Complete Git Guide : Understand and master Git and GitHub

(udemy course)

Section 1 → INTRODUCTION TO THE GIT AND GITHUB || DIFFERENCE BETWEEN GIT AND GITHUB

Section 2 → INSTALLATION OF THE GIT

Section 3 → BASIC SHELL COMMANDS

Section 4 → HOW GIT WORKS UNDER THE HOOD

Section 5 → BASIC GIT COMMANDS

Section 6 → GIT BRANCHES AND HEAD

Section 7 → CLONING PUBLIC REPOSITORIES

Section 8 → MERGING BRANCHES

Section 9 → GITHUB AND REMOTE REPOSITORIES

Section 10 → GIT PUSH, FETCH AND PULL

Section 11 → PULL REQUESTS

Section 12 → FORKS AND CONTRIBUTION TO THE PUBLIC REPOSITORIES

Section 13 → GIT TAGS

Section 14 → REBASING

Section 15 → IGNORING FILES IN GIT

Section 16 → DETACHED HEAD

Section 17 → ADVANCED GIT

Section 18 → GITHUB PAGES

Section 19 → GIT HOOKS

Section 1 : Introduction to the Git and GitHub

What is version control?

revision control

source control

Version control is management of changes made to the documents but it isn't limited to tracking changes of different version of those documents

Version control can actually contain a bunch of information about who made the changes, but it also contains when was these changes made and what were the actual changes down to the line

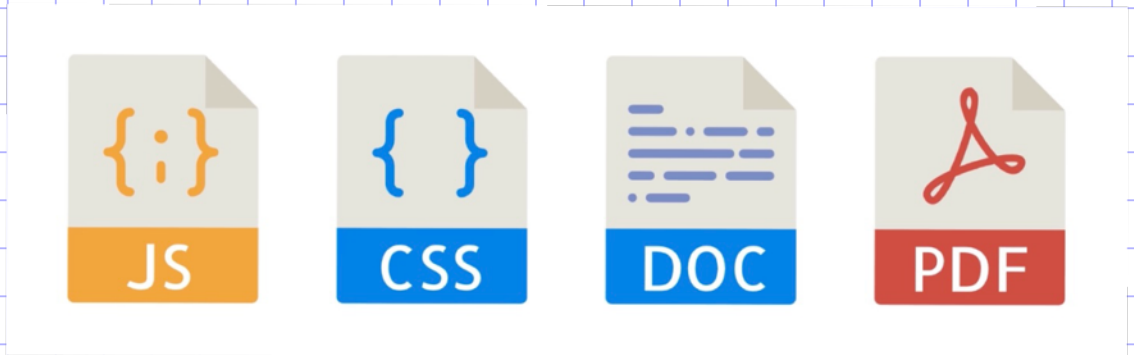
Why is version control so important?

→ keeping track of changes

→ ability to go back to a previous working version

→ easily add someone else's work into our own

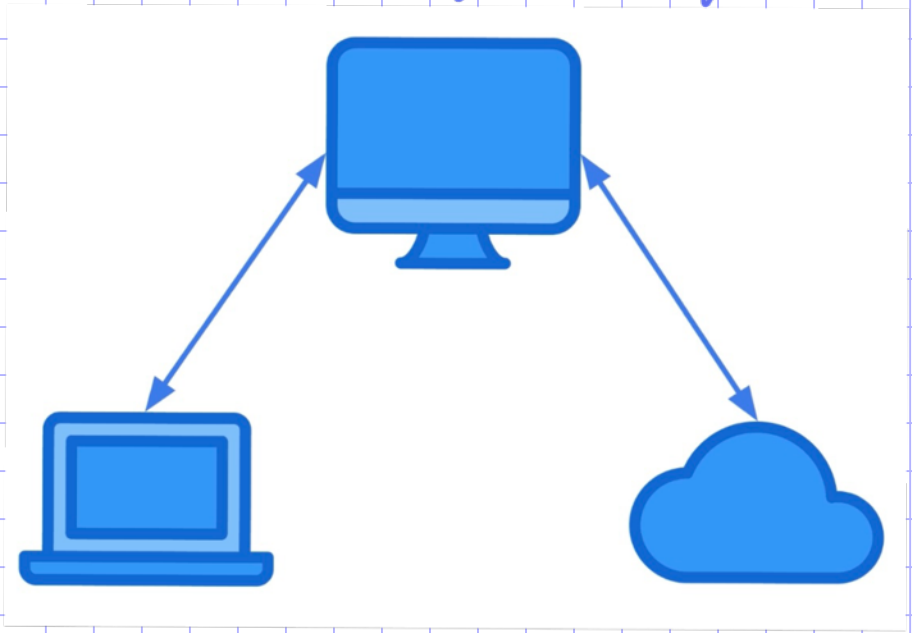
Where can I use git version control?



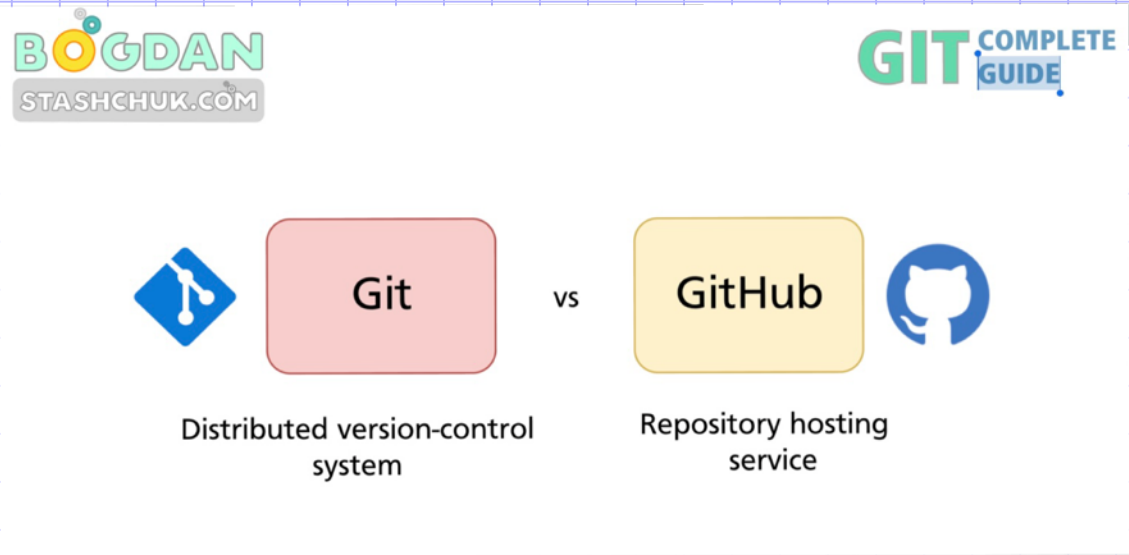
How do development teams use version control?

Eventually when more people join a team it is important to share code with them, so they can collaborate on a single respository, so more than one person has access to the exact same code.

And together they can make changes to exact same files and have version control manage these changes, keep track of them and know that we can always go back to a version if we did something wrong.



What is Git and what is GitHub and how are they actually related?



#Git

→ Git can track changes in files in any folder called resposittory

→ Git preserves history of all the changes that you make with fiils in this resposi tory

But how git does that? How it stores all the changes of all files in the folder under the hood?

→ Under the hood, git has its own file structure and every file is stored in a separate document and every file has special hash and that's why we can say that git is simply persistent hashmap that stored key-value pairs.

→ Key is the hash of the file and value is the content of the file

→ Git could be used locally on your computer and locally you will have full access to entire history of all changes that you made in the repository. In other words, locally you will

have full access to the files changes history.

→ Git doesn't depend on internet connection.

→ Git stores all the changes locally.

→ It stores all the data locally on your computer and that is a great benefit of git.

But why is it called distributed version control System ?

That's because you may collaborate with other people and work on the same project.

And that's where Github comes in :-

#Github

→ Github helps to manage your git repositories and usually Github is used when different people work on the same project.

→ But of course you may use github, even if you dont have any other collaborators. You might simply use Github as a backup service for your repositories.

→ In short, you may use Github just for backup of your local Git repositories.

→ Even if you delete you local repository, remote repository at Github will still be available and you will be able to load your remote repositories anytime from Github again locally to your computer and continue to work in them. Distributed teams of people that want to work on the same projects will benefit from Github the most, i.e. Github is used primarily by distributed teams that work on the same projects

SUMMARY

Git

→ Git is distributed version control system

→ If you want to use git, you must install it on your local computer and after installation of git you can start tracking any changes in any of your folders on your local computer.

→ And again, for that you don't require any internet connection and under the hood git in fact is special file system that allows you to store changes that you make in any folder.

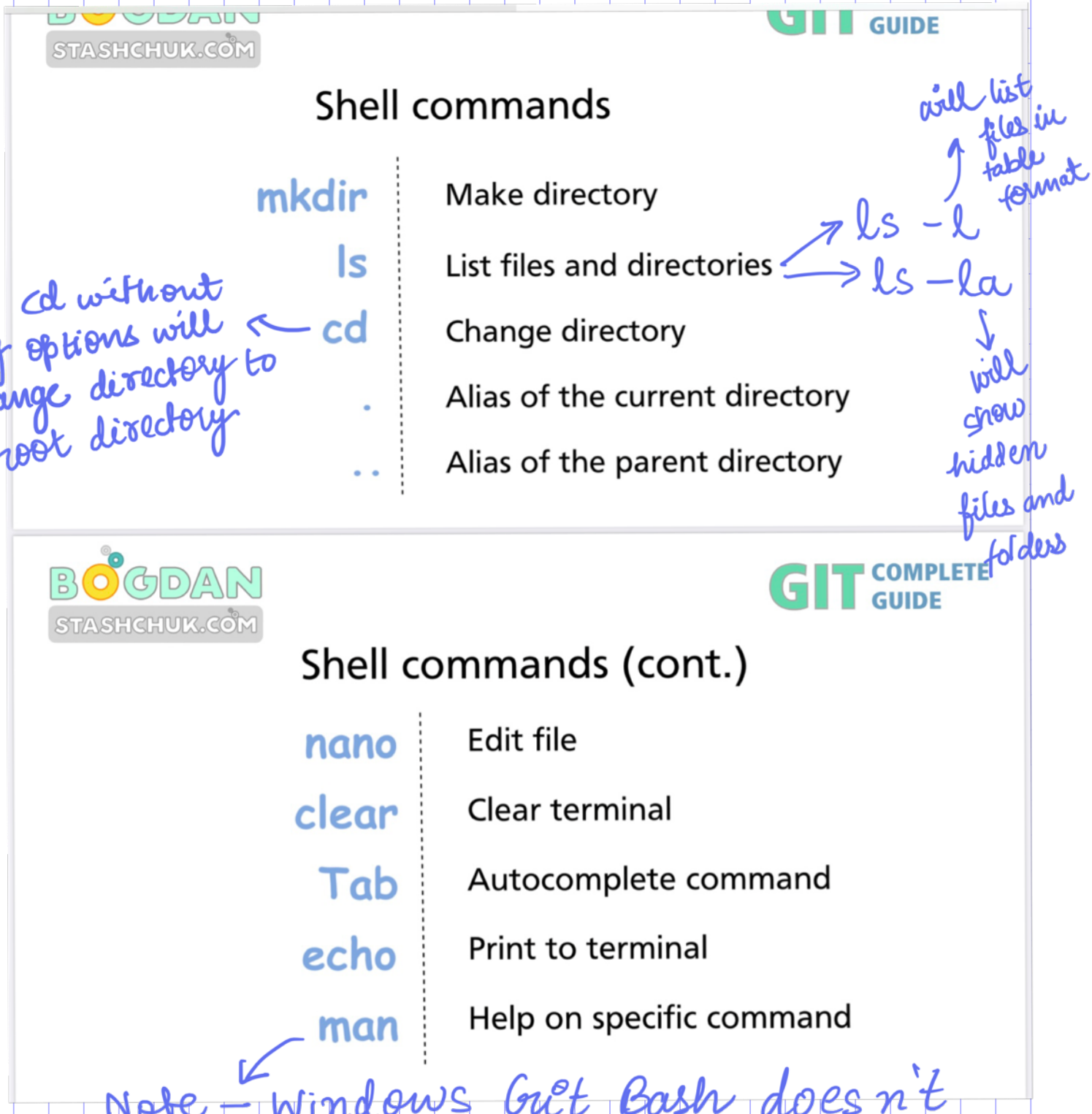
Github

→ Github is a repository hosting service and it allows you to create remote copies of your local git repositories and more than that it allows us or people to collaborate on your repositories, - download, change and so on.

→ Github has many features like website hosting and Github Desktop. With Github, you are able to create website with hosting domain. It will be subdomain of github.io and you will also get the free SSL encryption.

Section 2 : Installation of the Git and configuration of the Shell

Section 3: Basic Shell Commands



BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Shell commands (cont.)

touch	Create new file
>	Write to the file
>>	Append to the file
cat	List contents of the file
rm	Remove files and directories

rm -rf → for directories (folders)

Section 4: How Git works under the hood?

You should enter this command while you are already in the new folder or existing folder you want to create a new repository and for that you should first use cd command or if directory is absent, you should first create it using mkdir command and after that cd to newly created directory.

And only when you are already in the required directory you could use git init command and this command will actually create new empty git repository and after the creation of new git repository, you will be able to add new files to it.

NOTE: Git repository will be initially empty even if directory where you initialize Git is not empty and contains other directories.

But how it actually works? What allows you to add files to git repository?

Under the hood, "git init" command will create new hidden folder called .git and you can explore the contents of .git folder.

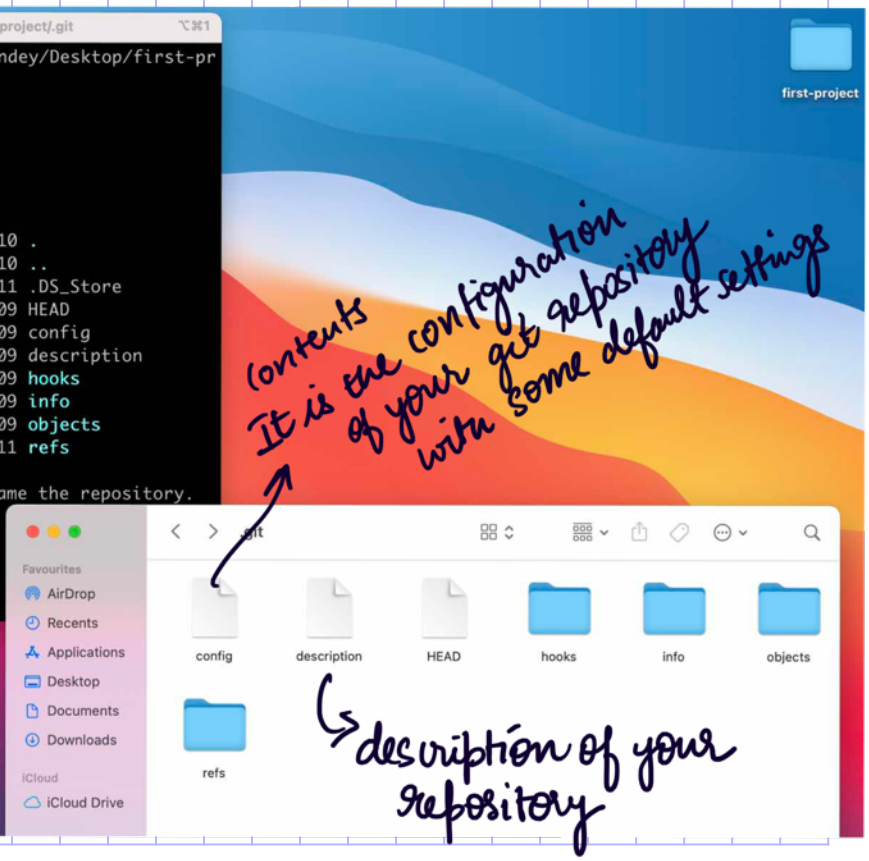
```
Last login: Thu Aug  5 13:29:32 on ttys001
➜  ~ cd Desktop
➜  Desktop mkdir first-project
➜  Desktop ls -la
total 16
drwx------+  5 hemakshipandey  staff   160 Aug  5 16:39 .
drwxr-xr-x+ 20 hemakshipandey  staff   640 Aug  5 16:39 ..
-rw-r--r--@  1 hemakshipandey  staff  6148 Aug  5 16:39 .DS_Store
-rw-r--r--   1 hemakshipandey  staff     0 Jul 26 19:51 .localized
drwxr-xr-x   2 hemakshipandey  staff    64 Aug  5 16:39 first-project
➜  Desktop cd first-project
➜  first-project git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/hemakshipandey/Desktop/first-project/.git/
➜  first-project git:(master) ls -la
total 0
drwxr-xr-x  3 hemakshipandey  staff   96 Aug  5 16:39 .
drwx------+ 5 hemakshipandey  staff  160 Aug  5 16:39 ..
drwxr-xr-x  9 hemakshipandey  staff  288 Aug  5 16:39 .git
➜  first-project git:(master) ls -l
➜  first-project git:(master)
```

default branch is master and after the initialization of repository you will be on the master branch

indicates that this directory is in git control

OVERVIEW OF .GIT FOLDER

```
hemakshipandey@Hemakshis-MacBook-Pro:~/Desktop/first-project/.git  ⌥⌘1
Initialized empty Git repository in /Users/hemakshipandey/Desktop/first-pr
oject/.git/
➜  first-project git:(master) open .
➜  first-project git:(master) cat HEAD
cat: HEAD: No such file or directory
➜  first-project git:(master) ✗ cd .git
➜  .git git:(master) ls -la
total 48
drwxr-xr-x  10 hemakshipandey  staff   320 Aug  5 18:10 .
drwxr-xr-x   4 hemakshipandey  staff   128 Aug  5 18:10 ..
-rw-r--r--@  1 hemakshipandey  staff  8196 Aug  5 18:11 .DS_Store
-rw-r--r--   1 hemakshipandey  staff    23 Aug  5 18:09 HEAD
-rw-r--r--   1 hemakshipandey  staff   137 Aug  5 18:09 config
-rw-r--r--   1 hemakshipandey  staff    73 Aug  5 18:09 description
drwxr-xr-x  15 hemakshipandey  staff   480 Aug  5 18:09 hooks
drwxr-xr-x   3 hemakshipandey  staff    96 Aug  5 18:09 info
drwxr-xr-x   4 hemakshipandey  staff   128 Aug  5 18:09 objects
drwxr-xr-x   5 hemakshipandey  staff   160 Aug  5 18:11 refs
➜  .git git:(master) cat description
Unnamed repository; edit this file 'description' to name the repository.
➜  .git git:(master) cat HEAD
ref: refs/heads/master
➜  .git git:(master) cat refs
cat: refs: Is a directory
➜  .git git:(master)
```



(contents) It is the configuration of your git repository with some default settings

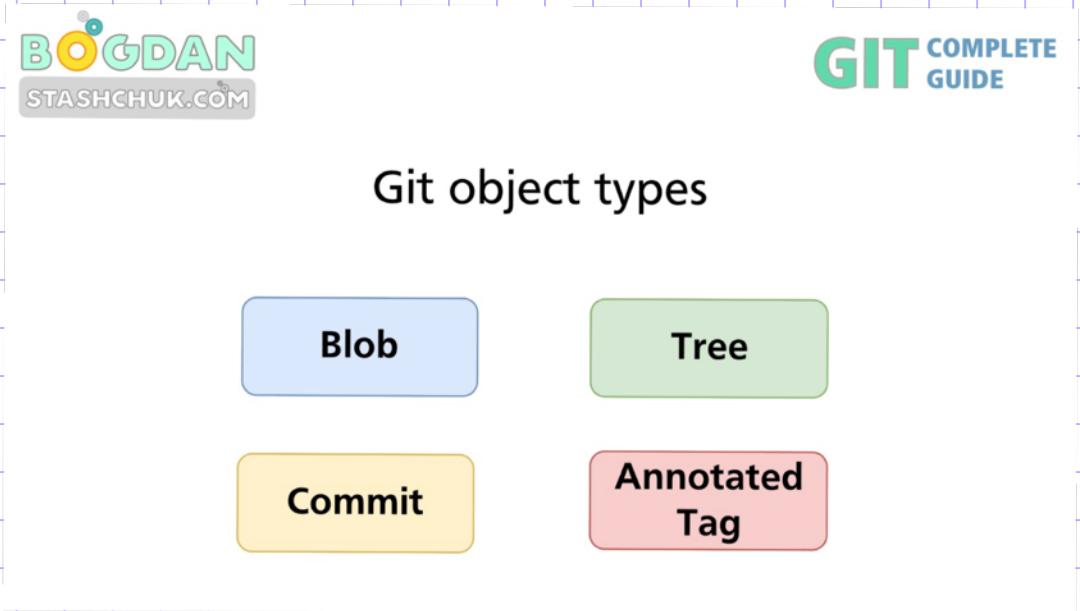
description of your repository

GIT OBJECT TYPES

→ We have explored hidden .git folder and it is highly not recommended to make any changes in this folder manually.

→ Git has it's own file system and it stores its own objects and there are several types of objects that git can store.

→ There are four object types in git.



→ These four object types allow git to store all necessary data required for file taken for, tracking of changes of the file and so on.

So what do object types represent?

Blob : In blob object types git stores files; any file with any extensions, either video files or pictures or text files. Any files of any types are stored as blob.

Tree: with the help of tree object type, git actually stores information about directories and the same as in any other file system, directory may be empty or contain files or contains other directories or have mix of files and directories. Trees indeed may be set of blobs or set of blobs and other trees. Tree is representation of folder or directory in git

Commit: With help of commit object type, we are able to actually store different versions of our project.

Annotated Tag: Annotated tag is actually persistent text pointer to specific commit.

→ Blob represents a single file in a file system and Tree represent directory or folder

→ For management of blobs and trees, we will use some low level git commands like :—

GIT LOW LEVEL COMMANDS

git hash-object

git cat-file

→ with the help of git hash-object, we will be able to create new object in structure.

→ with the help of git cat-file command, we will be able to read git objects,

git mktree

→ with the help of git mktree command we will be able to create new tree object.

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Git low-level commands

git hash-object

git cat-file

git mktree

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

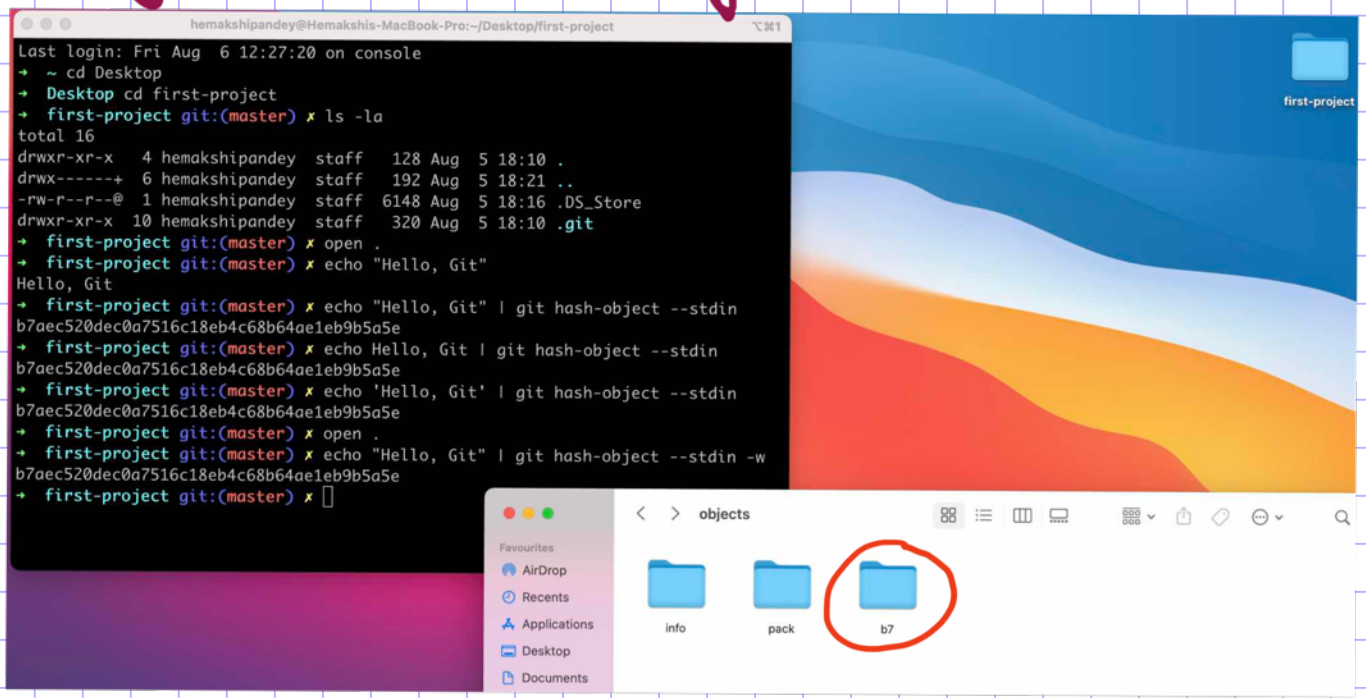
git hash-object

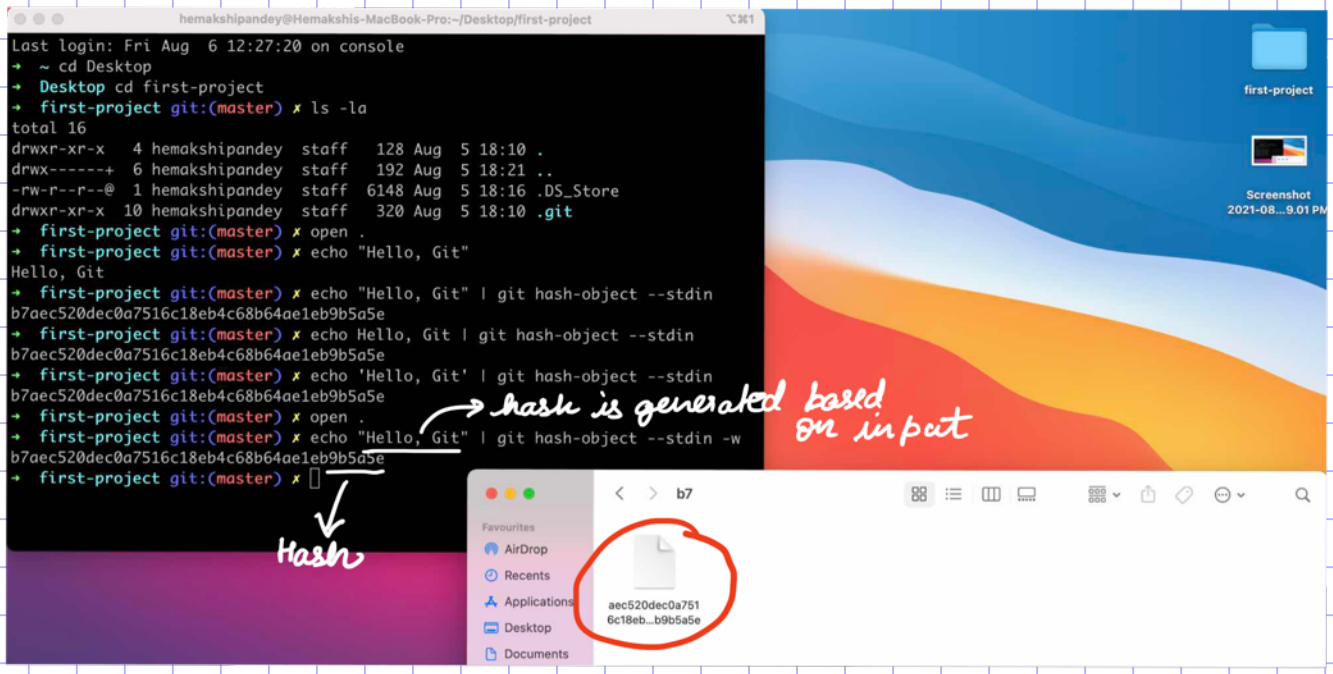
```
echo "Hello, Git" | git hash-object --stdin -w

git hash-object <filename> -w
```

Take input from standard input

Create Git Object.

git hash-object





Folder name + File name = Hash

→ We have used git hash object command and with the help of this command. We have created new git object that was written to objects folder and new folder (b7) inside of the object folder was creakd and new file inside of this folder was creakd (ae.... se)

→ It means that Git creates names for folders and files inside of those folder based on hash and hash is generated based on Input.

JSON v/s GIT database

→ This format is used for data exchange b/w different servers. For example, you can retrieve some data via API application programming interface from server web browser. You can also send data from database in JSON format to another server and so on. It is just format for data exchange.

→ We can see in example the sample structure of JSON Object.

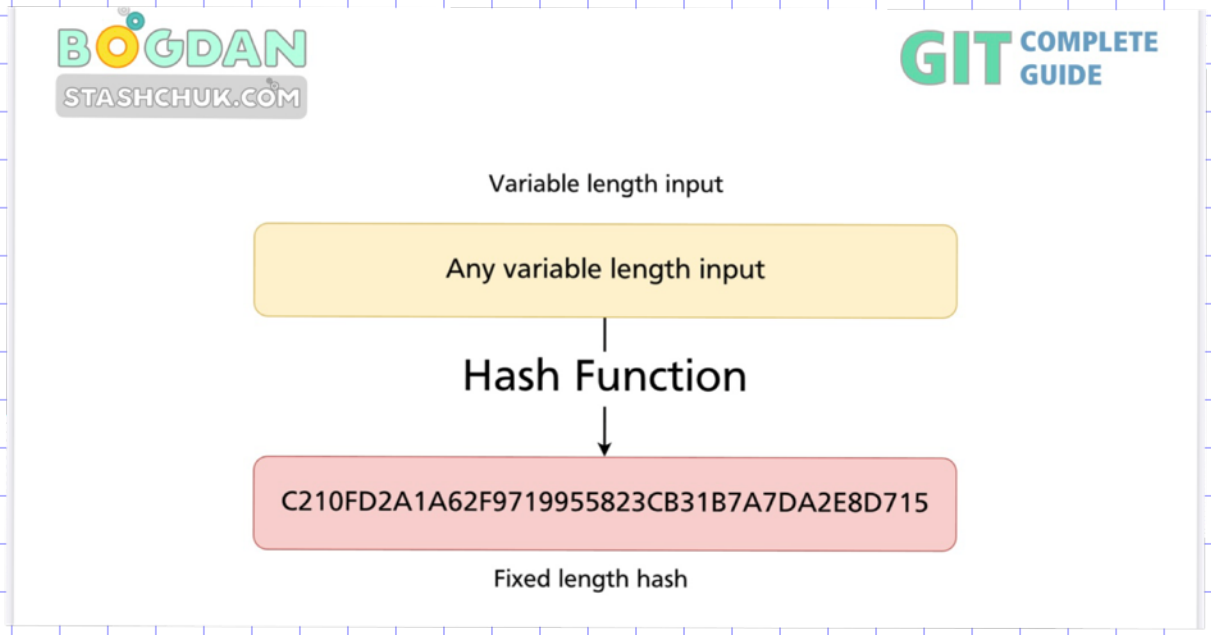
→ {} → curly braces represents starting and ending point for this object.

→ Object itself consist of set of keys value pairs.

→ Keys here are all distinct. It's not possible to have same keys in same JSON object, values can be same. In Git, key is generated based on value.

NOTE: According to specification keys in JSON should be unique and it is technically possible to create same key, but not recommended.

WHAT IS HASH FUNCTION ?



→ Hash function is a function that takes any variable length input and creates fixed length hash independent of the input.

→ Note: Hash is generated based on the contents of the file

→ Any file works here and hash function will always create hash of fixed lengths and that is very important characteristic of hash function.

→ If you know hash but do not know the input, you are not able to create the input based on hash and such functions are called one way functions

→ We are able to create hash based on input but we are not able to find out which input was used to create specific hash. That's why very often, passwords that you are entering on any website are stored in databases as hashes.

Example :- For higher security and confidentiality user passwords are usually stored in databases as hashes.

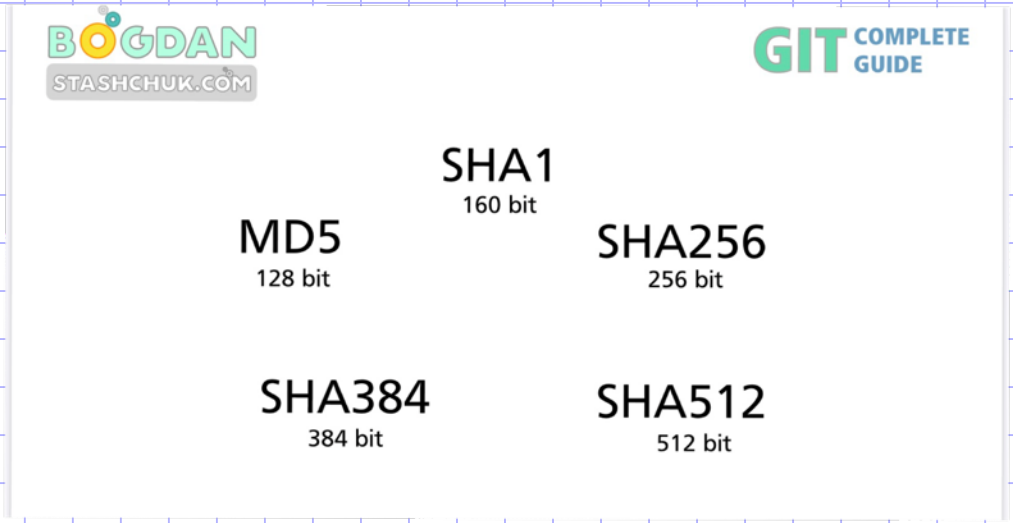
When user logs in and enters password, server creates hash of entered password and compares it to the hash stored in database.

① → Hash functions are "one-way" functions.

② → Same hash function will always create same hash for the same input

HASH FUNCTION OVERVIEW

→ There are many hash functions but here, you see most popular, example MD5

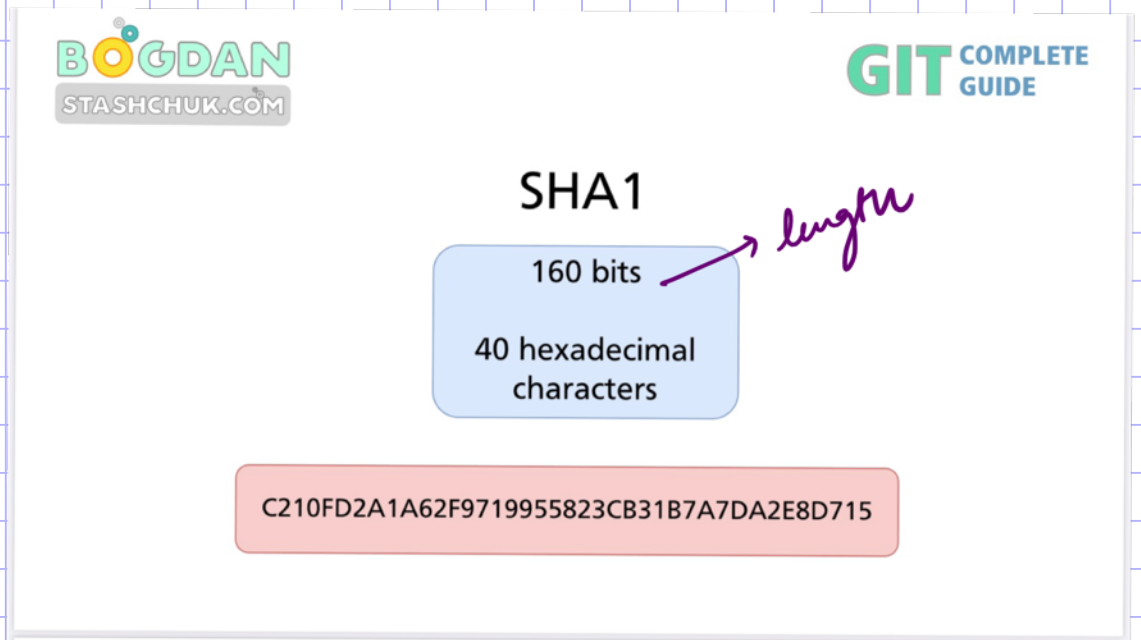


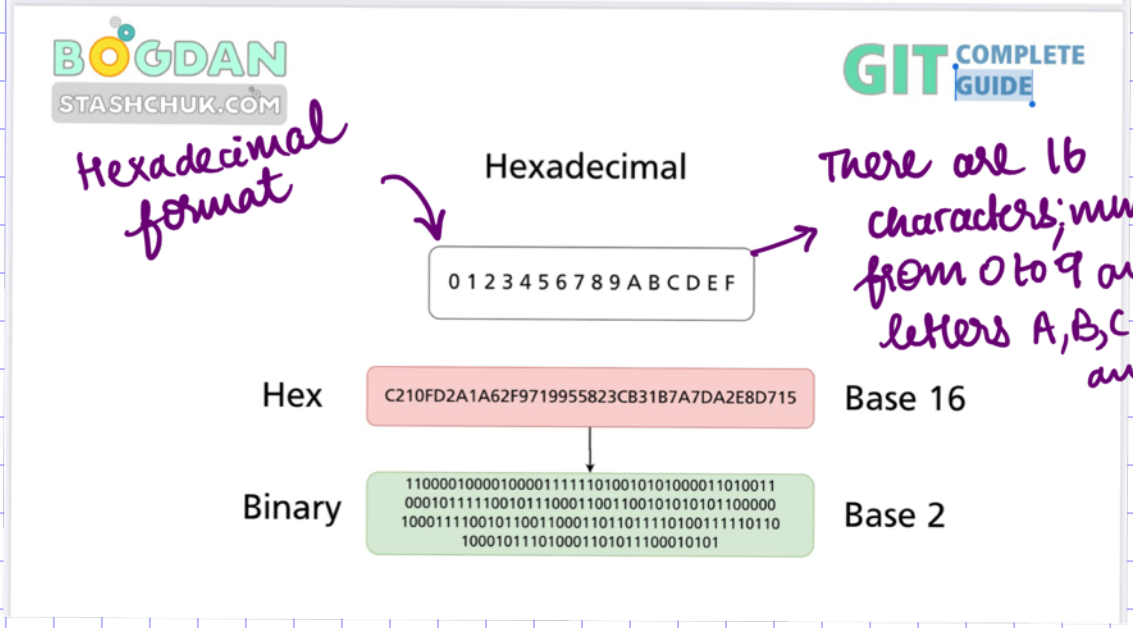
→ you can try to generate MD5 and SHA hash online.

→ There are several hash function and git utilizes SHA1 hash function.

SHA1 Hash function

→ SHA1 hash function is used by git.



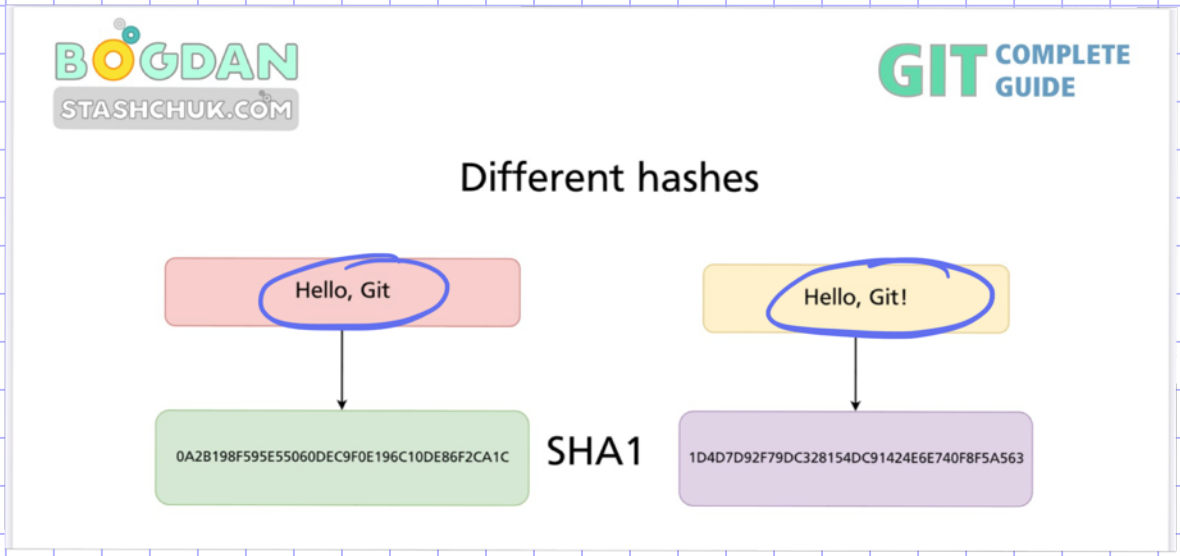


→ Binary format is used under the hood on your computer for storage of all data.

→ All data is converted to binary format and after that stored on specific media (eg- harddisk).

→ Binary format takes much more space and it is really hard to read for heeman. That's why base 16 format is used for representation of hash and using hexadecimal format we can very fast and easy compare two hashes and we can just look at the first

two or three characters and decide whether our hashes are equal or not, that is, hexadecimal format.



→ There is only a single small difference in those strings. But if we will create SHA1 hashes of those strings we will get two completely different hashes and that is one more characteristic of every hash function. Even small change of input data will lead to creation of completely different hash. That's why it is hard to guess input based on hash, if you don't know input.

```
..first-project (-zsh)                                    ⌘1
Last login: Fri Aug  6 13:56:06 on ttys000
➜  ~ man shasum
[1]  + 1390 suspended  man shasum
➜  ~ clear
➜  ~ echo "Hello, Git"
Hello, Git
➜  ~ echo -n "Hello, Git"
Hello, Git%
➜  ~ echo "Hello, Git" | shasum
c9d5d04925b93d2fb99c73ab2b5869bde7405ca4  -
➜  ~ echo -n "Hello, Git" | shasum
0a2b198f595e55060dec9f0e196c10de86f2ca1c  -
➜  ~ echo -n 'Hello, Git!' | shasum
1d4d7d92f79dc328154dc91424e6e740f8f5a563  -
➜  ~
```

→ print or check SHA checksums

→ -n to avoid automatic line break

→ It is important because hashes from the strings with line break in the end and without it will be completely different.

→ This hash was generated without ending line break.

→ even small change in input data will lead to different hash.

How many files Git could store

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

How many files Git can store in the same repository?

What is the chance of producing same **exact** hash for different files?

What is the chance of producing same **any** hash for different files (hash collision)?

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Combinations

$2^2 = 4$

00
01
10
11

$2^3 = 8$

000	100
001	101
010	110
011	111

→ only two possible character- 0 and 1

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

SHA1 combinations

2^{160}

Ans to the Ques How many files git can store in the same repository

1461501637330902918203684832716283019655932542976

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Probability theory in dice game

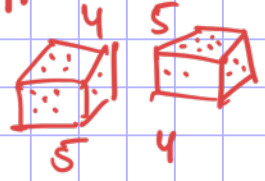
$$\frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$

Exact combination probability when **order does matter**

0.028 ~ 3%

↳ probability of the same numbers of two dices

Ques:- what is the probability of getting pair of the two different number? eg pair (4,5)



Probability of the pair of non matching numbers like (4,5) is 0.028 * 2 = 0.056 ~ 6%

It is because order of the numbers doesn't matter We can get (4,5) or (5,4) on a pair of dice

Git hash collision probability

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

$$\frac{1}{2^{160}}$$

Combinations

what is the Probability of each SHA1 hash?

6.84e-49

→ very very small value

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

what is Probability of the same specific SHA1 hash?

$$\frac{1}{2^{160}} \times \frac{1}{2^{160}} = \frac{1}{2^{320}}$$

2^{320} = 2.13e+96

$\frac{1}{2^{320}}$ = 4.68e-97 = 0.000...004

96 zeros

→ much much small value

→ Hash collision probability is very very low.

More details on hash collision probability

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Probability of **any** same number on **two** dices

$$\frac{1}{6} \times \frac{1}{6} \times 6 - \frac{1}{36} \times 6 - \frac{6}{36} - \frac{1}{6}$$

Probability **0.17**

Probability of **any** number on the **single** dice

Possible pairs

1	1
2	2
3	3
4	4
5	5
6	6

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Probability of **any same** number on **three** dices

$$\frac{1}{6} \times \frac{1}{6} \times \frac{1}{6} \times 6 - \frac{1}{216} \times 6 - \frac{6}{216} - \frac{1}{36}$$

Probability **0.03**

Possible combinations

1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6

→ all numbers must match

→ probability of any same number on three dices, we need to get this total quantity of all possible combination i.e 6 and divide them by total quantity of all possible combinations. — 216

N → quantity of dices

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Probability of **any** same number on **N** dices

$$\frac{1}{6} \times \frac{1}{6} \times \dots \times \frac{1}{6} \times 6 - \frac{1}{6^N} \times 6 - \frac{1}{6^{N-1}}$$

Possible combinations

1	1	...	1
2	2	...	2
3	3	...	3
4	4	...	4
5	5	...	5
6	6	...	6

6 possible combinations where all number are the same.

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Any two dices must have same number, but all dices could have same number

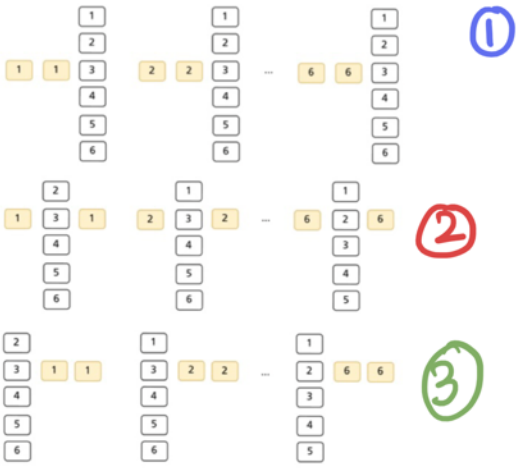
Probability of **any** same number on **any pair** of three dices

$$\frac{1}{6} \times \frac{1}{6} \times \frac{1}{6} \times 96 - \frac{16}{36}$$

Probability 0.44

Matching results $6 * 6 + 5 * 6 + 5 * 6 = 96$

Possible pairs



- ① last dice may have any number, we have six different cases and each case gives us six different combination → $6 \times 6 = 36$

② we dont count the repeated number as it has already been included in case ①. So we have 6 different cases and each case gives us five different combination. $6 \times 5 = 30$

③ 6 different possible cases and each cases produces five different combination. $6 \times 5 = 30$

Probability of all different numbers.

BOGDAN STASHCHUK.COM

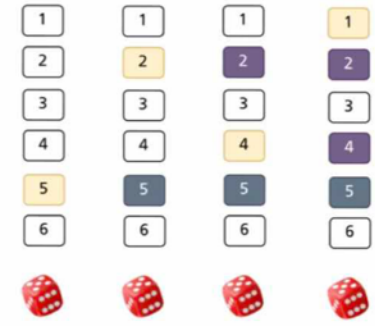
GIT COMPLETE GUIDE

Probability of **all different** numbers on **all** dices

$$\frac{6}{6} \times \frac{5}{6} \times \dots \times \frac{6+1-N}{6} = \frac{5!}{(6-N)! \times 6^{N-1}}$$

if N = 4

Example with 4 dices



$$\frac{6}{6} \times \frac{5}{6} \times \frac{4}{6} \times \frac{3}{6} = \frac{5}{18}$$

Probability **0.28**

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Probability of **any** same number on **any pair** of **N** dices

(for N < = 6)

$$1 - \frac{6}{6} \times \frac{5}{6} \times \dots \times \frac{6+1-N}{6} = 1 - \frac{5!}{(6-N)! \times 6^{N-1}}$$

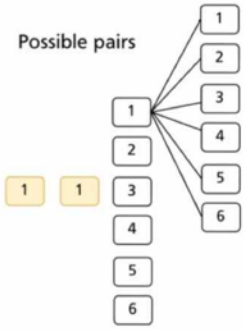
Probability of getting all different numbers on all dices

Examples

N = 2	0.17
N = 3	0.44
N = 4	0.72
N = 5	0.91
N = 6	0.99
N > 6	1

$\frac{1}{6}$

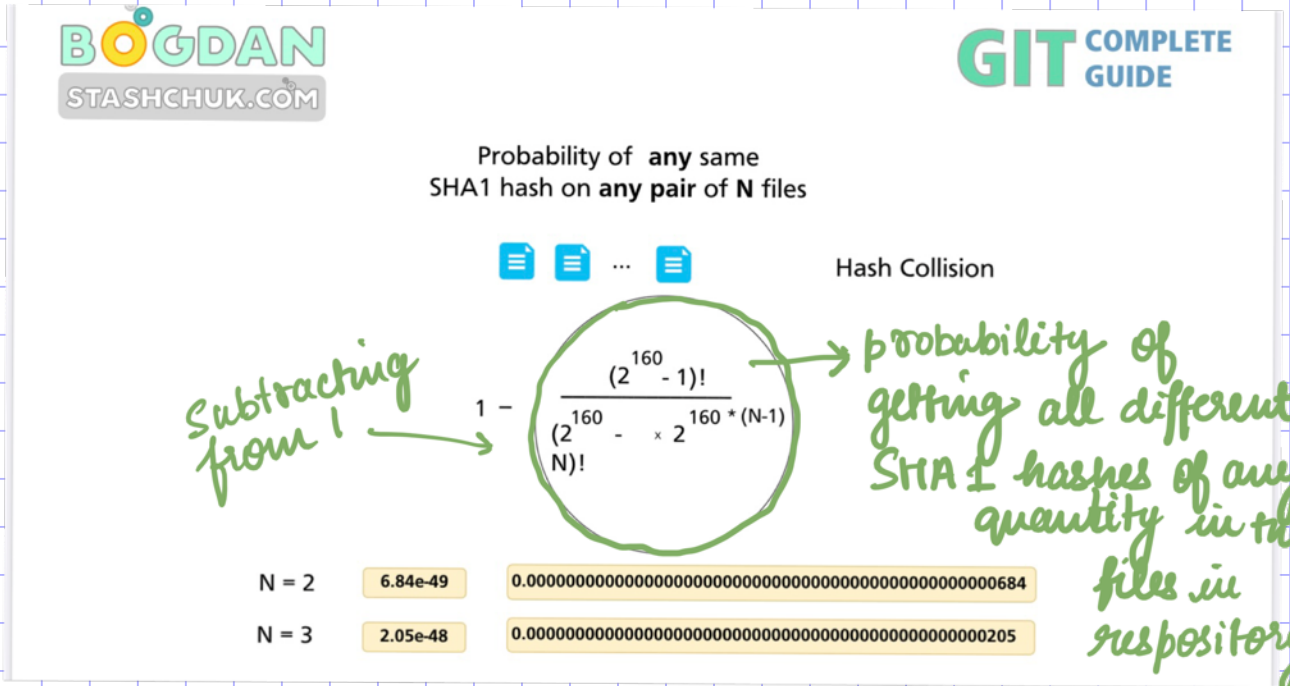
Possible pairs



Question: If we know probability of all different numbers on all dices, what is the probability of getting at least two same numbers?

Ans → $1 - 0.28 = 0.72$ (72.1.)

→ quantity of the files in the repository must be less than 2^{160}



→ If quantity of the files in repository is larger than maximum possible quantity of SHA1 hash combinations then probability of hash collision is 100%.

→ probability of the hash collision grows with quantity of the files in the repository.

Exploring Git objects with git cat-file command

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

git cat-file options

`git cat-file -p <hash>`	Contents of the object
`git cat-file -s <hash>`	Size of the object
`git cat-file -t <hash>`	Type of the object

```
➜  ~ echo Hello, Git | git hash-object --stdin
b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
➜  ~ clear
➜  ~ cd Desktop
➜  Desktop cd first-project
➜  first-project git:(master) ✗ git cat-file -p b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
Hello, Git
➜  first-project git:(master) ✗ git cat-file -t b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
blob
➜  first-project git:(master) ✗ git cat-file -s b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
11
➜  first-project git:(master) ✗ git cat-file -s b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5a
fatal: git cat-file: could not get object info
➜  first-project git:(master) ✗
```

error because there is no such file name

Create new Git Blob based on the file

```
➜  ~ clear
➜  ~ cd Desktop
➜  Desktop echo "Second file in Git repository" > new-file.txt
➜  Desktop ls -la
total 24
drwx------@  7 hemakshipandey  staff   224 Aug 10 19:49 .
drwxr-xr-x+ 20 hemakshipandey  staff   640 Aug 10 20:19 ..
-rw-r--r--@  1 hemakshipandey  staff  6148 Aug 10 20:19 .DS_Store
-rw-r--r--   1 hemakshipandey  staff     0 Jul 26 19:51 .localized
lrwxr-xr-x@  1 hemakshipandey  staff    29 Aug 10 19:26 Relocated Items.nosync -> /Users/Shared/Relocated Items
drwxr-xr-x@  4 hemakshipandey  staff   128 Aug 10 19:48 first-project
-rw-r--r--   1 hemakshipandey  staff    30 Aug 10 20:19 new-file.txt
➜  Desktop cat new-file.txt
Second file in Git repository
➜  Desktop
```

New file could be created using "touch", "nano" or by writing contents to the file using ">".

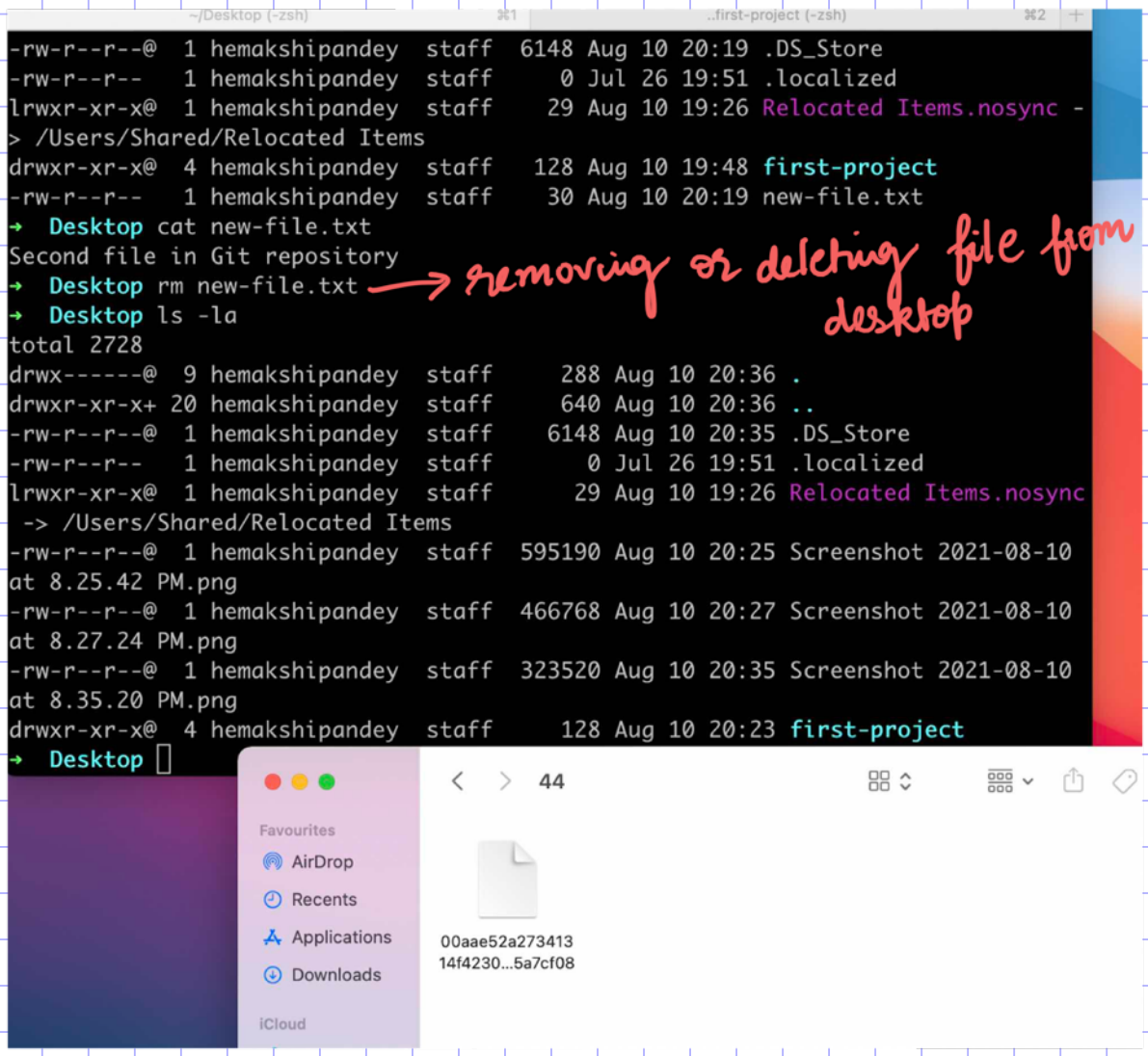
```
➜  ~ cd Desktop
➜  Desktop cd first-project
➜  first-project git:(master) ✗ pwd
/Users/hemakshipandey/Desktop/first-project
➜  first-project git:(master) ✗ git hash-object ../new-file.txt
4400aae52a27341314f423095846b1f215a7cf08
➜  first-project git:(master) ✗ git hash-object ../new-file.txt -w
4400aae52a27341314f423095846b1f215a7cf08
➜  first-project git:(master) ✗
```

parent directory

file name

write git object to repository


```
➜  first-project git:(master) ✗ git cat-file -p 4400aae52a27341314f423095846b1f215a7cf08
Second file in Git repository
➜  first-project git:(master) ✗ git cat-file -t 4400aae52a27341314f423095846b1f215a7cf08
blob
➜  first-project git:(master) ✗ git cat-file -s 4400aae52a27341314f423095846b1f215a7cf08
30
➜  first-project git:(master) ✗
```



→ file will get removed from Desktop but it will remain in git repository.

→ Git repository stores files independently in its own "file system" in the "objects" directory.

Why Git doesn't require filename for the file that you add to the Git database?

Ans → Git blobs don't have filenames

Git blobs don't store filenames

→ "git cat-file" command doesn't have option for retrieving filename from the blob because blob don't store filenames.

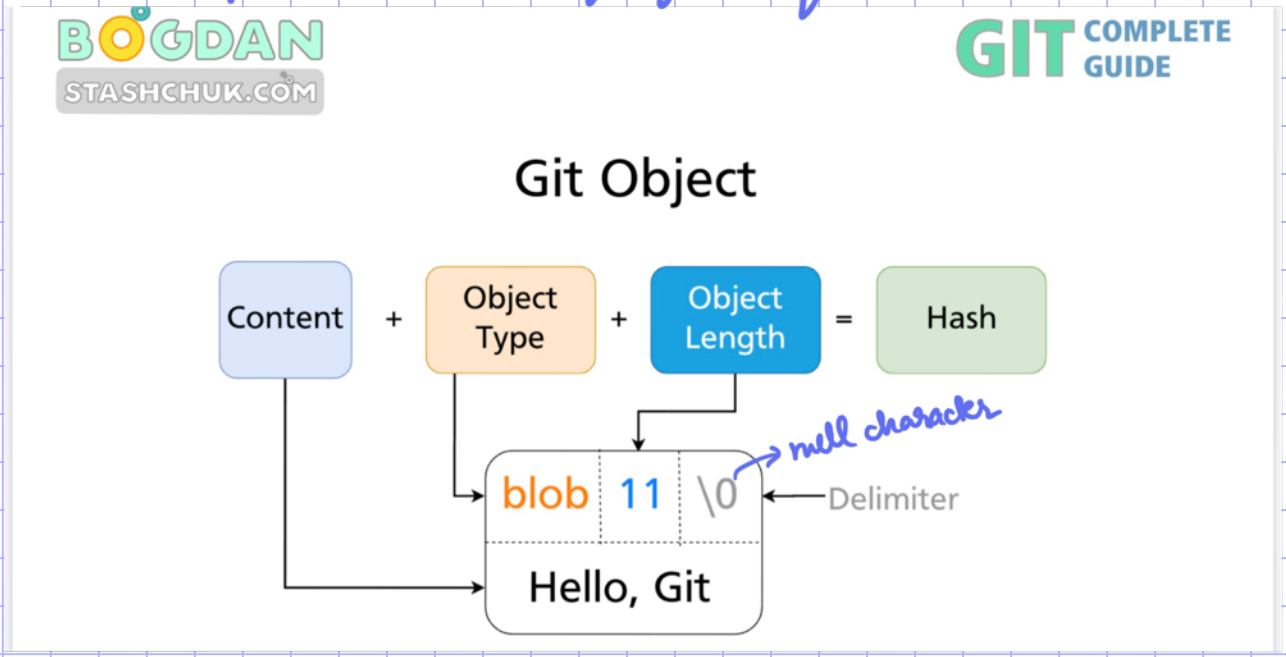
→ where size and type of each object are stored in Git ?

→ this information → size and type of each object is stored directly inside each blob.

→ Git generates SHA1 hash based on the input + type + size.

Contents of Git Objects

→ Git objects consists of four fields :-



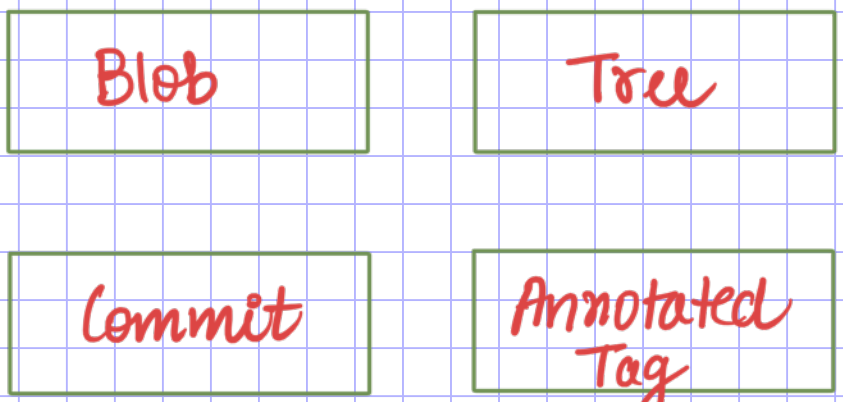

```
➜  ~ cd Desktop
➜  Desktop cd first-project
➜  first-project git:(master) ✗ cd .git/objects
➜  objects git:(master) ls -la
total 24
drwxr-xr-x@  7 hemakshipandey  staff   224 Aug 10 20:23 .
drwxr-xr-x@ 10 hemakshipandey  staff   320 Aug 10 20:23 ..
-rw-r--r--@  1 hemakshipandey  staff  8196 Aug 10 20:38 .DS_Store
drwxr-xr-x@  3 hemakshipandey  staff    96 Aug 10 20:23 44
drwxr-xr-x@  3 hemakshipandey  staff    96 Aug 10 19:48 b7
drwxr-xr-x@  2 hemakshipandey  staff    64 Aug 10 19:48 info
drwxr-xr-x@  2 hemakshipandey  staff    64 Aug 10 19:48 pack
➜  objects git:(master) cd 44
➜  44 git:(master) ls -la
total 8
drwxr-xr-x@ 3 hemakshipandey  staff   96 Aug 10 20:23 .
drwxr-xr-x@ 7 hemakshipandey  staff  224 Aug 10 20:23 ..
-r--r--r--  1 hemakshipandey  staff   46 Aug 10 20:23 00aae52a27341314f423095846
b1f215a7cf08
➜  44 git:(master) cat 00aae52a27341314f423095846b1f215a7cf08
➜  44 git:(master)
```

```
➜  ~ echo "blob 11\0Hello, Git"
blob 11Hello, Git
➜  ~ echo "blob 11\0Hello, Git" | shasum
b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  -
➜  ~ echo -e "blob 11\0Hello, Git" | shasum
b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  -
➜  ~ echo -e "blob 11\0Hello, Git" | shasum
b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  -
➜  ~ echo -e  "blob 11\0Hello, Git"
blob 11Hello, Git
➜  ~ echo "blob 30\0Second file in Git repository"
blob 30Second file in Git repository
➜  ~ echo "blob 30\0Second file in Git repository" | shasum
4400aae52a27341314f423095846b1f215a7cf08  -
➜  ~
```

→ Filenames for the blobs are stored in trees.

Tree Objects in Git

→ Git object types :-



→ Tree is one of four objects type in git.

→ Blobs don't store file names of the source files and each blob consists of contents of the file, type of the file, size and delimeter- backslash zero and name of each blob based on SHA1 hash of each file.

→ Tree object type in git is a representation of directories and each tree may contain blobs and other trees.

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Git Tree Objects

```
100644 blob 57537e1d8fba7d80c5bcca8b04e49666b1c1790f    .babelrc
100644 blob 1fed445333e85fb9996542978fa56866de90a2fb    .flowconfig
040000 tree c167c406a72fd6f3ab232b905cc730588167340e    dist
100644 blob cb82ecb6e7b8f3d4be8f188fc495063aa5ae0c48    package.json
040000 tree 5ddb0cfa6f45d4bce18247297b89353c9ae47af5    src
100644 blob 8ce3a3772a346a19afc3bc9ee3a7fa999b36c5d0    yarn-error.log
100644 blob 7fdb8f594ccda44eba69ea7ce95ea84c93bbfac0    yarn.lock
```

permission

type of object

SHA1 hash

name of files and directories

Git object permissions

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Git objects permissions

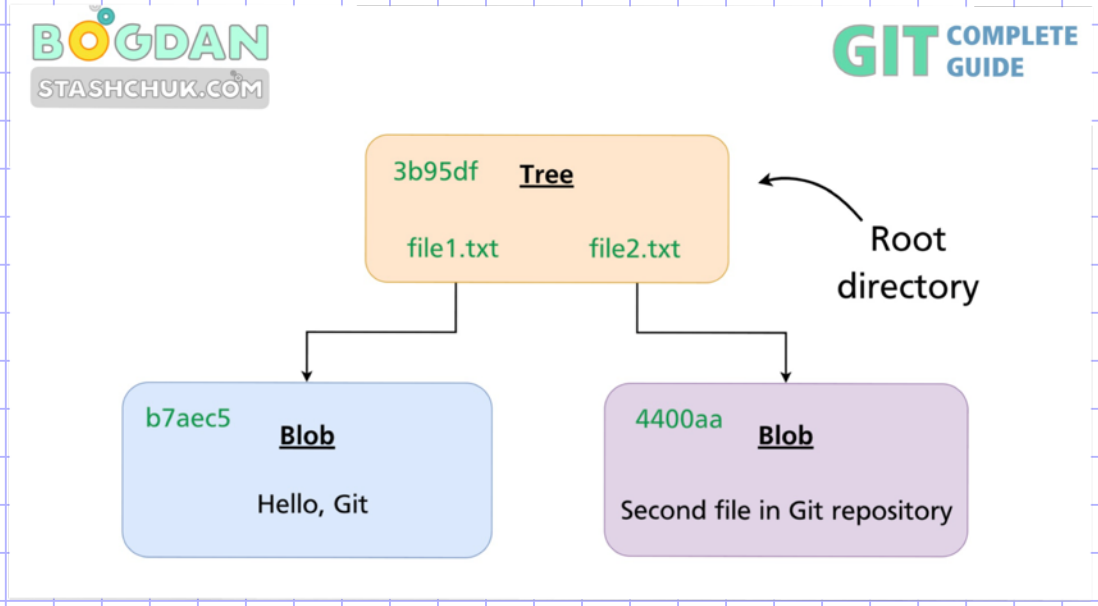
common	**040000**	Directory
	100644	Regular non-executable file
	100664	Regular non-executable group-writeable file
	100755	Regular executable file
	120000	Symbolic link
	160000	Gitlink

→ There are six possible permissions.

Why do we need those git objects permissions?

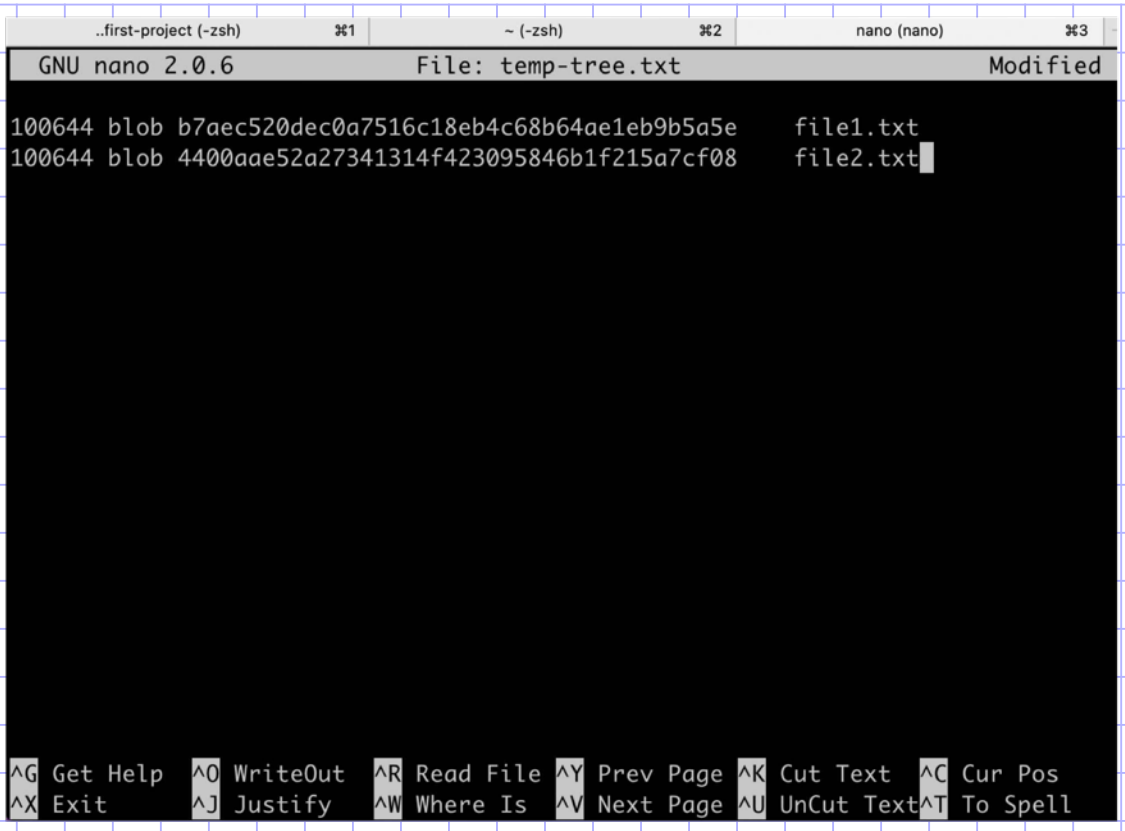
→ Git repository is independent from your file system, this own file system of the git and based on git repository you are able to read files and folders on your computer.

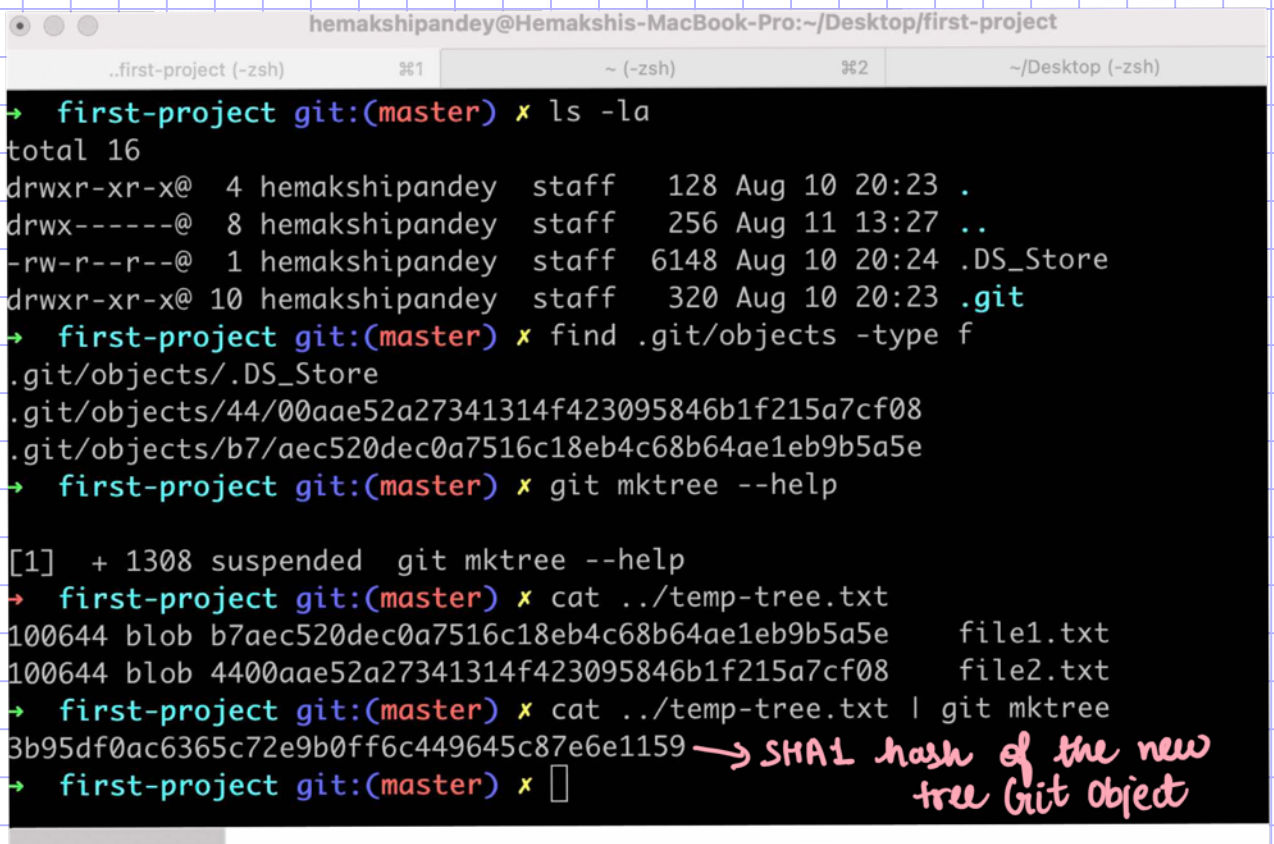
Creating Git Tree Object



→ Git low-level commands → git mktree

```
Last login: Wed Aug 11 13:16:47 on ttys001
➜  ~ cd Desktop
➜  Desktop nano temp-tree.txt
➜  Desktop ls -la
total 920
drwx------@  8 hemakshipandey  staff     256 Aug 11 16:15 .
drwxr-xr-x+ 20 hemakshipandey  staff     640 Aug 11 16:15 ..
-rw-r--r--@  1 hemakshipandey  staff    6148 Aug 11 16:15 .DS_Store
-rw-r--r--   1 hemakshipandey  staff       0 Jul 26 19:51 .localized
lrwxr-xr-x@  1 hemakshipandey  staff      29 Aug 10 19:26 Relocated Items.nosync
 -> /Users/Shared/Relocated Items
-rw-r--r--@  1 hemakshipandey  staff  455361 Aug 11 16:14 Screenshot 2021-08-11
at 4.14.15 PM.png
drwxr-xr-x@  4 hemakshipandey  staff     128 Aug 10 20:23 first-project
-rw-r--r--   1 hemakshipandey  staff     126 Aug 11 16:15 temp-tree.txt
➜  Desktop cat temp-tree.txt
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
➜  Desktop
```


..first-project (-zsh) ⌘1
~ (-zsh) ⌘2
nano (nano) ⌘3
GNU nano 2.0.6 File: temp-tree.txt Modified
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08 file2.txt
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

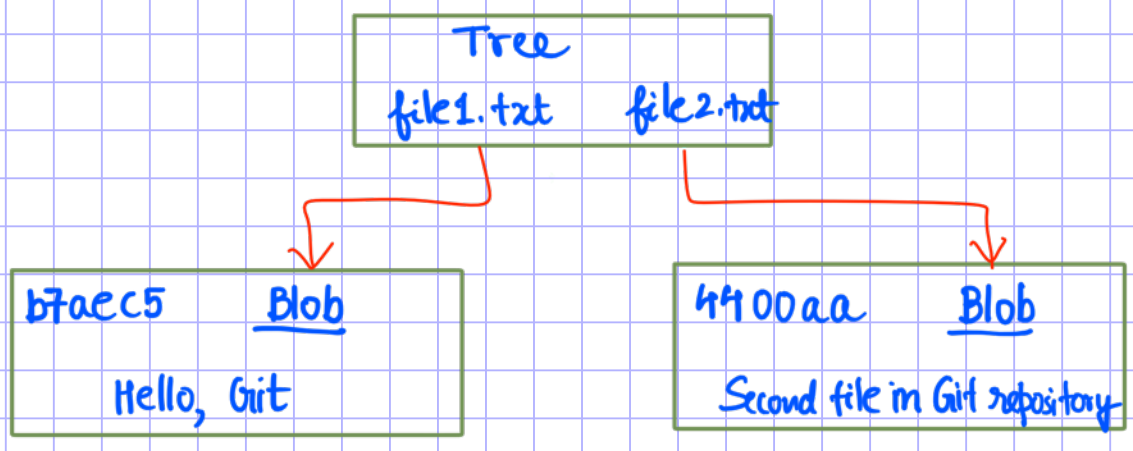

hemakshipandey@Hemakshis-MacBook-Pro:~/Desktop/first-project
..first-project (-zsh) ⌘1
~ (-zsh) ⌘2
~/Desktop (-zsh)
→ first-project git:(master) ✗ ls -la
total 16
drwxr-xr-x@ 4 hemakshipandey staff 128 Aug 10 20:23 .
drwx------@ 8 hemakshipandey staff 256 Aug 11 13:27 ..
-rw-r--r--@ 1 hemakshipandey staff 6148 Aug 10 20:24 .DS_Store
drwxr-xr-x@ 10 hemakshipandey staff 320 Aug 10 20:23 .git
→ first-project git:(master) ✗ find .git/objects -type f
.git/objects/.DS_Store
.git/objects/44/00aae52a27341314f423095846b1f215a7cf08
.git/objects/b7/aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
→ first-project git:(master) ✗ git mktree --help
[1] + 1308 suspended git mktree --help
→ first-project git:(master) ✗ cat ../temp-tree.txt
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08 file2.txt
→ first-project git:(master) ✗ cat ../temp-tree.txt | git mktree
3b95df0ac6365c72e9b0ff6c449645c87e6e1159
→ first-project git:(master) ✗
→ SHA1 hash of the new tree Git Object


objects
Favourites
AirDrop
Recents
Applications
Downloads
iCloud
info
pack
b7
44
3b

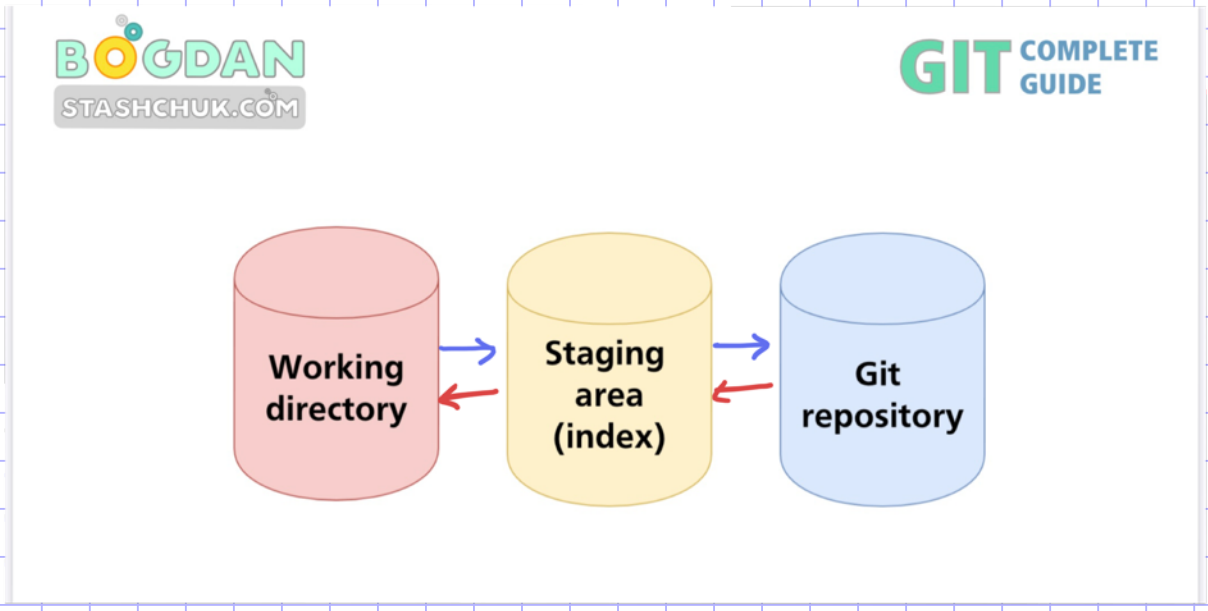
Examining Tree Object

```
→  Desktop cat temp-tree.txt
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
→  Desktop rm temp-tree.txt
→  Desktop ls -la
total 4088
drwx------@  9 hemakshipandey  staff     288 Aug 11 16:34 .
drwxr-xr-x+ 20 hemakshipandey  staff     640 Aug 11 16:34 ..
-rw-r--r--@  1 hemakshipandey  staff    6148 Aug 11 16:22 .DS_Store
-rw-r--r--   1 hemakshipandey  staff       0 Jul 26 19:51 .localized
lrwxr-xr-x@  1 hemakshipandey  staff      29 Aug 10 19:26 Relocated Items.nosync
 -> /Users/Shared/Relocated Items
-rw-r--r--@  1 hemakshipandey  staff  455361 Aug 11 16:14 Screenshot 2021-08-11
at 4.14.15 PM.png
-rw-r--r--@  1 hemakshipandey  staff  692308 Aug 11 16:17 Screenshot 2021-08-11
at 4.17.24 PM.png
-rw-r--r--@  1 hemakshipandey  staff  928533 Aug 11 16:22 Screenshot 2021-08-11
at 4.22.18 PM.png
drwxr-xr-x@  4 hemakshipandey  staff     128 Aug 11 16:19 first-project
→  Desktop
```

```
→  first-project git:(master) ✗ find .git/objects -type f
.git/objects/3b/95df0ac6365c72e9b0ff6c449645c87e6e1159
.git/objects/.DS_Store
.git/objects/44/00aae52a27341314f423095846b1f215a7cf08
.git/objects/b7/aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
→  first-project git:(master) ✗ git cat-file -p 3b95d
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
→  first-project git:(master) ✗ git cat-file -s 3b95d
74
→  first-project git:(master) ✗ git cat-file -t 3b95d
tree
→  first-project git:(master) ✗
```



Working directory, Staging area, Git repository



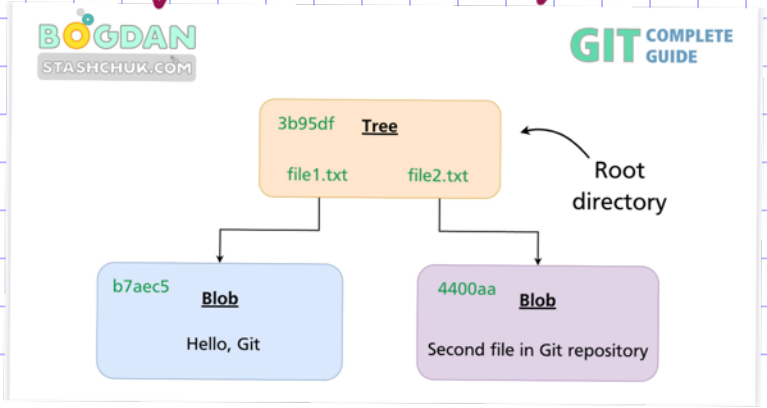
→ There are actually three main areas where our files and folders may live in git. They are :- working directory, staging area (index) and Git repository.

→ Staging area comes in between working directory and Git repository.

→ Staging area is usually called index and the staging area is actually responsible for preparing files to be inserted into git repository and in opposite way it prepares file taken from the repository, to be put into working directory.

Note :- putting file into staging area or index is mandatory step in all git operations.

Overview of current files distribution



→ These three object now are located in Git repository.

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

git ls-files -s

List files in the staging area

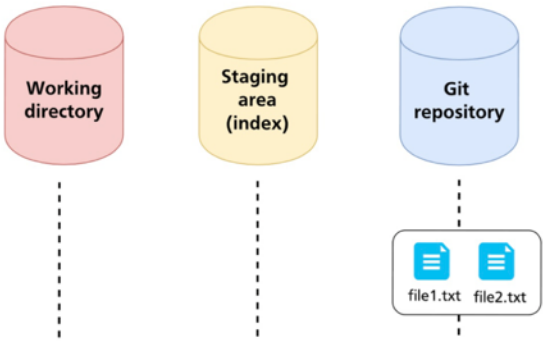
```
→  first-project git:(master) ✗ git ls-files
→  first-project git:(master) ✗
```

← staging area (index) is empty now

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Files in the "first-project"

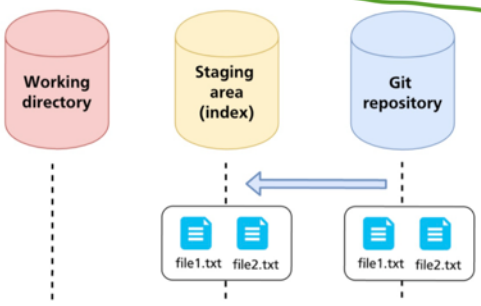


Git read-tree

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

git read-tree <hash> → SHA1 hash of tree object



We use this command to retrieve files from git repository and put them into a staging area.

```
→  first-project git:(master) ✗ find .git/objects -type f
.git/objects/3b/95df0ac6365c72e9b0ff6c449645c87e6e1159
.git/objects/.DS_Store
.git/objects/44/00aae52a27341314f423095846b1f215a7cf08
.git/objects/b7/aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
→  first-project git:(master) ✗ git cat-file 3b95 -t
tree
→  first-project git:(master) ✗ git read-tree 3b95
→  first-project git:(master) ✗
```

Looking at files in ".git/objects" we will not be able to determine types of the Git objects stores in those files.

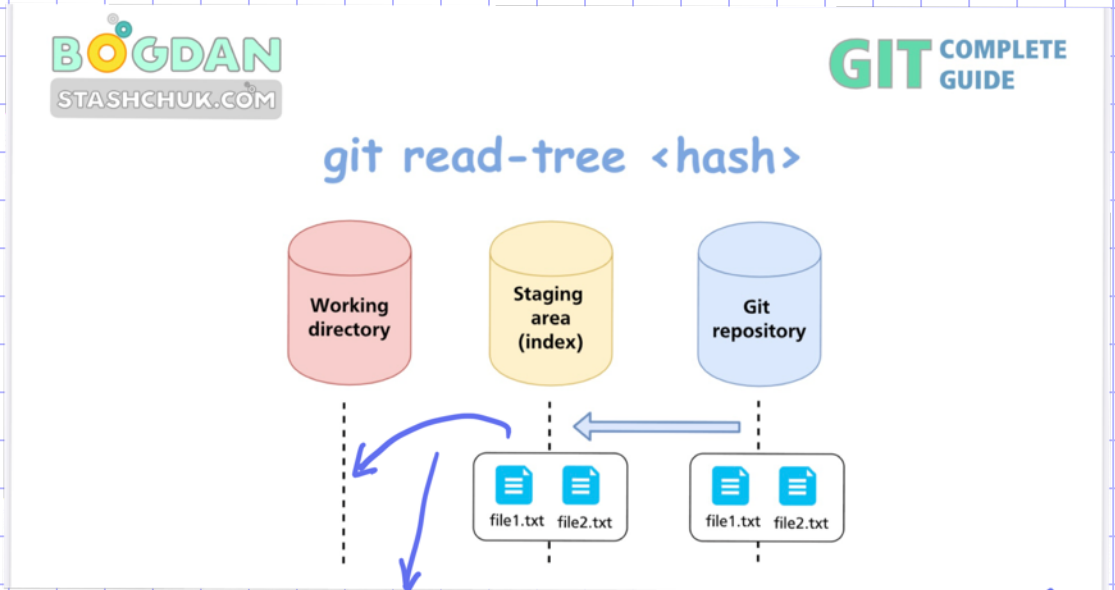
Read files in the staging area using git ls-files

```
→  first-project git:(master) ✗ git read-tree 3b95
→  first-project git:(master) ✗ git ls-files
file1.txt
file2.txt
→  first-project git:(master) ✗ git ls-files -s
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
→  first-project git:(master) ✗
```

{ staging area }

permission — SHA1 hash — same files no changes — names

Git checkout-index

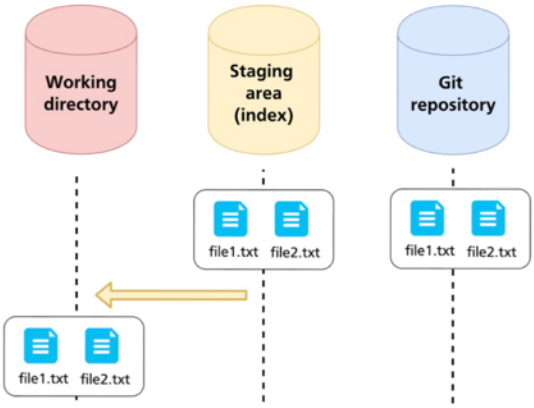


now we need to take these files and put them into our working directory. and in working directory, there are no files except .git folder.

BOGDAN
STASHCHUK.COM

GIT COMPLETE GUIDE

git checkout-index -a → put the files from staging area into working directory.

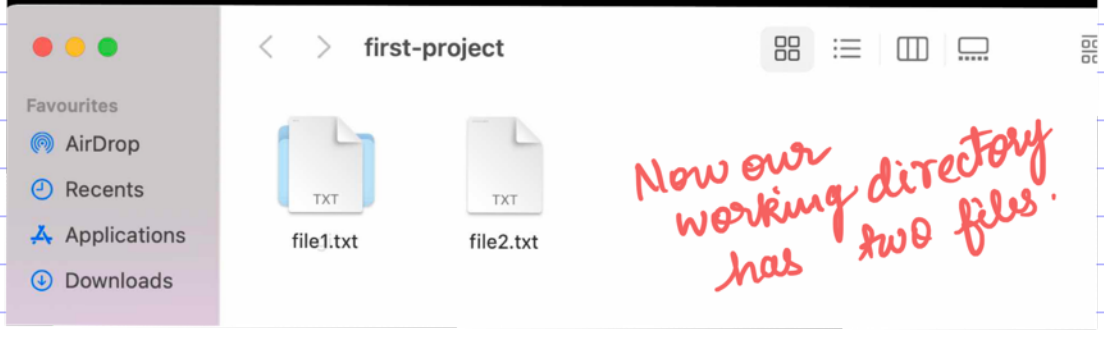



```
→  first-project git:(master) ✗ git ls-files -s
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
→  first-project git:(master) ✗ git checkout-index -a
→  first-project git:(master) ✗
```

```
Last login: Thu Aug 12 12:59:58 on console
→  ~ cd Desktop
→  Desktop cd first-project
→  first-project git:(master) ✗ ls
→  first-project git:(master) ✗ ls
file1.txt file2.txt
→  first-project git:(master) ✗ ls -l
total 16
-rw-r--r--  1 hemakshipandey  staff  11 Aug 12 13:18 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 12 13:18 file2.txt
→  first-project git:(master) ✗ cat file1.txt
Hello, Git
→  first-project git:(master) ✗ cat file2.txt
Second file in Git repository
→  first-project git:(master) ✗
```

initialy working directory is empty

after git checkout-index -a



Now our working directory has two files.

How many folder could be created for objects in ".\git\objects"? → Ans → 16 x 16 = 256

git status command → displays the state of the working directory and the staging area.

```
→  first-project git:(master) ✗ ls
file1.txt file2.txt
→  first-project git:(master) ✗ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

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

→  first-project git:(master) ✗
```

Section 5 : Basic Git Operations

What is commit?

Commit is another object type of git. It has the same structure as blob and tree.

what are contents of git commit?

Each git commit has following information inside:-

→ Each commit has its own SHA1 hash

What is the purpose of commit? Why do we need special object type?

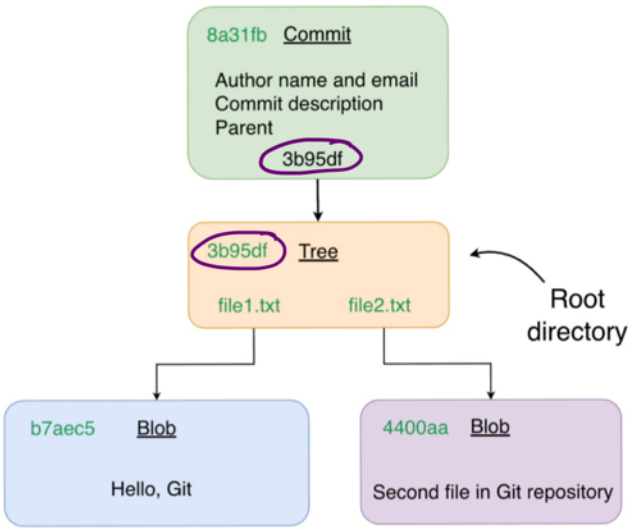
Commit actually allow us to store in git database different versions of our projects or different snapshots. After that we are able to pretty easily and fast move to any version of our project by checking out specific commit

How commit stores current "version" or "snapshot" of the project?

→ Commit is just "wrapper" for the tree object and contains pointer to specific tree. And, of course, you may have multiple trees in you git repository, multiple pointer and multiple commits to different trees.

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE



→ By moving to different commits (checking out), you are able to "travel" between different "versions" of the project.

→ check out means taking files from Git repository and put them into your working directory.

Configure Git authorname and email

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Set Git Author name and email

```
git config --global user.name <Name>
git config --global user.email <Email>
```

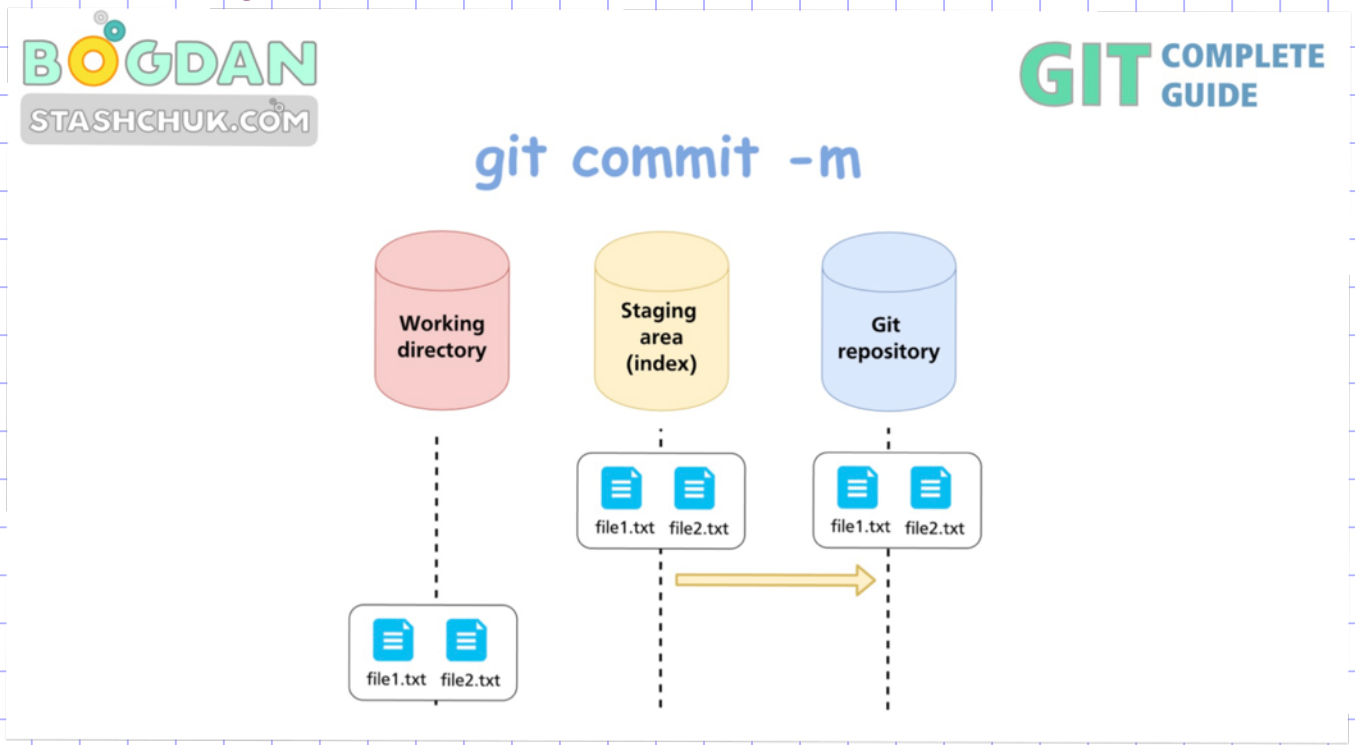
} → used for setting username and email

```
git config --list
```

} → used for reading configuration of git.

```
Second file in Git repository
credential.helper=osxkeychain
user.name=hemakshi1234
user.email=mahe173fas@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
(END)
credential.helper=osxkeychain
user.name=hemakshi1234
user.email=mahe173fas@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
```

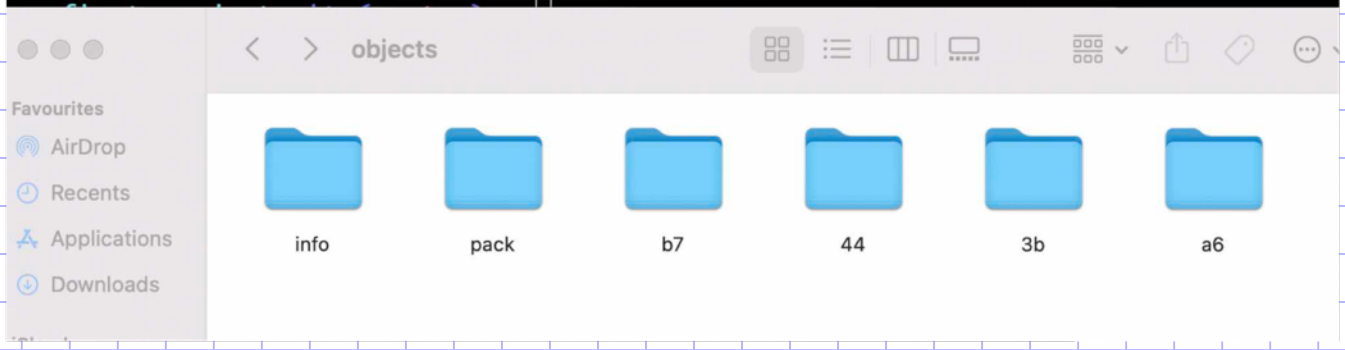
Creating first Commit



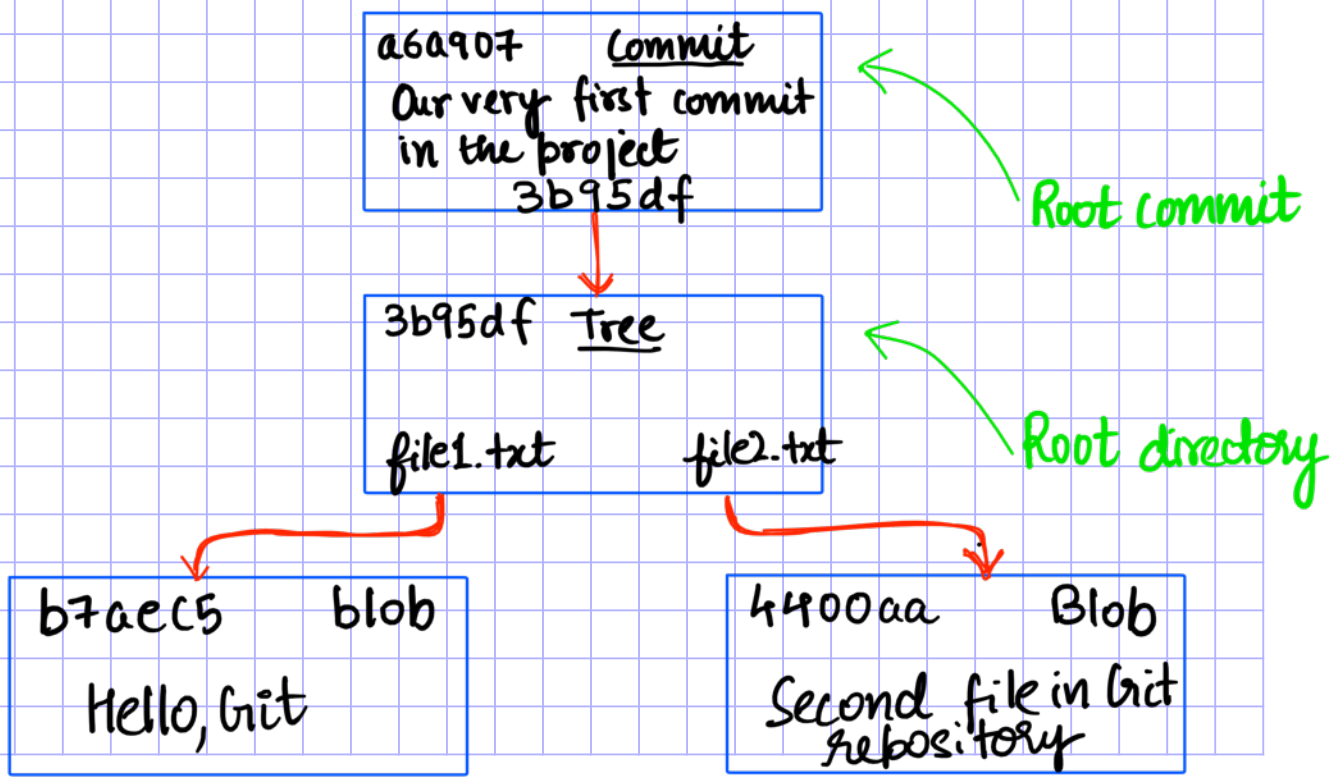
```
→  first-project git:(master) ✗ git commit -m "Our very first commit in the proj
ect"
[master (root-commit) a6a9072] Our very first commit in the project
 2 files changed, 2 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
```

Exploring Commit Object

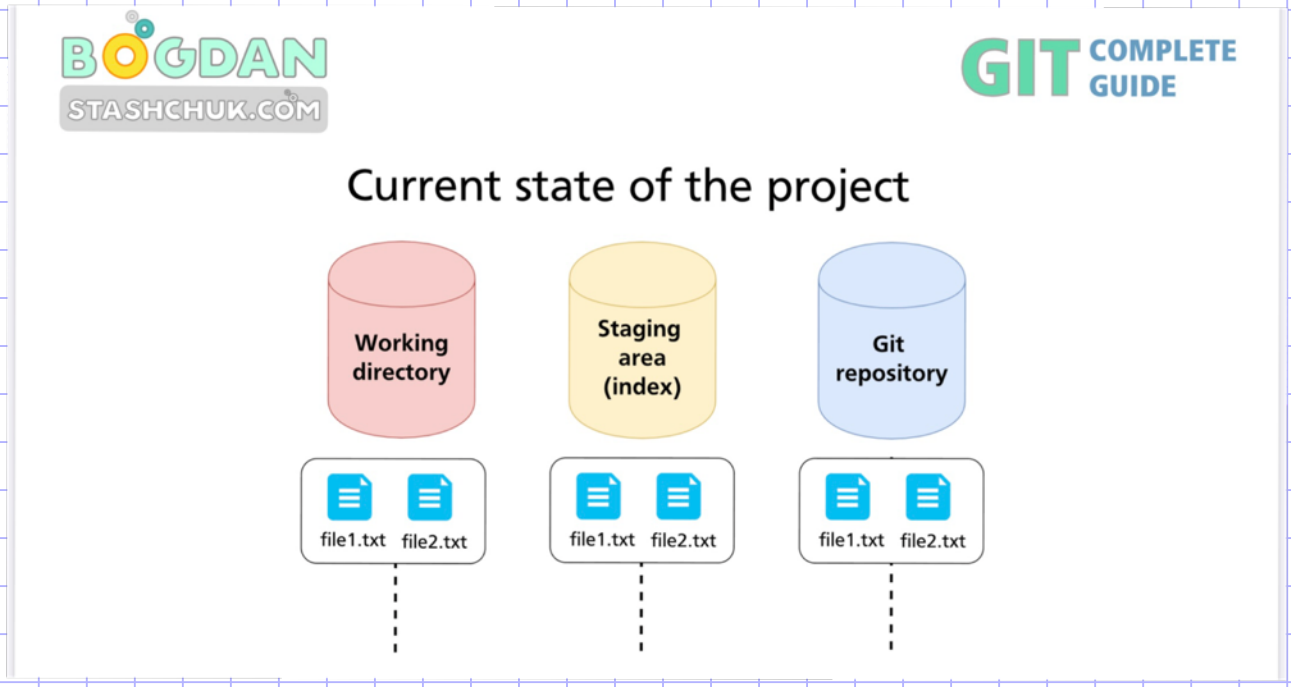
```
→  first-project git:(master) ✗ git cat-file -p a6a9
tree 3b95df0ac6365c72e9b0ff6c449645c87e6e1159
author hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530

Our very first commit in the project
→  first-project git:(master) ✗ git cat-file -s a6a9
207
→  first-project git:(master) ✗ git cat-file -t a6a9
commit
```



Current project state Overview



```
➜  first-project git:(master) ✗ ls
file1.txt file2.txt
➜  first-project git:(master) ✗ ls -l
total 16
-rw-r--r--  1 hemakshipandey  staff  11 Aug 12 13:18 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 12 13:18 file2.txt
➜  first-project git:(master) ✗ git ls-files -s
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
➜  first-project git:(master) ✗ find .git/objects -type f
.git/objects/3b/95df0ac6365c72e9b0ff6c449645c87e6e1159
.git/objects/.DS_Store
.git/objects/44/00aae52a27341314f423095846b1f215a7cf08
.git/objects/b7/aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
.git/objects/a6/a90726f0c078e392724f40e1f295539c9c16b3
```



Basic Git commands

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Basic Git commands

git status	Current state of Git repository
git add	Add files to staging area
git commit	Write changes to Git repository
git log	History of changes (commits)
git checkout	Checkout commit or branch

→ Other git commands that are used for interaction with remote Git repository.

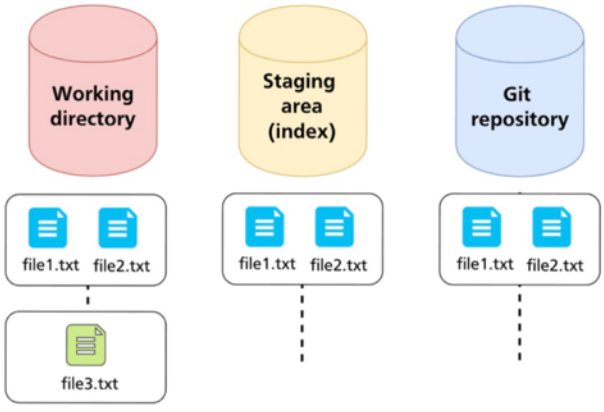
1. git pull
2. git push
3. git fetch
4. git clone

Adding new file to working directory

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

Create new file in the working directory



```
→  first-project git:(master) git status
On branch master
nothing to commit, working tree clean
→  first-project git:(master) git log

[2]  + 2108 suspended  git log
→  first-project git:(master) ls
file1.txt file2.txt
→  first-project git:(master) nano file3.txt
```

at the moment there are two files

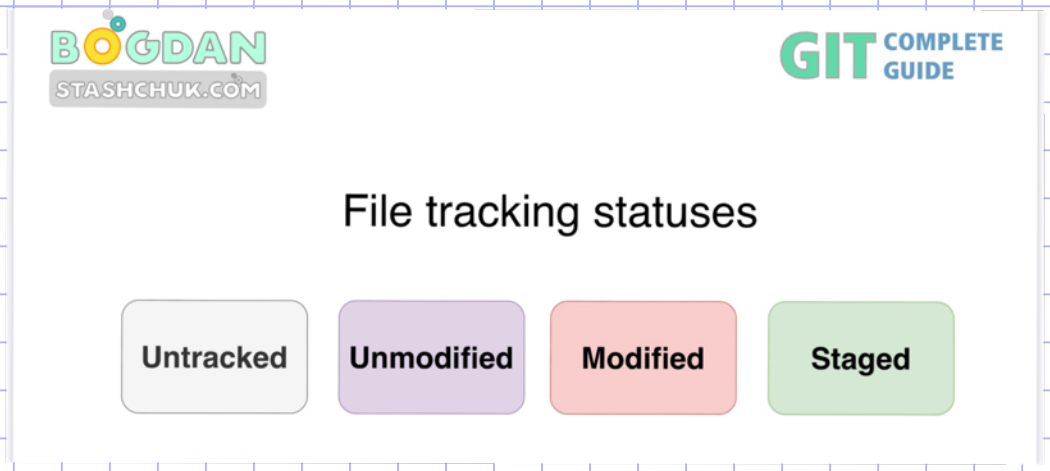
```
→  first-project git:(master) nano file3.txt
→  first-project git:(master) ✗ ls -l
total 24
-rw-r--r--  1 hemakshipandey  staff  11 Aug 12 13:18 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 12 13:18 file2.txt
-rw-r--r--  1 hemakshipandey  staff  41 Aug 12 20:35 file3.txt
→  first-project git:(master) ✗ git ls-file -s
git: 'ls-file' is not a git command. See 'git --help'.

The most similar command is
        ls-files
→  first-project git:(master) ✗ git ls-files -s
100644 fc22720f2257083bd71bfadc42654cd8b481b6eb 0       .DS_Store
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
→  first-project git:(master) ✗ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt

nothing added to commit but untracked files present (use "git add" to track)
→  first-project git:(master) ✗
```

→ creating third file

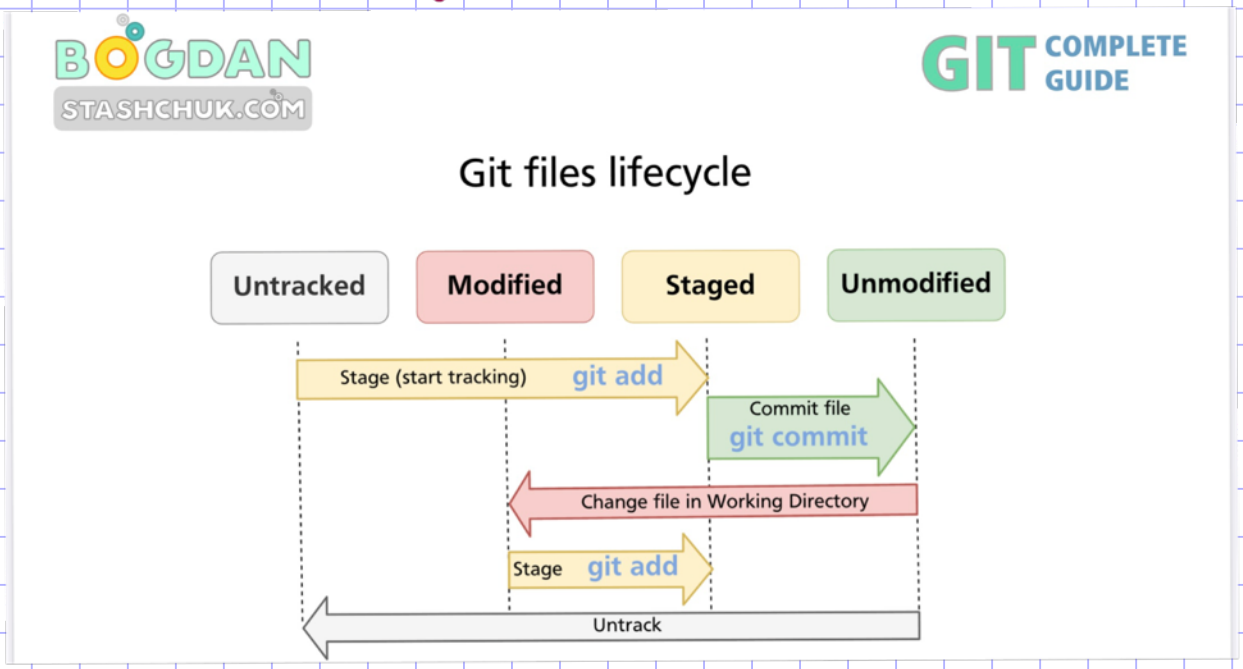
now we can see one more file in our working directory

file3.txt is absent now in the staging area.

→ Every file in Git may have one of 4 tracking statuses:-



Git files Lifecycle



Stage file

```
➜  first-project git:(master) ✗ git add file3.txt
➜  first-project git:(master) ✗ git ls-files
.DS_Store
file1.txt
file2.txt
file3.txt
➜  first-project git:(master) ✗ git ls-files -s
100644 fc22720f2257083bd71bfadc42654cd8b481b6eb 0       .DS_Store
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
100644 289960fd4f172ef29a9bf30f9c49c1725ca253dc 0       file3.txt
➜  first-project git:(master) ✗
```

} listing files in staging area

Unstage file using git rm

```
→  first-project git:(master) ✗ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file3.txt

→  first-project git:(master) ✗ git rm --cached file3.txt
rm 'file3.txt'
→  first-project git:(master) ✗ git ls-files -s
100644 fc22720f2257083bd71bfadc42654cd8b481b6eb 0       .DS_Store
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
→  first-project git:(master) ✗ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt

nothing added to commit but untracked files present (use "git add" to track)
→  first-project git:(master) ✗
```

→ unstage specific file

→ but this file is still in working directory

Commit Changes

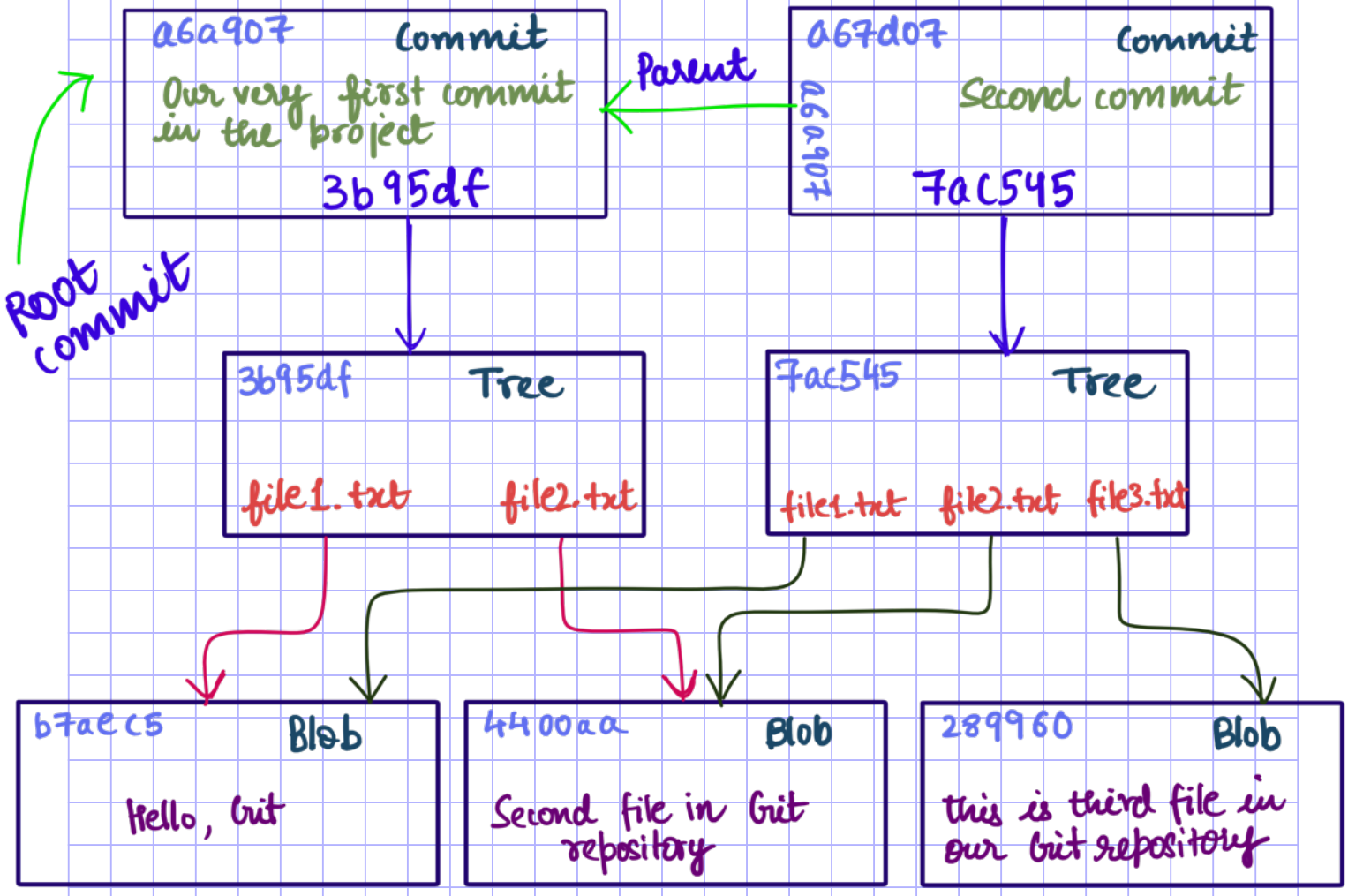
```
→  first-project git:(master) ✗ git commit -m "Second commit"
[master a67d07d] Second commit
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt
→  first-project git:(master) cat file3.txt
This is third file in our Git repository
→  first-project git:(master)
```

→ git should create new object for commit.
→ Create new blob object for file3.txt.
→ Create new tree object.

Exploring changes in Git respository

```
➜  first-project git:(master) git status
On branch master
nothing to commit, working tree clean
➜  first-project git:(master) git log

[1]  + 1018 suspended  git log
➜  first-project git:(master) git cat-file -p a67d07
tree 7ac5455f6f836212b6935b91dde39a55c4f5c4a4
parent d0fa6cfc6f9bdef2c2ac54224f9eb5c09215814f
author hemakshi1234 <mahe173fas@gmail.com> 1628846061 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628846061 +0530

Second commit
➜  first-project git:(master) git cat-file -p d0fa6c
tree 6949aa9ffa787b0dd5629e991cc64a67106bf0aa
parent a6a90726f0c078e392724f40e1f295539c9c16b3
author hemakshi1234 <mahe173fas@gmail.com> 1628780463 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628780463 +0530

first commit
➜  first-project git:(master) git cat-file -p a6a907
tree 3b95df0ac6365c72e9b0ff6c449645c87e6e1159
author hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530

Our very first commit in the project
➜  first-project git:(master) git cat-file -p 7ac545
100644 blob fc22720f2257083bd71bfadc42654cd8b481b6eb    .DS_Store
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
100644 blob 289960fd4f172ef29a9bf30f9c49c1725ca253dc    file3.txt
➜  first-project git:(master)
```

Current diagram of Git repository

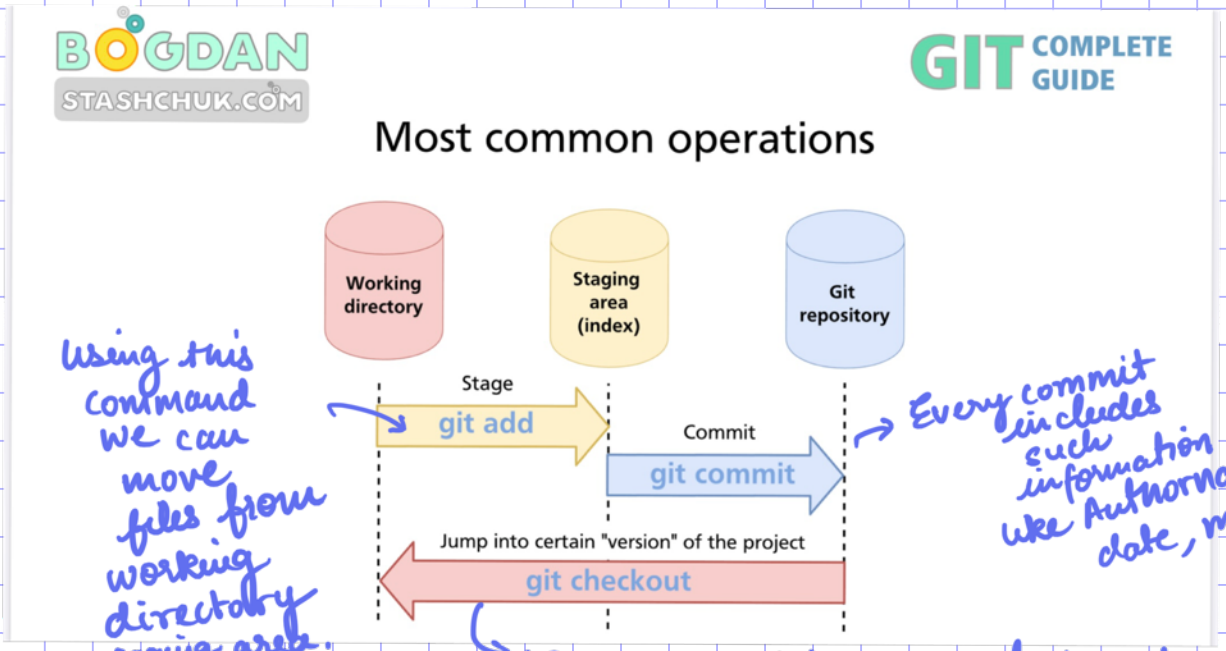


Time →

(ignore d0fa6c → first commit)
in previous page exploring changes in git repository

Section 6: Git branches and HEAD

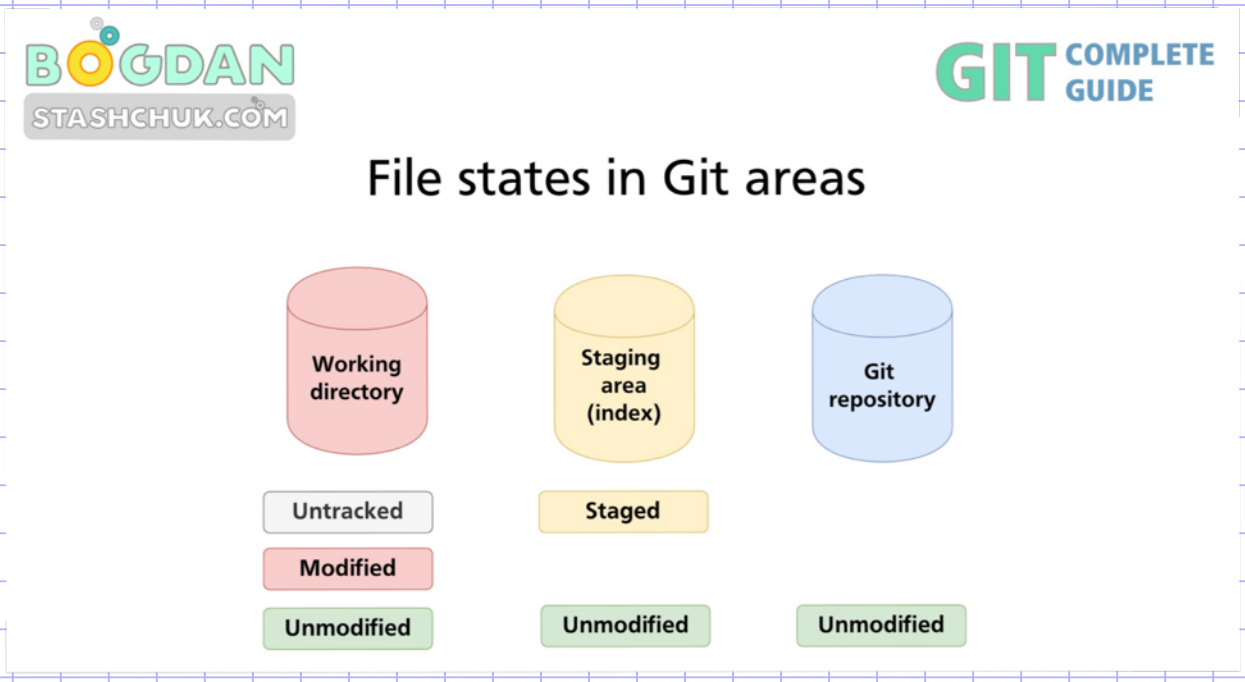
Most common Git Operations



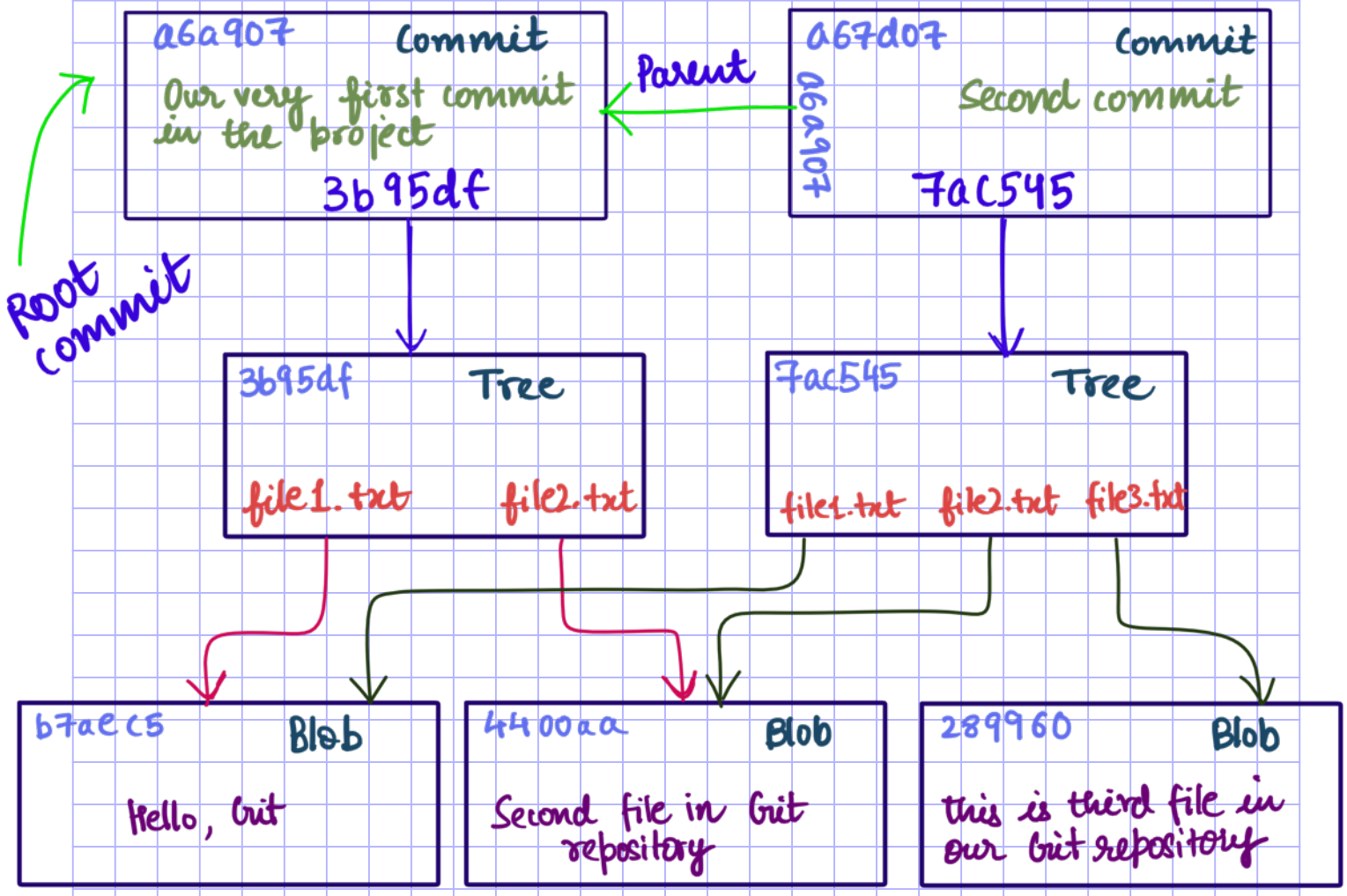
Using this command we can move files from working directory to staging area. (untracked files or modified files)

Every commit includes such information like Author name, date, message.

you can use this command in order to retrieve specific version of your project from the Git Repository



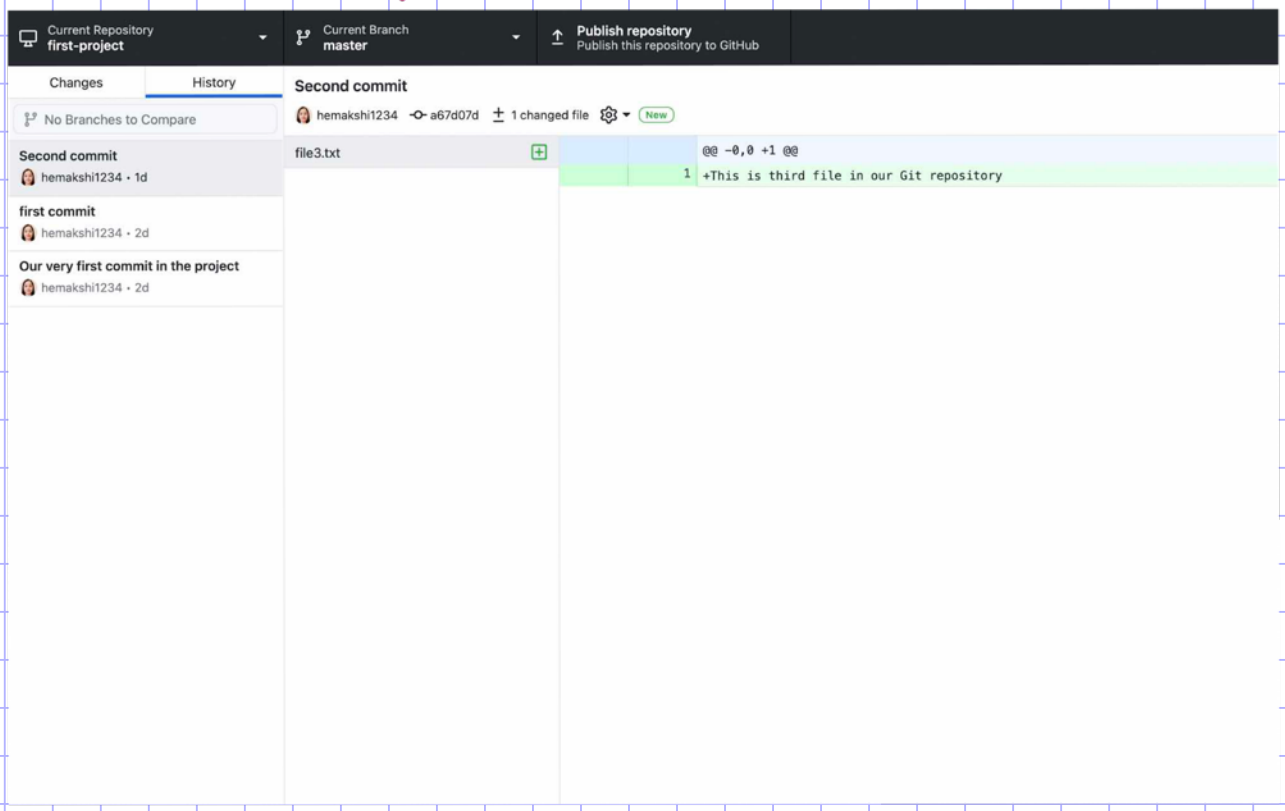
Overview of current project state

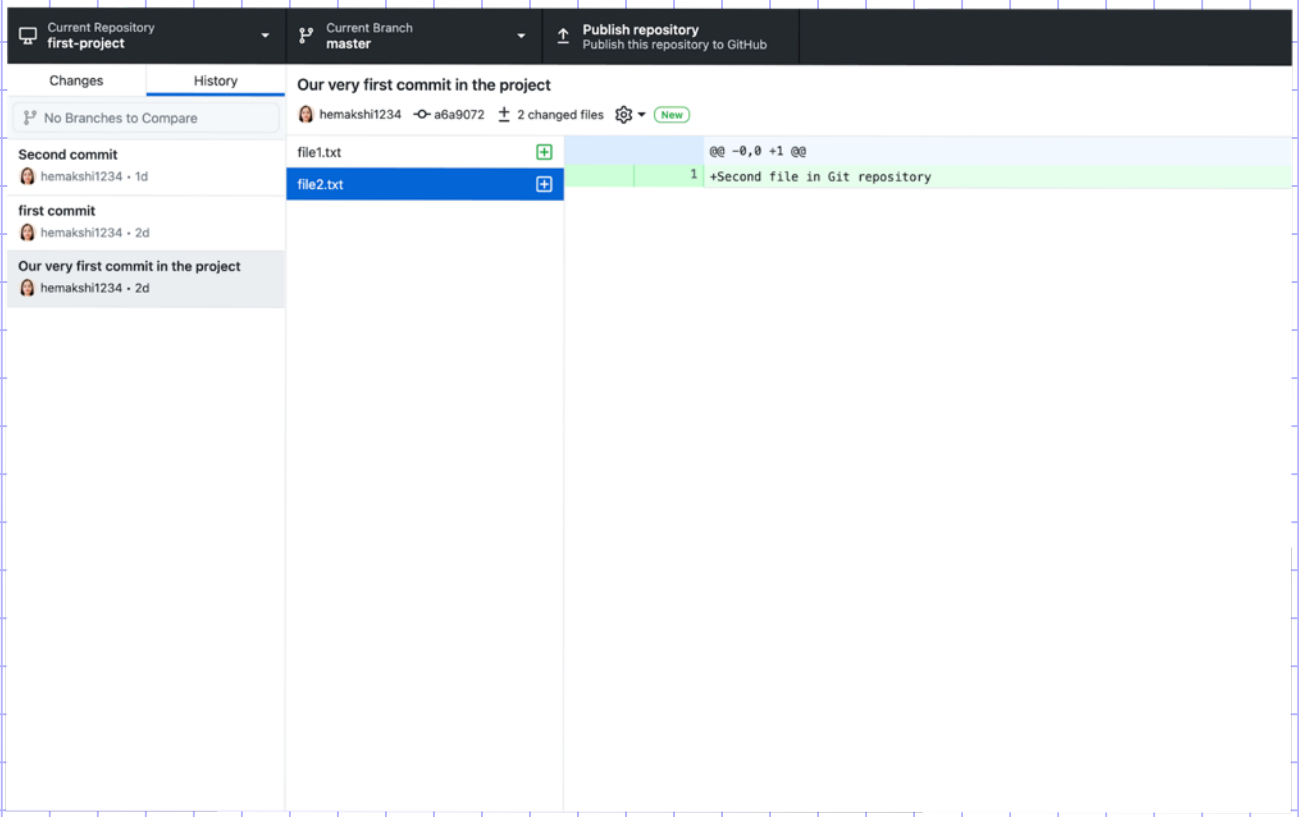


Time →

(ignore d0fa6c → first commit)
in previous page exploring changes in git repository

Github Desktop Overview





What is branch in Git

BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

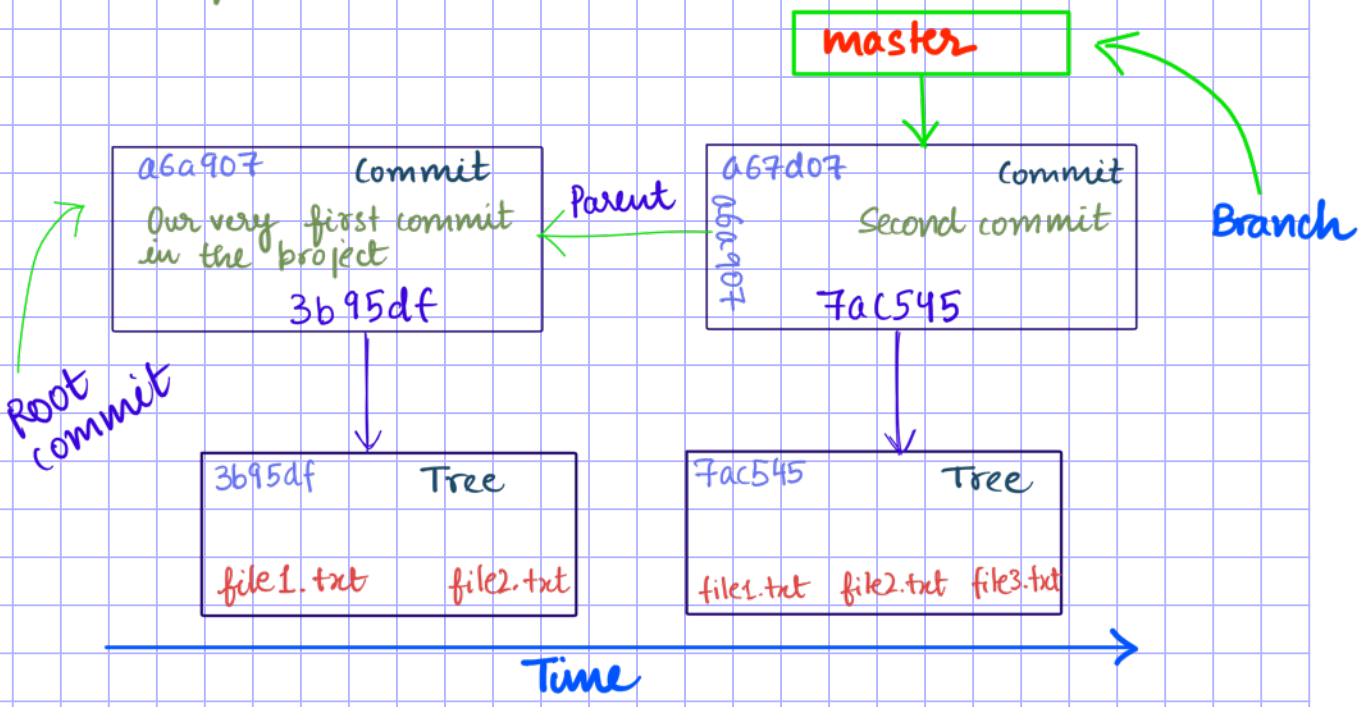
Branch is just text reference to the commit

1. Default branch is **master**
2. Multiple branches can exist in the same repository
3. Pointers for all branches are located in **.git/refs/heads** folder
4. Current branch tracks new commits
5. Branch pointer moves automatically after every new commit
6. Change branch **git checkout <branch>**

master new-branch DEV-1512

↳ branch name do not have spaces

Which commit current default "master" branch points now to?



Ans → master branch now points to the second commit. It is the last commit in the chain of commits

```
Last login: Sun Aug 15 11:01:55 on console
→  ~ cd Desktop/first-project
→  first-project git:(master) cd .git/refs/heads
→  heads git:(master) ls -l
total 8
-rw-r--r--  1 hemakshipandey  staff  41 Aug 13 14:44 master
→  heads git:(master) cat master
a67d07dc0ea4af75344d161b22c4c1455093a8f1
→  heads git:(master)
```

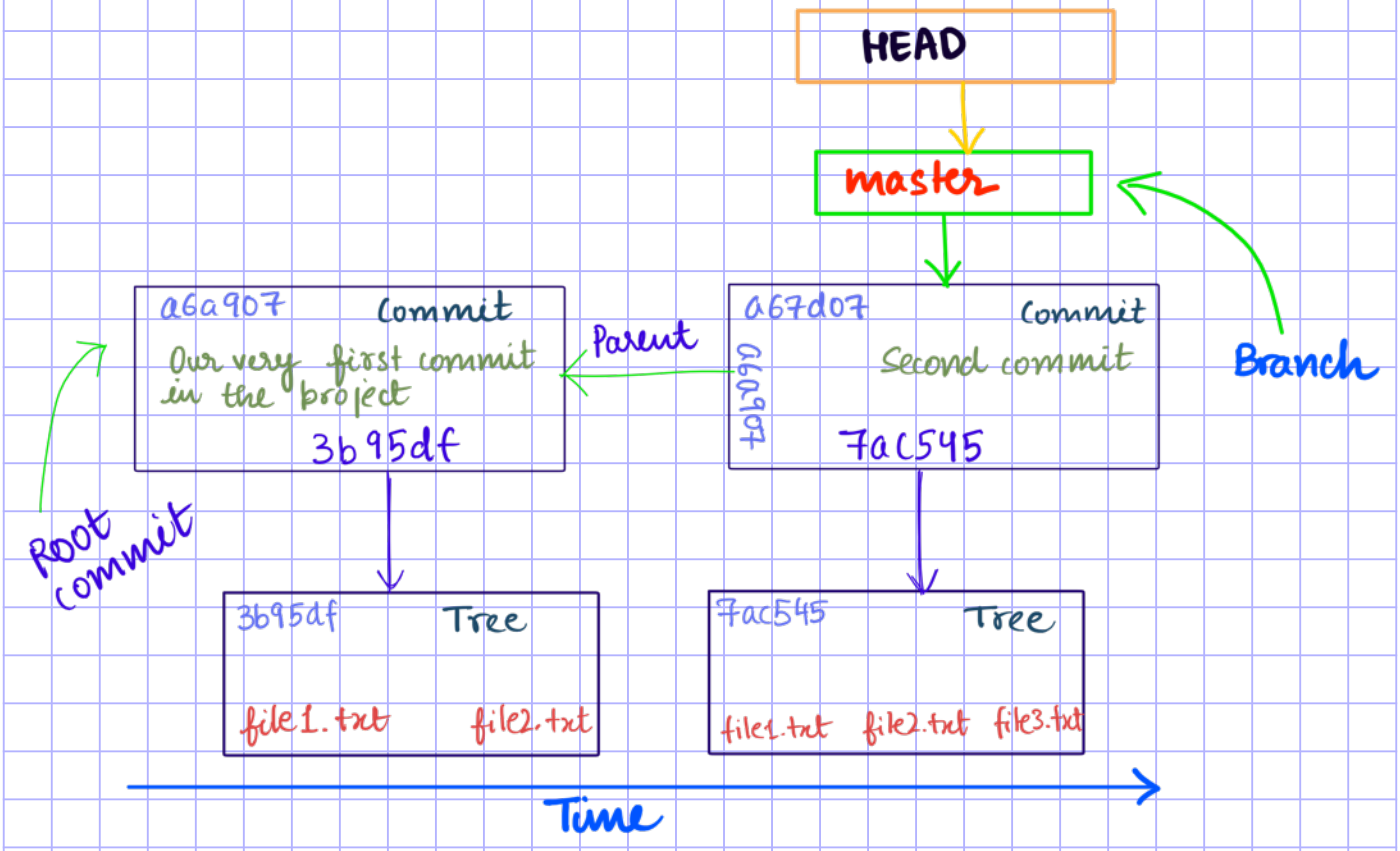
SHA1 hash of the last commit in the branch.

What is HEAD in Git

→ Your working directory may contain only single "version" of your project at any moment of time.

Q How Git know which branch is current?

→ HEAD is just pointer to specific branch or specific commit. There may be multiple branches or commit but HEAD is always one. It is a pointer to current snapshot of your project.



BOGDAN STASHCHUK.COM

GIT COMPLETE GUIDE

HEAD is reference to the currently checked-out branch or commit

HEAD

1. HEAD is locally significant
2. Pointer is located in the **.git/HEAD** file
3. Default pointer is **ref: refs/heads/master**
4. Change reference to specific branch **git checkout <branch>**
5. Change reference to specific commit **git checkout <sha1>**

```
➜  heads git:(master) cd ../..
➜  .git git:(master) ls -la
total 64
drwxr-xr-x@ 13 hemakshipandey  staff   416 Aug 13 14:44 .
drwxr-xr-x@  7 hemakshipandey  staff   224 Aug 13 14:44 ..
-rw-r--r--@  1 hemakshipandey  staff  8196 Aug 15 12:07 .DS_Store
-rw-r--r--   1 hemakshipandey  staff    14 Aug 13 14:44 COMMIT_EDITMSG
-rw-r--r--   1 hemakshipandey  staff    23 Aug  5 18:09 HEAD
-rw-r--r--   1 hemakshipandey  staff   137 Aug  5 18:09 config
-rw-r--r--   1 hemakshipandey  staff    73 Aug  5 18:09 description
drwxr-xr-x@ 15 hemakshipandey  staff   480 Aug 10 19:48 hooks
-rw-r--r--   1 hemakshipandey  staff   353 Aug 13 14:44 index
drwxr-xr-x@  3 hemakshipandey  staff    96 Aug 10 19:48 info
drwxr-xr-x@  4 hemakshipandey  staff   128 Aug 13 14:44 logs
drwxr-xr-x@ 14 hemakshipandey  staff   448 Aug 13 14:44 objects
drwxr-xr-x@  5 hemakshipandey  staff   160 Aug 12 18:32 refs
➜  .git git:(master) cat HEAD
ref: refs/heads/master
➜  .git git:(master) cat refs/heads/master
a67d07dc0ea4af75344d161b22c4c1455093a8f1
➜  .git git:(master)
```

SHA1 hash to second commit (last commit)

```
commit a67d07dc0ea4af75344d161b22c4c1455093a8f1 (HEAD -> master)
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Fri Aug 13 14:44:21 2021 +0530

    Second commit

commit d0fa6cfc6f9bdef2c2ac54224f9eb5c09215814f
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 20:31:03 2021 +0530

    first commit

commit a6a90726f0c078e392724f40e1f295539c9c16b3
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 18:32:44 2021 +0530

    Our very first commit in the project
(END)
```

Third Commit

```
➜  .git git:(master) clear
➜  .git git:(master) cd ..
➜  first-project git:(master) ls -l
total 24
-rw-r--r--  1 hemakshipandey  staff  11 Aug 12 13:18 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 12 13:18 file2.txt
Third commit
-rw-r--r--  1 hemakshipandey  staff  41 Aug 12 20:35 file3.txt
➜  first-project git:(master) rm file1.txt
➜  first-project git:(master) ✗ rm file2.txt
➜  first-project git:(master) ✗ rm file3.txt
➜  first-project git:(master) ✗ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    file1.txt
        deleted:    file2.txt
        deleted:    file3.txt

no changes added to commit (use "git add" and/or "git commit -a")
➜  first-project git:(master) ✗ git add .
➜  first-project git:(master) ✗ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    file1.txt
        deleted:    file2.txt
        deleted:    file3.txt

➜  first-project git:(master) ✗ git commit
[master f76c7fc] Third commit
 3 files changed, 3 deletions(-)
 delete mode 100644 file1.txt
 delete mode 100644 file2.txt
 delete mode 100644 file3.txt
➜  first-project git:(master)
```

```
Third commit
# 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:
#       deleted:    file1.txt
#       deleted:    file2.txt
#       deleted:    file3.txt
#
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

→ git commit
↳ a text editor will pop up.

① insert mode
② Command mode (default mode)

→ to make change in file
press 'I' key for insert mode

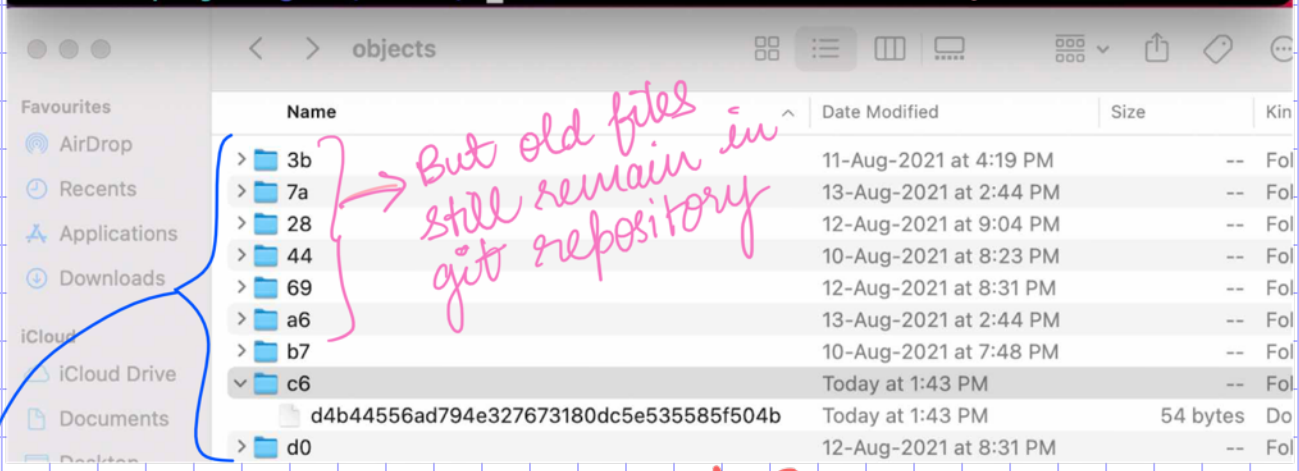
→ press ESC in order to exit from insert mode in VIM editor.

→ In 'command mode' type ":wq" to save changes and exit from VIM editor

Git repository changes after third commit

```
commit f76c7fcf9fdaa7a694bb777abafe06980ce4c200 (HEAD -> master)
Author: Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com>
Date:   Sun Aug 15 13:43:30 2021 +0530

    Third commit

commit a67d07dc0ea4af75344d161b22c4c1455093a8f1
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Fri Aug 13 14:44:21 2021 +0530

    Second commit

commit d0fa6cfc6f9bdef2c2ac54224f9eb5c09215814f
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 20:31:03 2021 +0530

    first commit

commit a6a90726f0c078e392724f40e1f295539c9c16b3
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 18:32:44 2021 +0530

    Our very first commit in the project
(END)
```

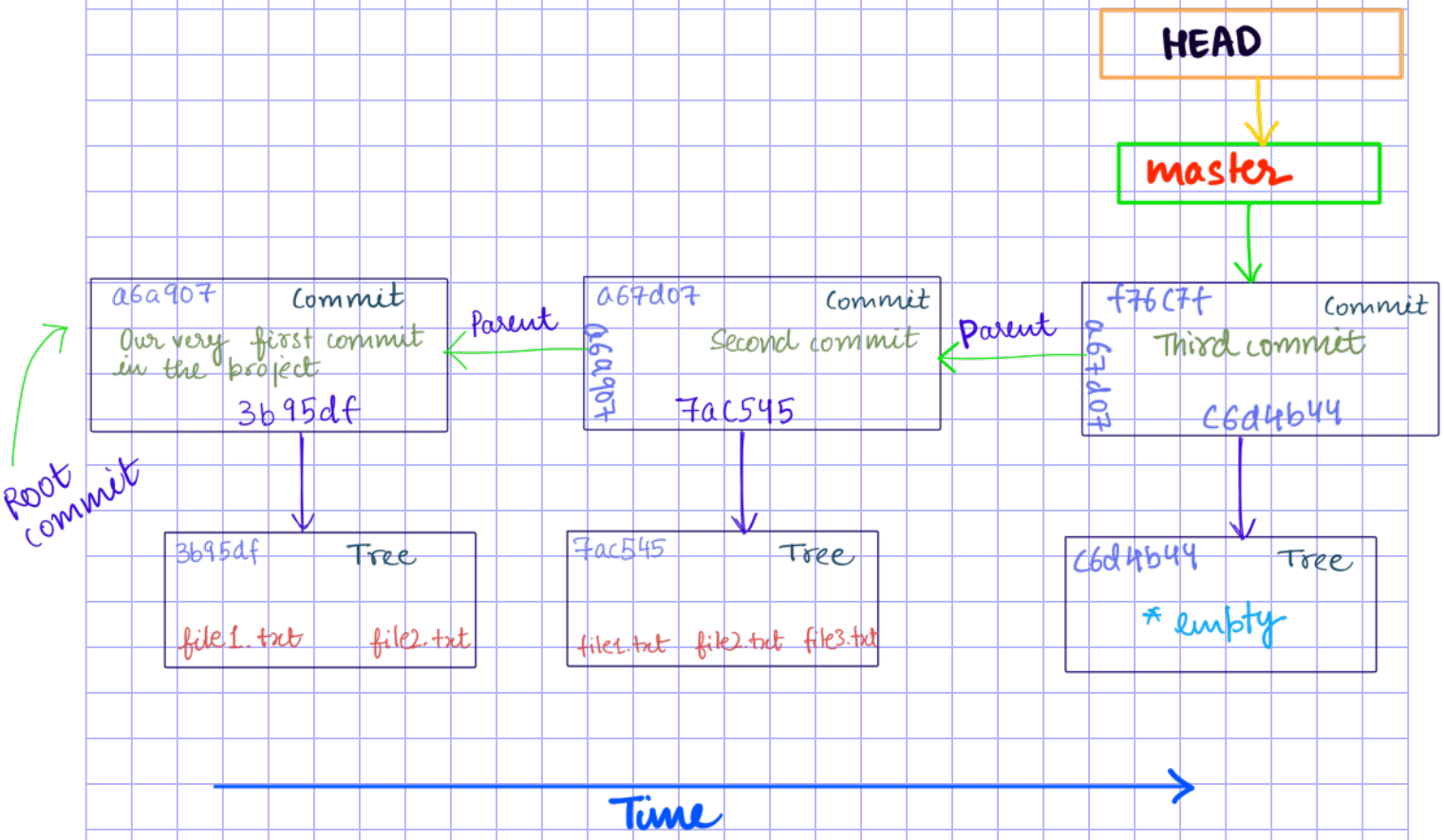
```
[3]  + 2176 suspended  git log
→  first-project git:(master) cat .git/HEAD
ref: refs/heads/master
→  first-project git:(master) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200 → SHA1 hash of new commit
→  first-project git:(master) git cat-file -p f76c7f → content of third commit
tree c6d4b44556ad794e327673180dc5e535585f504b → pointer to new tree object
parent a67d07dc0ea4af75344d161b22c4c1455093a8f1 → pointer to parent commit
author Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 16290152
10 +0530
committer Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 16290
15210 +0530

Third commit → commit message
→  first-project git:(master) git cat-file -p c6d4b4
100644 blob fc22720f2257083bd71bfadc42654cd8b481b6eb    .DS_Store
→  first-project git:(master) git cat-file -t c6d4b4
tree
→  first-project git:(master) git cat-file -s c6d4b4
37
→  first-project git:(master) git ls-files -s → staging area is now empty
100644 fc22720f2257083bd71bfadc42654cd8b481b6eb 0       .DS_Store
→  first-project git:(master) ls → working directory is empty
→  first-project git:(master)
```



Why this tree object is empty?

→ We have deleted all files from working directory. That's why this tree doesn't have links to any blobs or other trees

→ Git didn't delete any of the previously created files or folders in git repository and that means that git stores entire history, all previous trees and blobs are still in git repository and we will be able to pretty easily and fast go to previous commits and checkout previous commits.



Checkout specific commit

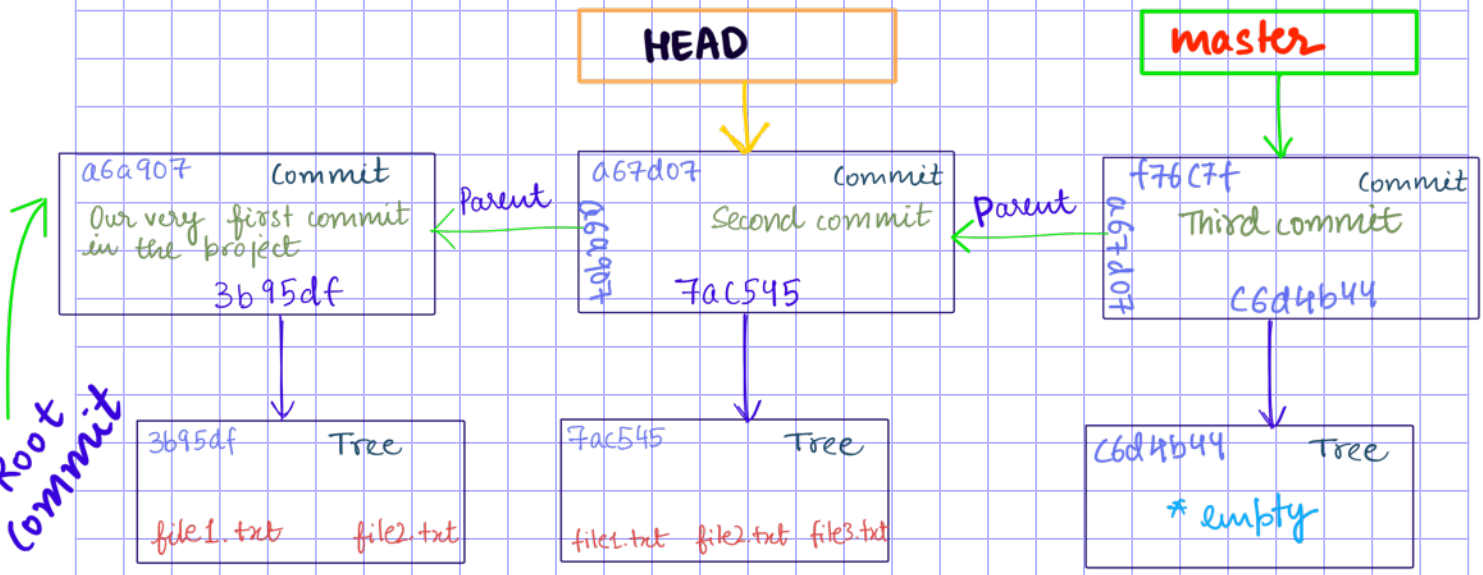
```
➜  first-project git:(master) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
➜  first-project git:(master) git cat-file -p f76c7f
tree c6d4b44556ad794e327673180dc5e535585f504b
parent a67d07dc0ea4af75344d161b22c4c1455093a8f1
author Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629015210 +0530
committer Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629015210 +0530

Third commit
➜  first-project git:(master) git checkout a67d07
Note: switching to 'a67d07'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at a67d07d Second commit
➜  first-project git:(a67d07d) cat .git/HEAD
a67d07dc0ea4af75344d161b22c4c1455093a8f1
➜  first-project git:(a67d07d) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
➜  first-project git:(a67d07d) ls -l
total 24
-rw-r--r--  1 hemakshipandey  staff  11 Aug 15 18:33 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 15 18:33 file2.txt
-rw-r--r--  1 hemakshipandey  staff  41 Aug 15 18:33 file3.txt
```

detached head state

normally head points to the branch and before this action head pointed to master branch. when head points to specific commit instead of specific branch, it is called "detached HEAD" state.

contents of HEAD file → reference to second commit

contents of master file master branch still points to last commit



→ git log

```
commit a67d07dc0ea4af75344d161b22c4c1455093a8f1 (HEAD)
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Fri Aug 13 14:44:21 2021 +0530

    Second commit

commit d0fa6cfc6f9bdef2c2ac54224f9eb5c09215814f
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 20:31:03 2021 +0530

    first commit

commit a6a90726f0c078e392724f40e1f295539c9c16b3
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 18:32:44 2021 +0530

    Our very first commit in the project
(END)
```

↳ detached head state

```
→  first-project git:(a67d07d) git cat-file -p d0fa6c
tree 6949aa9ffa787b0dd5629e991cc64a67106bf0aa
parent a6a90726f0c078e392724f40e1f295539c9c16b3
author hemakshi1234 <mahe173fas@gmail.com> 1628780463 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628780463 +0530

first commit
→  first-project git:(a67d07d) git cat-file -p a6a907
tree 3b95df0ac6365c72e9b0ff6c449645c87e6e1159
author hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530

Our very first commit in the project
→  first-project git:(a67d07d) git log
→  first-project git:(a67d07d) git checkout a6a907
Previous HEAD position was a67d07d Second commit
HEAD is now at a6a9072 Our very first commit in the project
→  first-project git:(a6a9072) cat .git/HEAD
a6a90726f0c078e392724f40e1f295539c9c16b3
→  first-project git:(a6a9072) ls -l
total 16
-rw-r--r--  1 hemakshipandey  staff  11 Aug 15 18:33 file1.txt
-rw-r--r--  1 hemakshipandey  staff  30 Aug 15 18:33 file2.txt
→  first-project git:(a6a9072) git ls-files -s
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
→  first-project git:(a6a9072) git log

[5]  + 2875 suspended  git log
→  first-project git:(a6a9072) git checkout master
Previous HEAD position was a6a9072 Our very first commit in the project
Switched to branch 'master'
→  first-project git:(master) git log

[6]  + 2909 suspended  git log
→  first-project git:(master) cat .git/HEAD
ref: refs/heads/master
→  first-project git:(master) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
→  first-project git:(master)
```

→ SHA1 hash to first commit

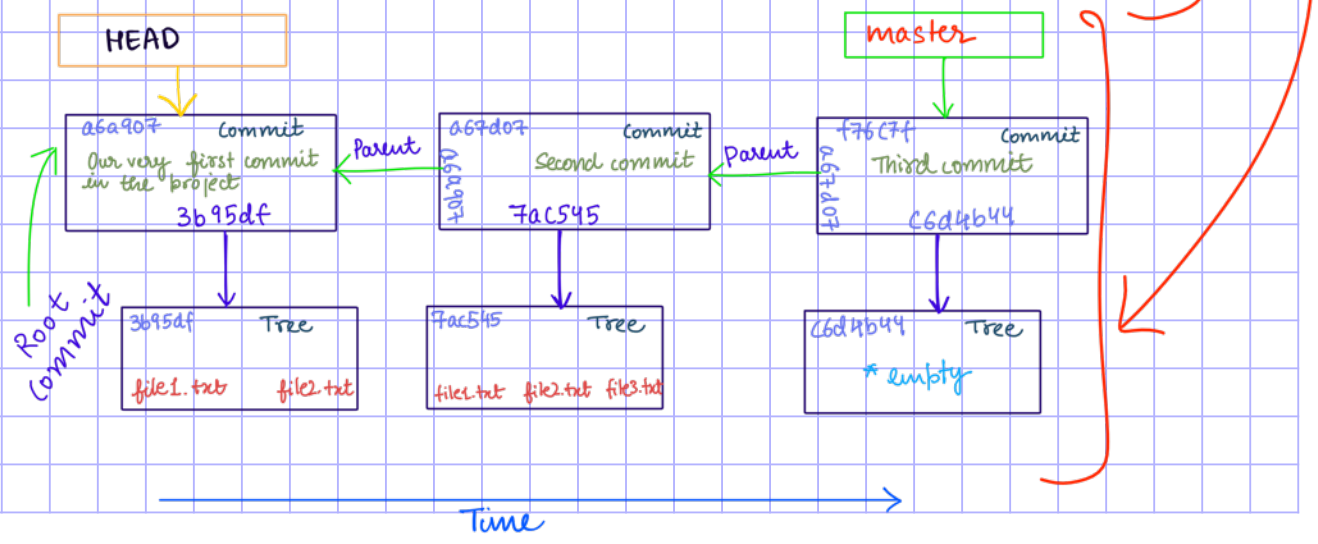
contents of HEAD file → SHA1 of first commit

now head points again at master branch

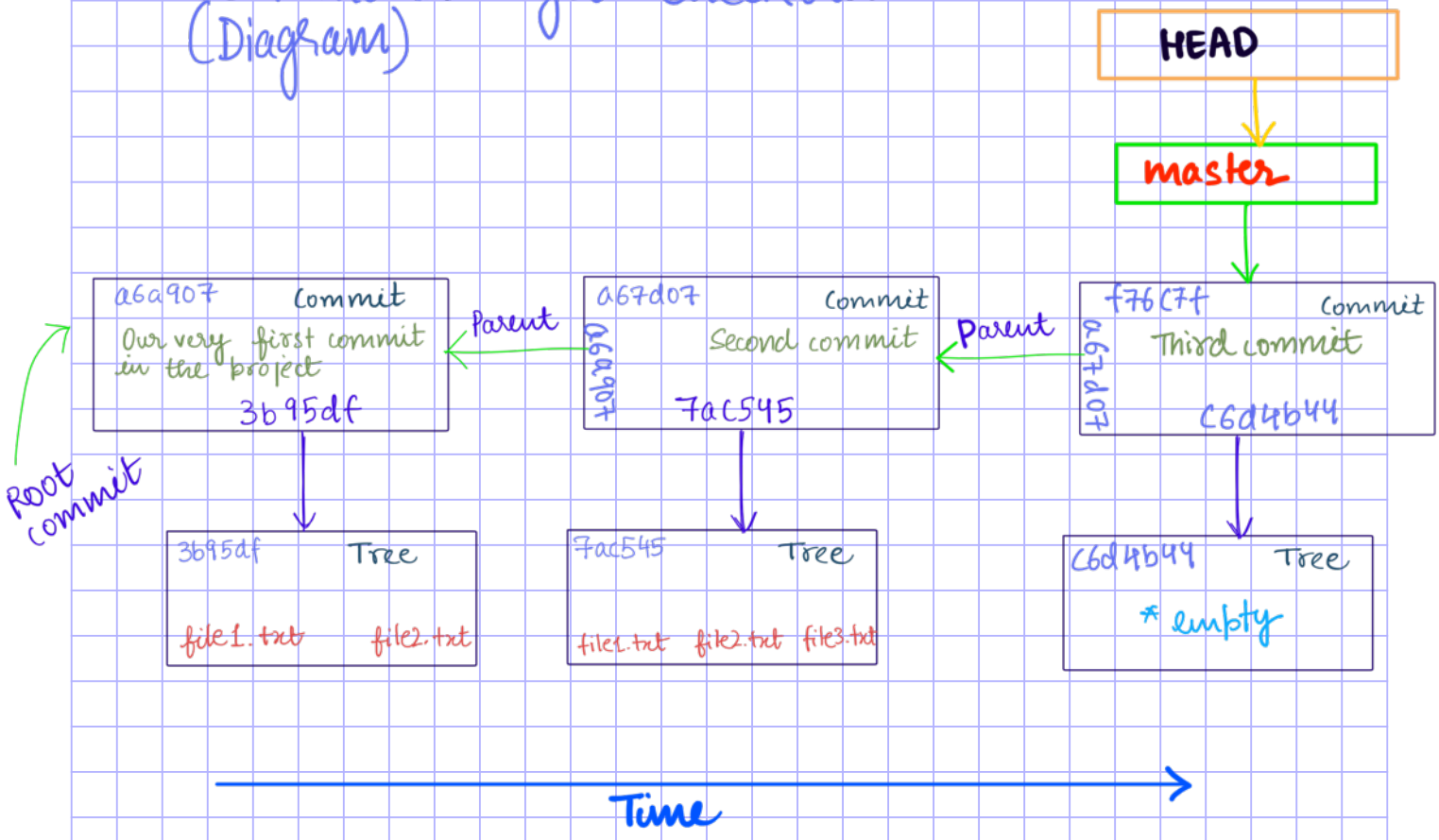
SHA1 hash of last commit

→ git log

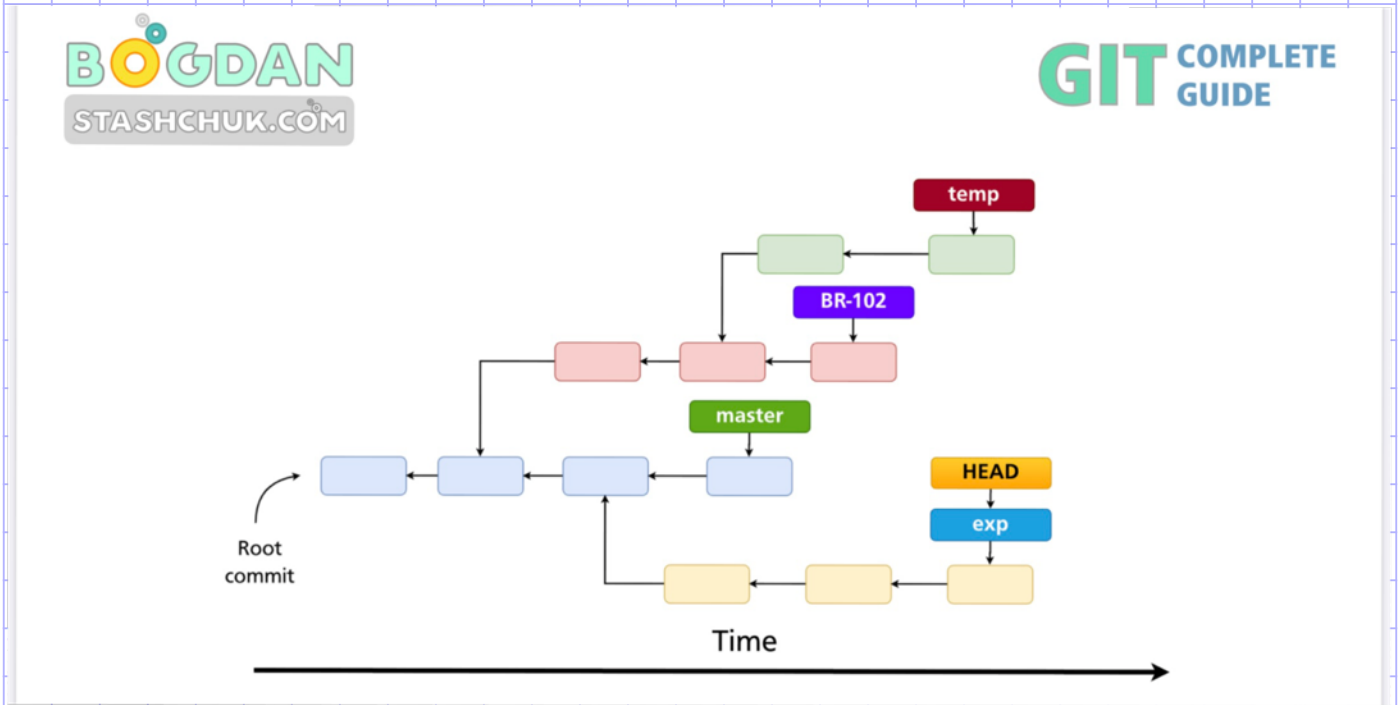
```
commit a6a90726f0c078e392724f40e1f295539c9c16b3 (HEAD)
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 18:32:44 2021 +0530

    Our very first commit in the project
(END)
```



→ Command → git checkout master (Diagram)



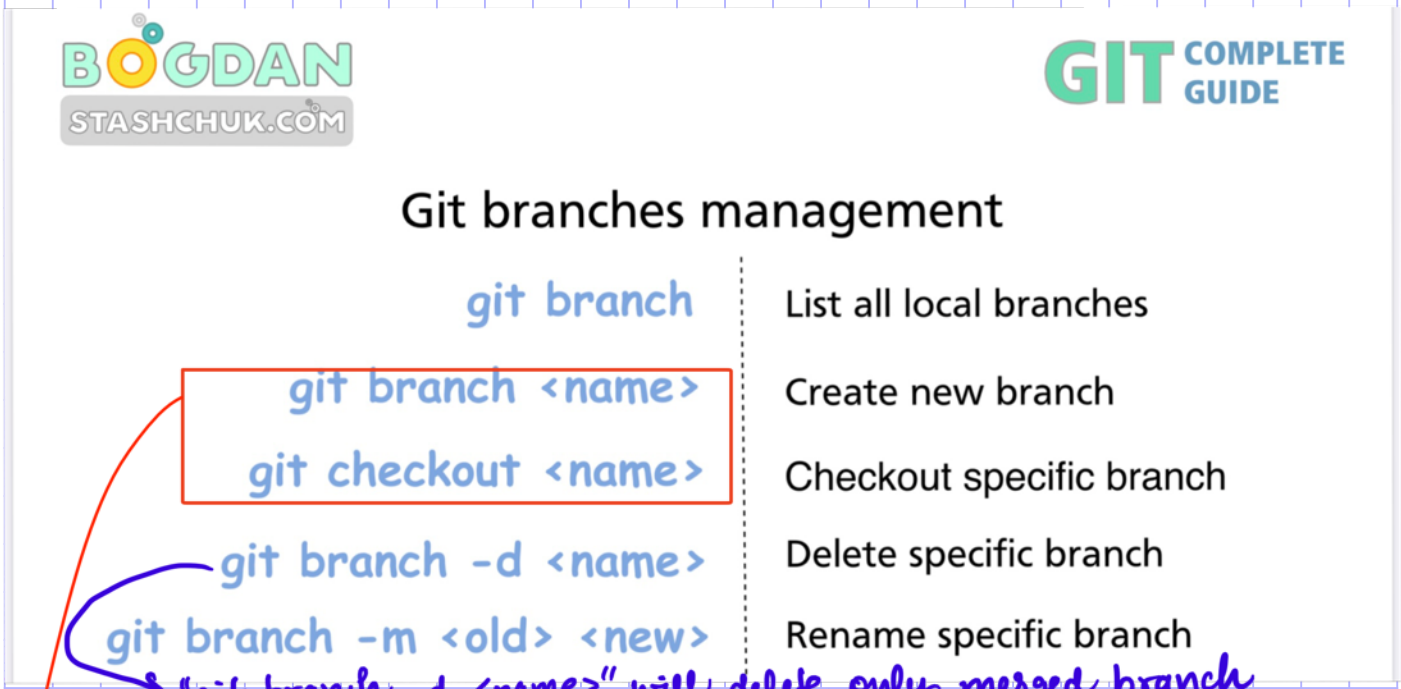
Why do we need branches



→ Every branch is actually separate and independent history of our project and we can develop different features in parallel.

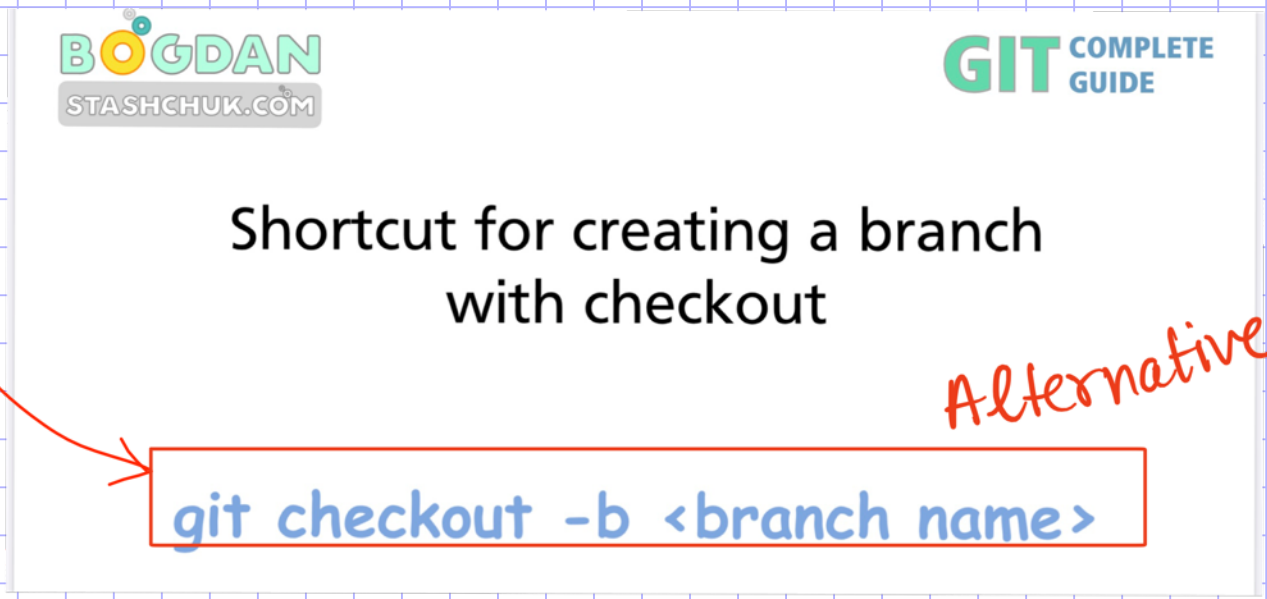
→ More than that, if multiple people are working on the same project, they can work in separate branches and they can do their job completely independent of each other. Branches greatly simplify collaboration process when multiple people work.

Git Branches Management



→ "git branch -d <name>" will delete only merged branch

If you want to delete non-merged branch use "-D" option



Create new branch

```
* master
~
```

```
→  first-project git:(master) git branch

[8]  + 3135 suspended  git branch
→  first-project git:(master) git branch temp
→  first-project git:(master) git branch

[9]  + 3168 suspended  git branch
→  first-project git:(master) ls .git/refs/heads
master temp
→  first-project git:(master) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
→  first-project git:(master) cat .git/refs/heads/temp
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
→  first-project git:(master)
```

listing branches

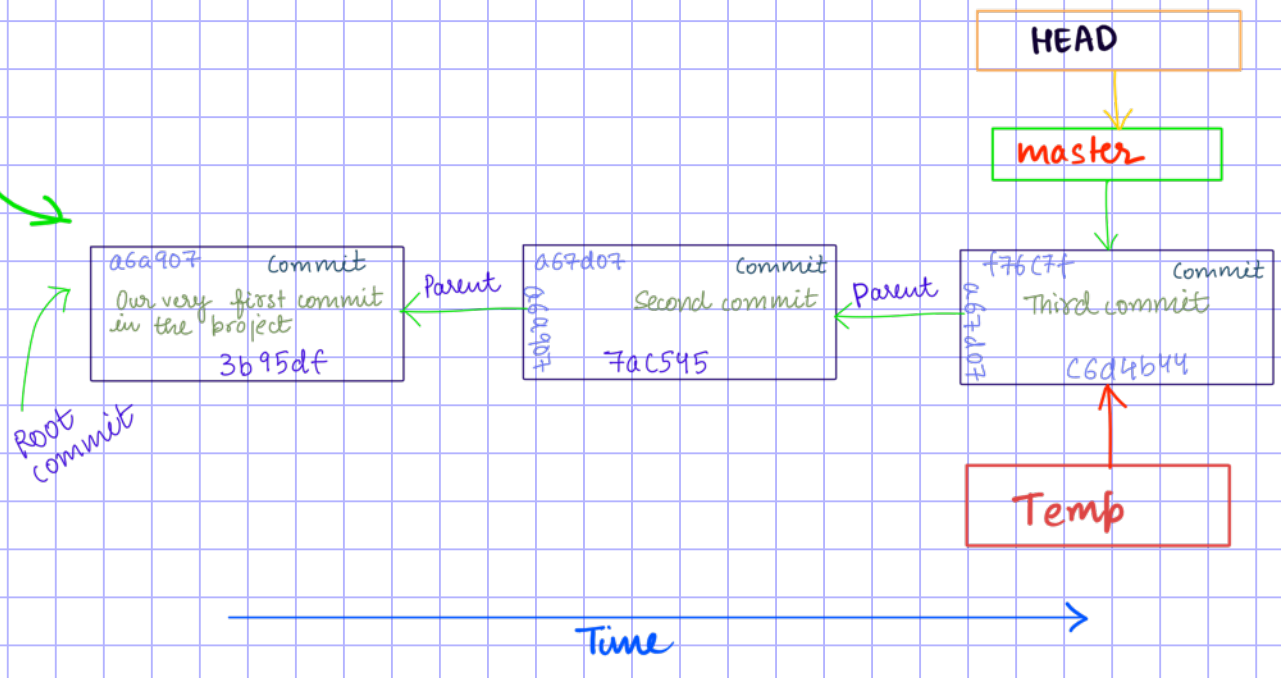
```
* master
  temp
(END)
```

SHA1 hash of third commit

same as contents of master file

```
→  first-project git:(master) cat .git/HEAD
ref: refs/heads/master
→  first-project git:(master) git checkout temp
Switched to branch 'temp'
→  first-project git:(temp) cat .git/HEAD
ref: refs/heads/temp
→  first-project git:(temp) cat .git/refs/heads/temp
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
→  first-project git:(temp) git checkout master
Switched to branch 'master'
→  first-project git:(master) cat .git/HEAD
ref: refs/heads/master
→  first-project git:(master) git branch -m temp new-temp
→  first-project git:(master)
```

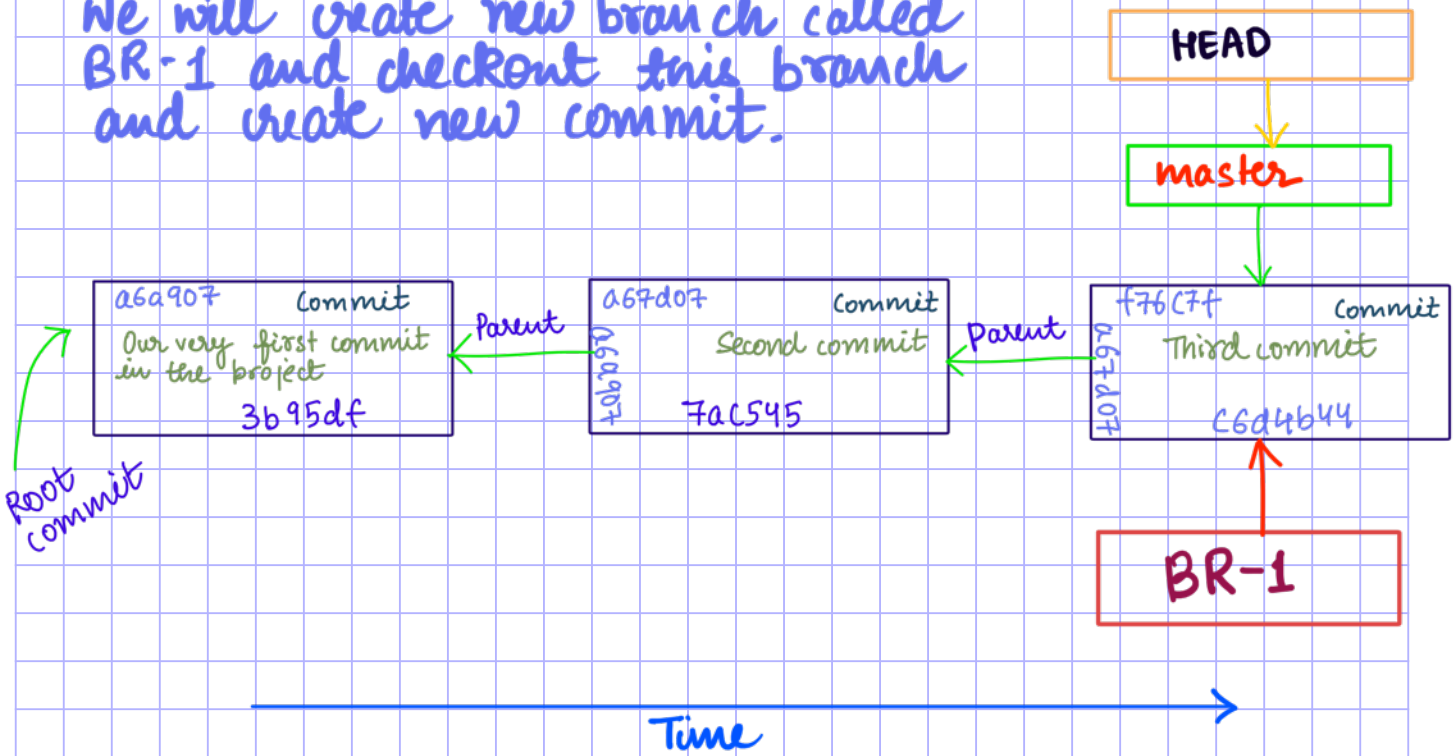
renaming temp

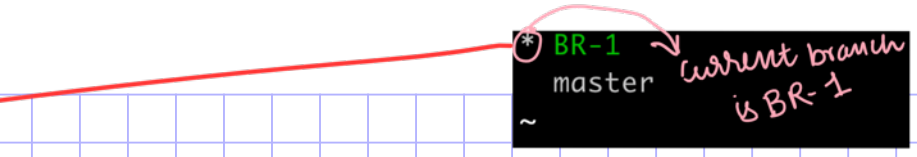


```
→  first-project git:(master) git branch

[10]  + 3356 suspended  git branch
→  first-project git:(master) git checkout new-temp
Switched to branch 'new-temp'
→  first-project git:(new-temp) git checkout master
Switched to branch 'master'
→  first-project git:(master) git branch -d new-temp
Deleted branch new-temp (was f76c7fc).
→  first-project git:(master)
```

Commit changes in new branch

We will create new branch called BR-1 and checkout this branch and create new commit.






```
➜  ~ cd Desktop/first-project
➜  first-project git:(master) git checkout -b BR-1
Switched to a new branch 'BR-1'
➜  first-project git:(BR-1) git branch

[1]  + 1308 suspended  git branch
➜  first-project git:(BR-1) ls
➜  first-project git:(BR-1) echo "Hello, Git" > file4.txt
➜  first-project git:(BR-1) ✗ ls
file4.txt
➜  first-project git:(BR-1) ✗ cat file4.txt
Hello, Git
➜  first-project git:(BR-1) ✗ git status
On branch BR-1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4.txt

nothing added to commit but untracked files present (use "git add" to track)
➜  first-project git:(BR-1) ✗ git add file4.txt
➜  first-project git:(BR-1) ✗ git status
On branch BR-1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file4.txt

➜  first-project git:(BR-1) ✗ git commit -m "First commit in the BR-1 branch"
[BR-1 d4fec14] First commit in the BR-1 branch
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
➜  first-project git:(BR-1) git status
On branch BR-1
nothing to commit, working tree clean
➜  first-project git:(BR-1) ls
file4.txt
➜  first-project git:(BR-1) git ls-files -s
100644 fc22720f2257083bd71bfadc42654cd8b481b6eb 0       .DS_Store
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file4.txt
➜  first-project git:(BR-1)
```

new branch BR-1 is created and we have automatically switched to new branch

There is nothing in our working directory

create new file

contents of working directory

content of file

adding to staging area

making commit while working in BR-1 branch

working directory

Staging area

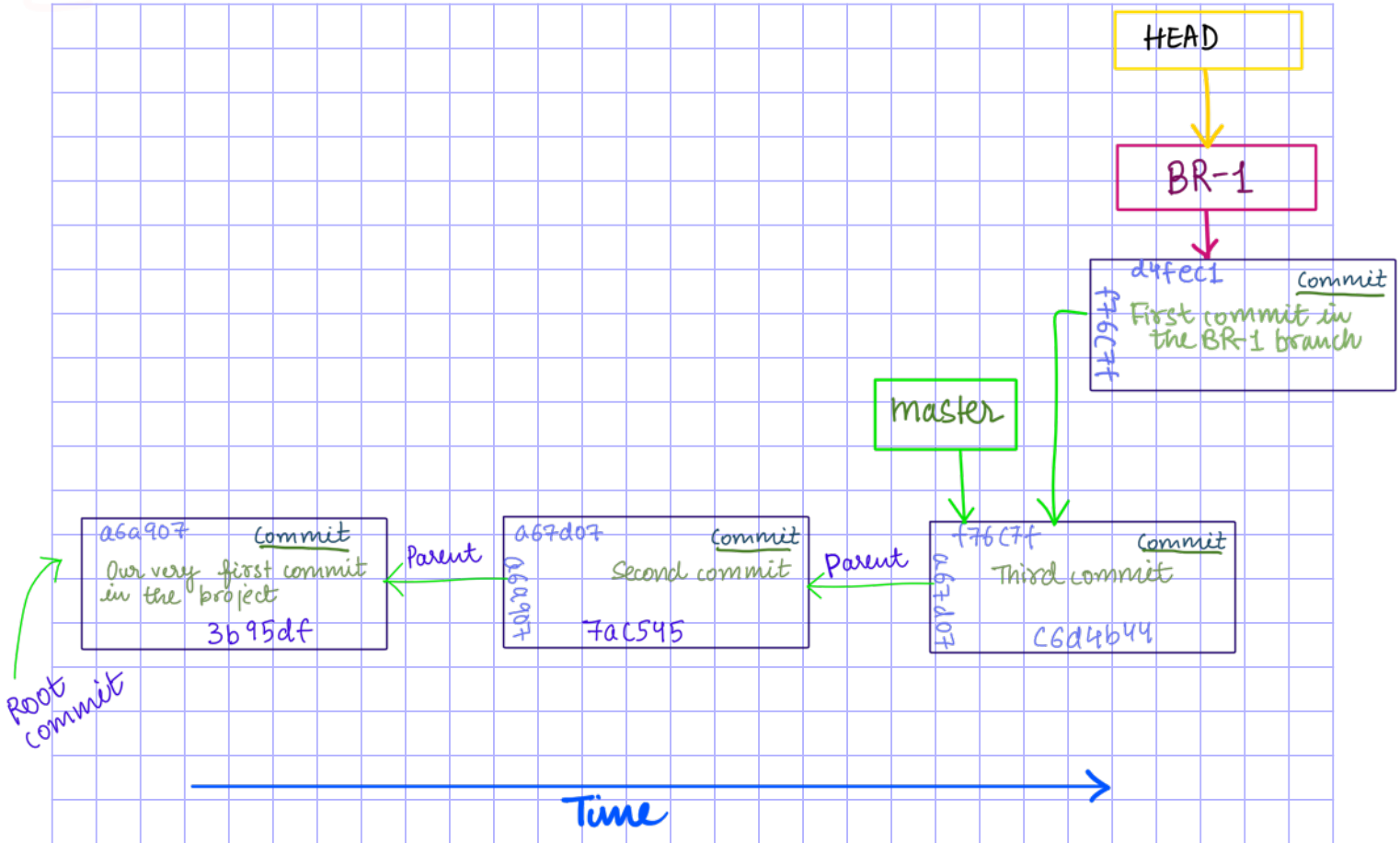
Explore commit in the new branch

→ git log

```
commit d4fec14d581e6bf5a6a7dd794e465ea7541f2cd2 (HEAD -> BR-1)
Author: Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com>
Date:   Mon Aug 16 17:00:51 2021 +0530

    First commit in the BR-1 branch

commit f76c7fcf9fdaa7a694bb777abafe06980ce4c200 (master)
Author: Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com>
Date:   Sun Aug 15 13:43:30 2021 +0530

    Third commit

commit a67d07dc0ea4af75344d161b22c4c1455093a8f1
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Fri Aug 13 14:44:21 2021 +0530

    Second commit

commit d0fa6cfc6f9bdef2c2ac54224f9eb5c09215814f
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 20:31:03 2021 +0530

    first commit

commit a6a90726f0c078e392724f40e1f295539c9c16b3
Author: hemakshi1234 <mahe173fas@gmail.com>
Date:   Thu Aug 12 18:32:44 2021 +0530

    Our very first commit in the project
```

```
➜  first-project git:(BR-1) git log

[2]  + 1639 suspended  git log
➜  first-project git:(BR-1) git cat-file -p d4fec1
tree a96375f369761690ca878d73b9900b43e9964727
parent f76c7fcf9fdaa7a694bb777abafe06980ce4c200
author Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629113451 +0530
committer Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629113451 +0530

First commit in the BR-1 branch
➜  first-project git:(BR-1) git cat-file -p f76c7fc
tree c6d4b44556ad794e327673180dc5e535585f504b
parent a67d07dc0ea4af75344d161b22c4c1455093a8f1
author Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629015210 +0530
committer Hemakshi Pandey <57315023+hemakshi1234@users.noreply.github.com> 1629015210 +0530

Third commit
➜  first-project git:(BR-1)
➜  first-project git:(master)
➜  first-project git:(BR-1) cat .git/HEAD
ref: refs/heads/BR-1
➜  first-project git:(BR-1) cat .git/refs/heads/master
f76c7fcf9fdaa7a694bb777abafe06980ce4c200
➜  first-project git:(BR-1)
```

SHA1 hash of third commit



Git reuses blob with the same contents

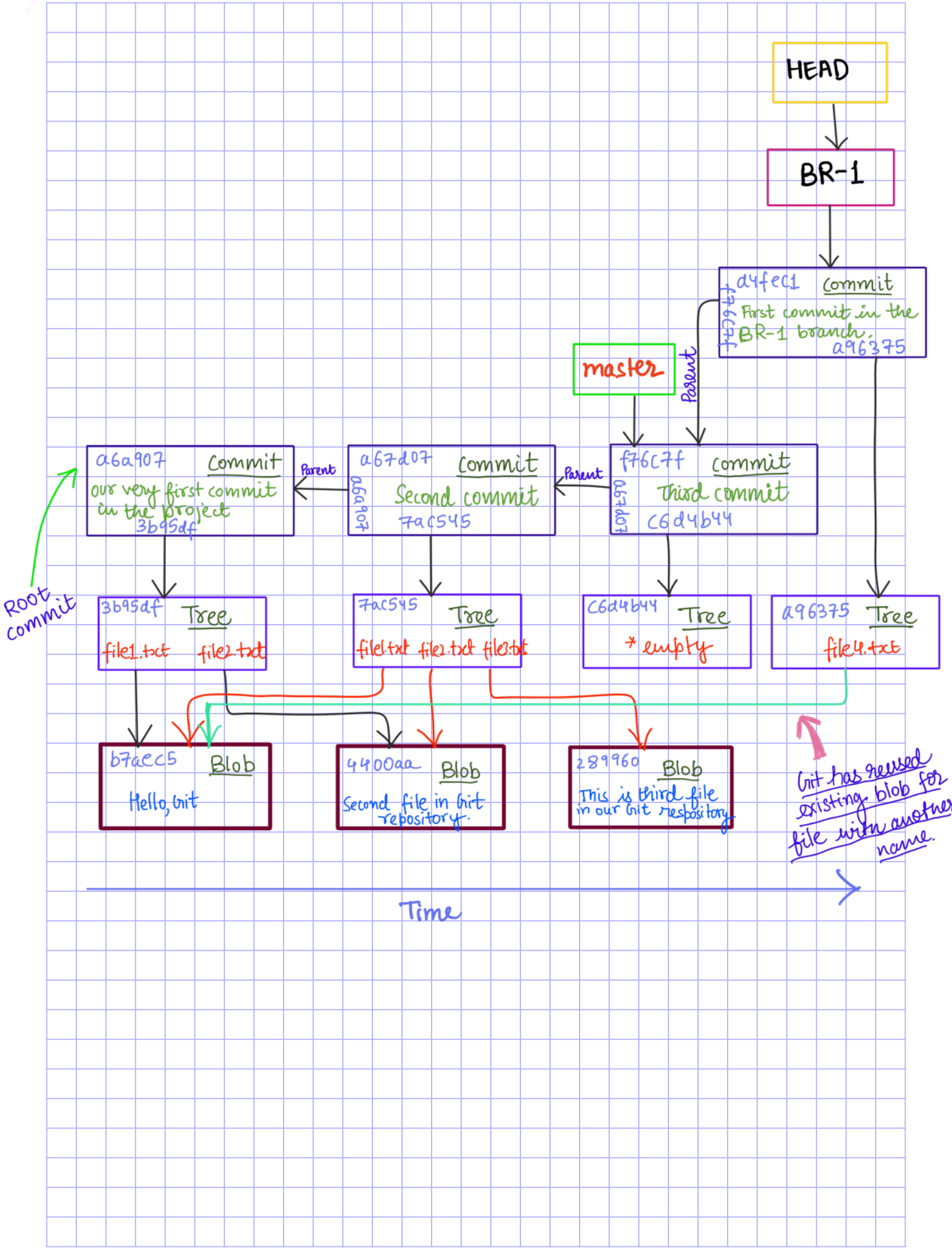
```
→  first-project git:(BR-1) git cat-file -p a96375f
100644 blob fc22720f2257083bd71bfadc42654cd8b481b6eb    .DS_Store
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file4.txt
→  first-project git:(BR-1) git cat-file -t b7aec5
blob
→  first-project git:(BR-1) ✗ git cat-file -p b7aec5
Hello, Git
→  first-project git:(BR-1) ✗
```

→ file1.txt and file4.txt have same contents "Hello, Git"

→ It means that Git has reused existing blob even if filename is different

```
→  first-project git:(BR-1) ✗ git checkout a6a907
error: Your local changes to the following files would be overwritten by checkout:
        .DS_Store
Please commit your changes or stash them before you switch branches.
Aborting
→  first-project git:(BR-1) ✗ git add .
→  first-project git:(BR-1) ✗ git commit -m "ds store first commit"
[BR-1 913f90d] ds store first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
→  first-project git:(BR-1) git checkout a6a907
Note: switching to 'a6a907'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at a6a9072 Our very first commit in the project
→  first-project git:(a6a9072) ls
file1.txt file2.txt
→  first-project git:(a6a9072) cat file1.txt
Hello, Git
```

```
→  first-project git:(a6a9072) git cat-file -p a6a907
tree 3b95df0ac6365c72e9b0ff6c449645c87e6e1159
author hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530
committer hemakshi1234 <mahe173fas@gmail.com> 1628773364 +0530

Our very first commit in the project
→  first-project git:(a6a9072) git cat-file -p 3b95df0
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
→  first-project git:(a6a9072) git checkout BR-1
Previous HEAD position was a6a9072 Our very first commit in the project
Switched to branch 'BR-1'
→  first-project git:(BR-1) ls
file4.txt
→  first-project git:(BR-1) git checkout master
Switched to branch 'master'
→  first-project git:(master) ls
→  first-project git:(master)
```


HEAD
BR-1
d4fec1 Commit
First commit in the BR-1 branch.
a96375
f76c7f
master
Parent
a6a907 Commit
our very first commit in the project
3b95df
Parent
a67d07 Commit
Second commit
7ac545
a6a907
Parent
f76c7f Commit
Third commit
C6d4b44
a67d07
Root commit
3b95df Tree
file1.txt file2.txt
7ac545 Tree
file1.txt file2.txt file3.txt
C6d4b44 Tree
* empty
a96375 Tree
file4.txt
b7aec5 Blob
Hello, Git
44004aa Blob
Second file in Git repository.
289960 Blob
This is third file in our Git respository
Git has reused existing blob for file with another name.
Time