updated to match the specified commit, but the working directory is not affected. This is the default option.

  After `mixed` **staging is empty always**. It drops all to the working directory.
- `--hard` The staged snapshot and the working directory are both updated to match the specified commit.

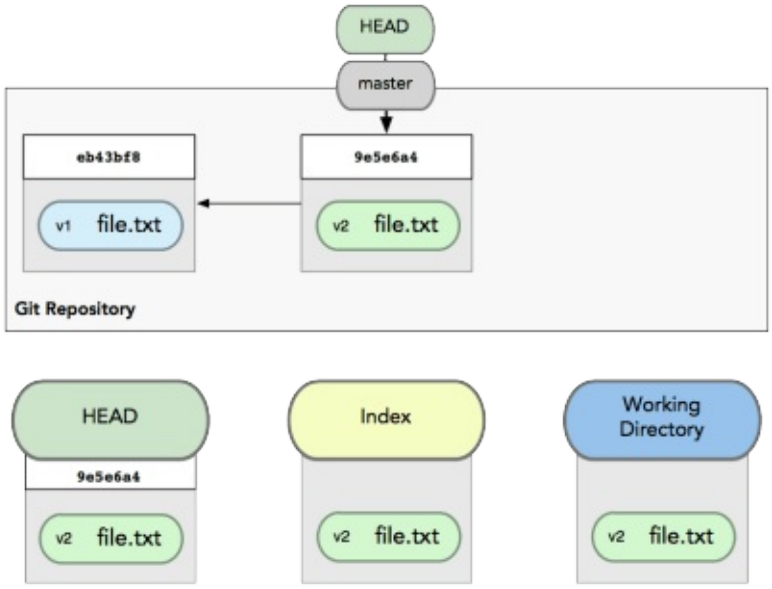
The scope of git reset's modes



> https://wac-cdn.atlassian.com/dam/jcr:2528918b-5c1a-4ab5-8454-88c3a66b14d1/03.svg?cdnVersion=fk

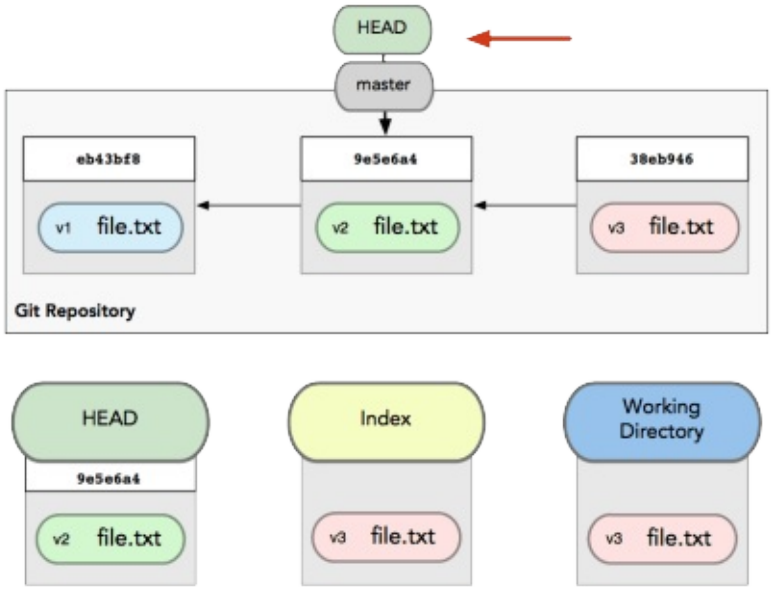
---

Before any `reset`



git commit

`--soft`



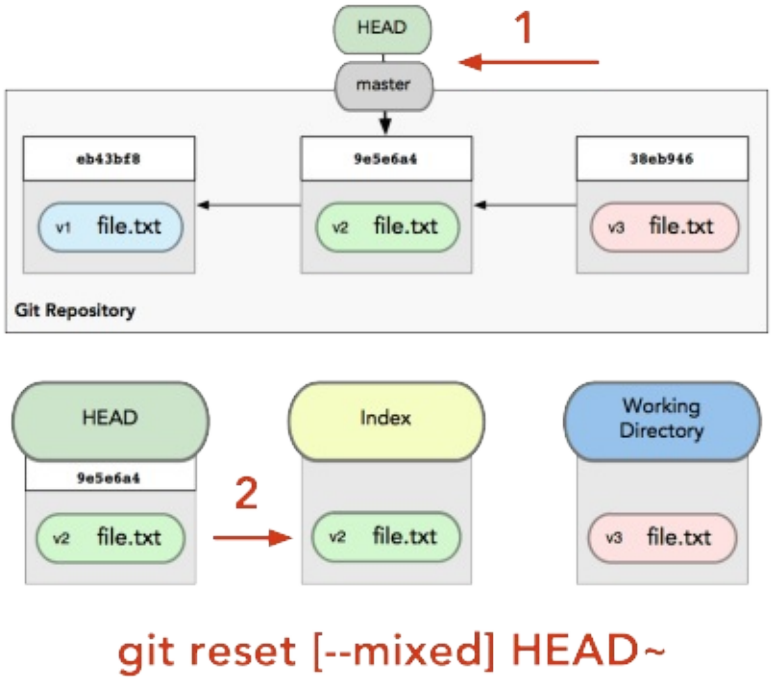
git reset --soft HEAD~

```
$ git reset --soft HEAD~

$ git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   test
```

`--mixed`



git reset [--mixed] HEAD~

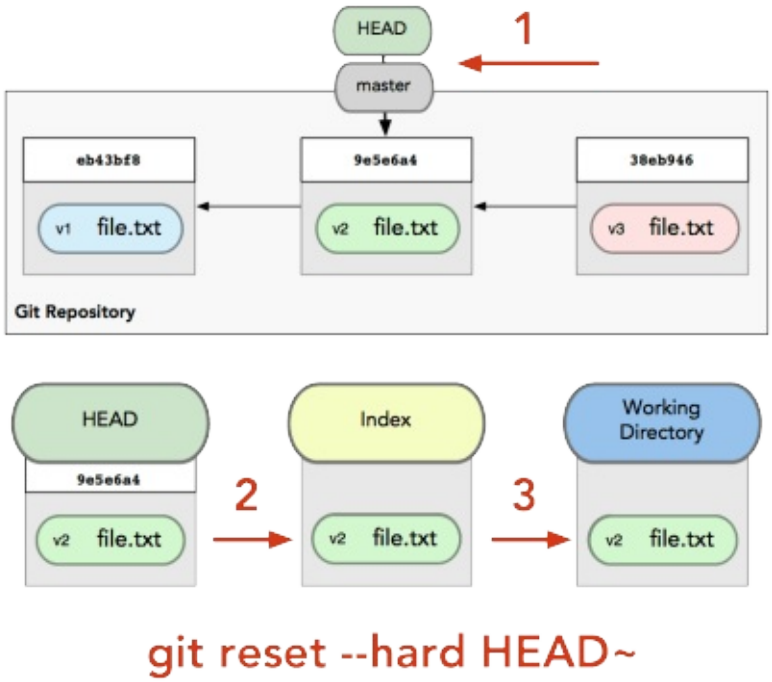
```
$ git reset --mixed HEAD~
Unstaged changes after reset:
M    test

$ git status
On branch test
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:   test

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

`--hard`



git reset --hard HEAD~

```
$ git reset --hard HEAD~
HEAD is now at aaa3981 Add test

$ git status
On branch test
nothing to commit, working tree clean
```

What about untracked files?

They won't be affected!

```
$ touch a.file

$ git status
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    a.file

nothing added to commit but untracked files present (use "git ad
d" to track)

$ git reset --hard HEAD~
HEAD is now at aaa3981 Add test

$ git status
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    a.file

nothing added to commit but untracked files present (use "git ad
d" to track)
```

But files that were added to the staging will be changed to the version that has new tree or **deleted** if they don't exist in the tree.

Recipes

... from `git help reset`

Undo a commit and redo

```
$ git commit ...
$ git reset --soft HEAD^      (1)
$ edit                        (2)
$ git commit -a -c ORIG_HEAD  (3)
```

1. This is most often done when you remembered what you just committed is incomplete, or you misspelled your commit message, or both. Leaves working tree as it was before "reset".
2. Make corrections to working tree files.
3. "reset" copies the old head to .git/ORIG_HEAD; redo the commit by starting with its log message. If you do not need to edit the message further, you can give -C option instead.

See also the --amend option to git-commit(1).

Undo a commit, making it a topic branch

```
$ git branch topic/wip     (1)
$ git reset --hard HEAD~3  (2)
$ git checkout topic/wip   (3)
```

1. You have made some commits, but realize they were premature to be in the "master" branch. You want to continue polishing them in a topic branch, so create "topic/wip" branch off of the current HEAD.
2. Rewind the master branch to get rid of those three commits.
3. Switch to "topic/wip" branch and keep working.

Further reading

- `$ git help reset`
- Reset, Checkout, and Revert
- Reset Demystified

```
$ git help checkout


GIT-CHECKOUT(1)                     Git Manual
GIT-CHECKOUT(1)


NAME
       git-checkout - Switch branches or restore working tree fi
les

SYNOPSIS
       git checkout [-q] [-f] [-m] [<branch>]
       git checkout [-q] [-f] [-m] --detach [<branch>]
       git checkout [-q] [-f] [-m] [--detach] <commit>
       git checkout [-q] [-f] [-m] [[-b|-B|--orphan] <new_branch
>] [<start_point>]
       git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [
<tree-ish>] [--] <paths>
...
       git checkout [-p|--patch] [<tree-ish>] [--] [<paths>...]
```

As you can see in the **NAME** section this command can do 2 things:

1. switch branches
2. restore working tree files

So, this command can change current **HEAD** and can change current working tree

---

Further reading

- What's it "--"?
- What's it "tree-ish"

1. Rename a local and remote branch in git
2. Pro Git book
3. Atlassian's Tutorials

Here you can find usefull tools for Mac OS X

- “tree” Command on Mac OS X
- Draw diagrams
- SourceTree - A free Git client for Windows and Mac

Common

Cleanup

`git-prune` - Prune all unreachable objects from the object database

```
git prune
```

Commit

Rename the last commit message

Sometimes I want to edit the last local commit message:

```
git commit --amend
```

Configured editor will appear. After editing save and close it. The commit's message will be updated

Or to re write the whole commit message in one line:

```
git commit --amend -m "New commit message"
```

Add a file to the last local commit

```
git add file-name
git commit --amend
```

Branches

Change branch HEAD

```
git reset --hard HEAD~1
git reset --hard <COMMIT_HASH>
```

Reset

- Remove files from staged area

  ```
  git reset
  ```

- Reset completely to `HEAD` state

  ```
  git reset --hard
  ```

- Pull remote ignoring local branch

  ```
  git fetch --all
  git reset --hard origin/<branch_name>
  ```

Checkout and track a remote

```
git checkout -b develop --track origin/develop
```

Rename your local branch.

If you are on the branch you want to rename:

```
git branch -m new-name
```

If you are on a different branch:

```
git branch -m old-name new-name
```

Delete remote branch.

```
git push origin :old-name
```

Delete the old-name remote branch and push the new-name local branch.

```
git push origin :old-name new-name
```

Reset the upstream branch for the new-name local branch.

Switch to the branch and then:

```
git push origin -u new-name
```

Push branch to remote by force

It's useful after local rebase a branch which was pushed to remote

```
git push -f origin branch-name
```

List of local branches

```
git branch
```

List of all branches

```
git branch --all
```

or

```
git branch -a
```

Fetch

Remove untracked local branches

`-p` , `--prune` After fetching, remove any remote-tracking branches which no longer exist on the remote

```
git fetch --prune
```

Tags

Delete remote tag

```
git tag -d tag-name
git push origin :refs/tags/tag-name
```

Move tag to other commit and push it to remote

```
git tag --force v1.0 <NEW-COMMIT-HASH>
git push --force --tags
```

> Stackoverflow question

Remove remote

```
git tag -d release01
git push origin :refs/tags/release01
```

> How do I remove or delete a tag from a Git repo

Rebase

Interactive Rebase

Start intercative rebase last **3** commits

```
git rebase -i HEAD~3
```

Here you can find special git commands

`cat-file`

Git has a tool to inspect its files: `cat-file`

git-cat-file - Provide content or type and size information for repository objects

- To get object **type**

```
$ git cat-file -t <HASH>
```

- To get object **content**

```
$ git cat-file -p <HASH>
```

Further reading

- `cat-file` at Pro Git

The `Index` next proposed commit snapshot

> https://git-scm.com/blog

What is HEAD?

`HEAD` is the snapshot of your last commit, next parent

```
$ cat .git/HEAD
ref: refs/heads/master

$ cat .git/refs/heads/master
e9a570524b63d2a2b3a7c3325acf5b89bbeb131e

$ git cat-file -p e9a570524b63d2a2b3a7c3325acf5b89bbeb131e
tree cfda3bf379e4f8dba8717dee55aab78aef7f4daf
author Scott Chacon  1301511835 -0700
committer Scott Chacon  1301511835 -0700

initial commit

$ git ls-tree -r cfda3bf379e4f8dba8717dee55aab78aef7f4daf
100644 blob a906cb2a4a904a152...   README
100644 blob 8f94139338f9404f2...   Rakefile
040000 tree 99f1a6d12cb4b6f19...   lib
```

What is current branch?

It's the branch which **HEAD will point to the next** created **commit** automatically

How Git knows which branch is current?

Inside `.git` folder there is a **simple text** file `HEAD` .

- Current brunch is `feature` (**points on a reference**)

  ```
  $ cat .git/HEAD
  ref: refs/heads/feature
  ```

- Detached `HEAD` state, **points on commit** `51d8df...`

```
$ cat .git/HEAD
51d8dfa0322483a91adda2ef2ef4fe1319ae5a1b
```

You can edit this file manually and it will change current branch.

What's it "--"?

The `--` separates the paths from the other options.

Image that you see this command:

```
$ git checkout A B C
```

What is `A`? `B` and `C`? Are they file names or git will checkout files `B` and `C` from **branch** `A`?

In “git checkout -- files”, what does “--” mean?

What's it `<tree-ish>` ?

> Tree-ish" is a term that refers to any identifier (as specified in the Git revisions documentation) that ultimately leads to a (sub)directory tree" by user456814 @ https://stackoverflow.com/a/18605496/2374209

For example:

- branch
- tag
- commit
- ref, like HEAD~3,

1. `init`
    i. create repo
    ii. `tree` - show .git contents
    iii. Source Tree
2. `add`
    i. create a file: `echo "111" > 1.txt`
    ii. `add .` to staging
    iii. `tree` - show .git contents
    iv. `git hash-object 1.txt`
    v. `mkdir files`
    vi. create file `echo "111" > files/2.txt`
    vii. `add files/2.txt` to staging
    viii. `tree` - show .git contents - NO NEW FILE!
3. `commit`
    i. create clean repo
    ii. `init`
    iii. create new file `1.txt` : "111"
    iv. `commit -m "Initial commit"`
    v. `tree` - show .git contents
    vi. OBJECTS ARE EQUAL LESS COMMIT!!!
    vii. `git cat-file -t`
    viii. `git cat-file -p`
    ix. Show first image
    x. Create another commit:

```
$ echo "222" > 2.txt
$ git add .
$ git commit -m "Second commit"
```

    xi. `tree` - show .git contents - NO NEW FILE!
    xii. `git cat-file -t`
    xiii. `git cat-file -p`
    xiv. show third image
4. `tag`

i. Add - Light first: `git tag the-beginning`
    i. `tree .git/refs`
    ii. `cat .git/refs/tags/the-beginning` JUST POINTS ON COMMIT
    iii. `git cat-file -t the-beginning`
    iv. `git cat-file -p the-beginning`
    v. `git tag first HEAD~2`
    vi. `git tag seconds develop`
ii. List `git tag`
iii. Show `git tag show the-beginning`
iv. Delete `git tag -d the-beginning`
v. Annotated `git tag -a -m "My annotation" another-feature`
    i. `git cat-file -p another-feature`
    ii. `tree .git/refs`
    iii. `cat .git/refs/tags/another-feature` POINTS ON COMMIT and HAS ANNOTATION + CREATOR INFO