17. Section 4 Introduction

1. Git stores four kinds of objects in repo.
   1. BLOB.
   2. Tree.
   3. Commit.
   4. Annotated Text.

PDF - How Git works under the hood

[4. How Git works under the hood (stashchuk.com)](https://stashchuk.com/static/git/4-how-git-works-under-the-hood.pdf)

19. Initialize new Git Repository

1. git init:
   1. To initialize **empty repo** even though directory may be not initialized.
   2. .git hidden folder will be created.
   3. .git is local git repo.
2. ls -la : To display hidden folders.
3. Bash Shell vs Git Bash

|  |  |
| --- | --- |
| Bash Shell | Git Bash |
| Doesn’t show if current dir is under git control. |  |
| Doesn’t show the current branch. |  |

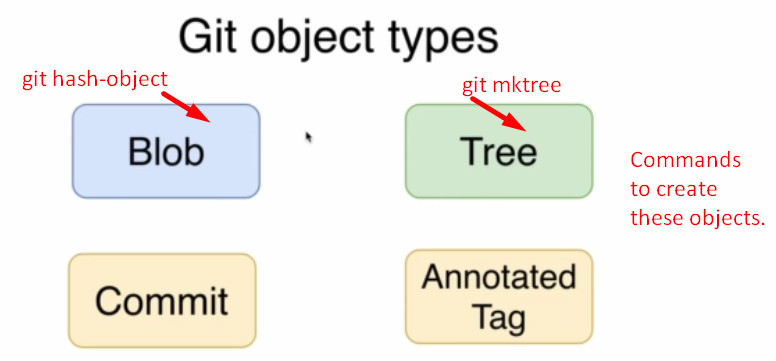
20. Overview of .git folder

1. Command: start . or explorer .
2. under .git  
   above, git has its own **file system**.  
   Objects directory stores git objects (BLOB, Tree, Commit, Annotated Text)

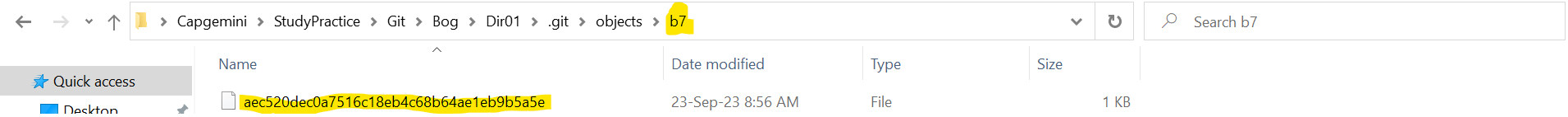
21. Git object Types

1. Those four objects are used to track files and changes in the files.
   * 1. **Blob**: Any kind of file is stored as blob.
     2. **Tree**: To store directory info. Dir maybe empty or contain other files and/or directories.
        1. Tree is set of blobs (files) and/or other trees (dirs)
     3. **Commit**: This object is used to store different versions of our project.
     4. **Annotated Tag:** Text pointer to other commits.
2. Low level commands:
   1. **git hash-object**: To create git object only BLOB.
   2. **git cat-file**: To read git object.

22. Writing new git object with hash-object.

1. 
2. 
3. d

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Linux Command | input: All formats allowed |  | git command | Input source | To save object under .git/objects |
| echo | “Hello, Git” | | | git hash-object | --stdio | -w |
|  | ‘Hello, Git’ |  |  |  |  |
|  | Hello, Git |  |  |  |  |

Let’s assume the hash for the BLOB 🡺 **b7**aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  
output: 

hash = Folder Name + File Name.

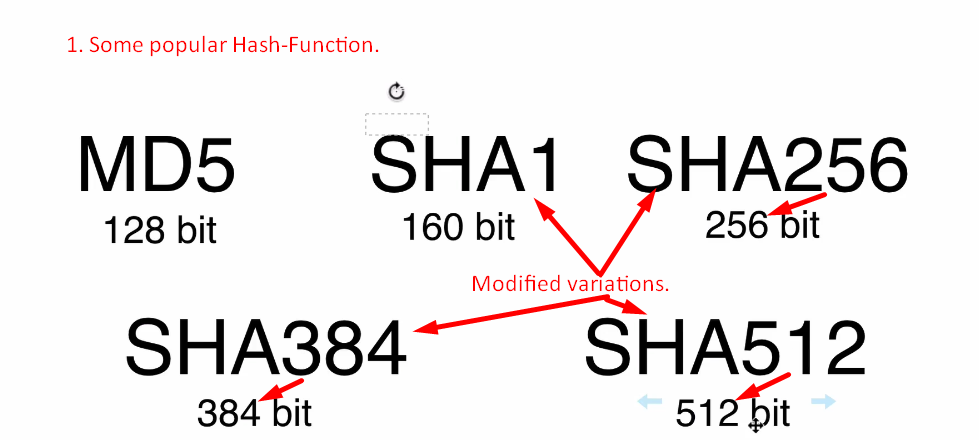
23. JSON vs Git database

1. Git DB Structure similar to JSON.
   1. **Key**: The hash of the file.
   2. **Value**: The file itself.  
      But in Git, even can not be duplicated as key is calculated based on file’s content.   
      If file content is same then same key will be achieved.
2. Hash = Folder Name + File Name.

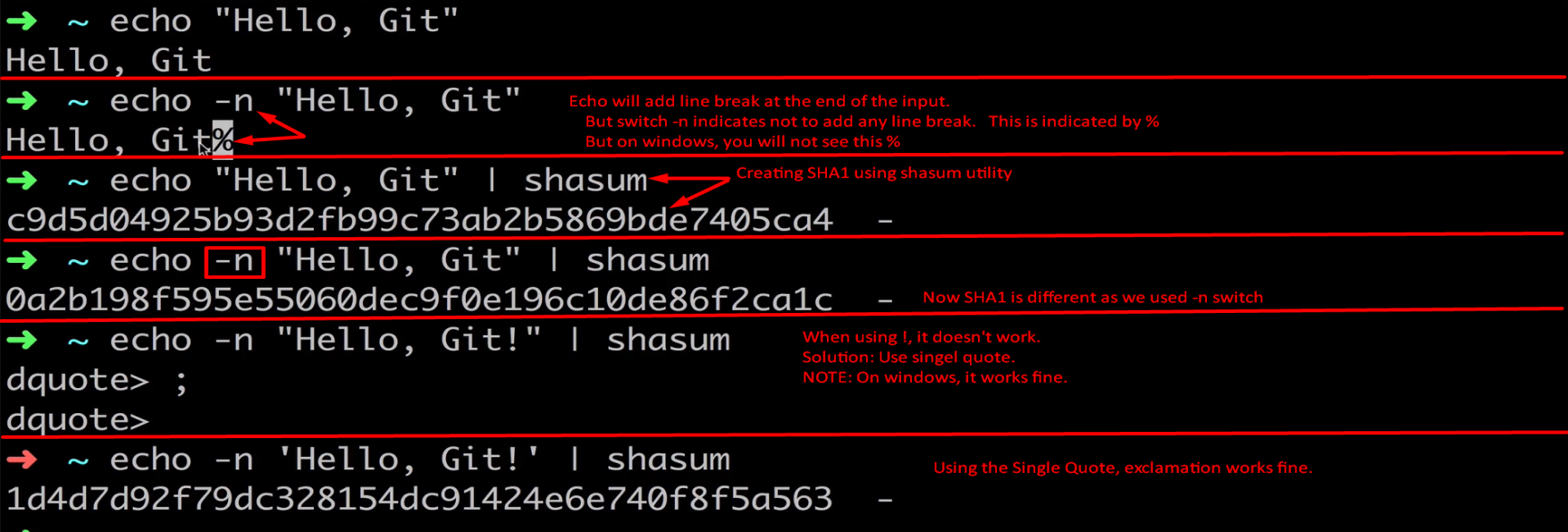
24. What is hash function

1. We know that based on file’s content, hash code is generated and then the generated blob is stored under .git/objects creating a new folder whose name is first 2 characters of the hash code and blob is stored inside this folder and its name is the remaining hash characters.  
   file content 🡪 hash-function() 🡪 BLOB 🡪 Folder
2. Always Fixed Length Hash.
3. Hash functions are “One-Way” function.
4. Same hash function will create same hash code for the same input.

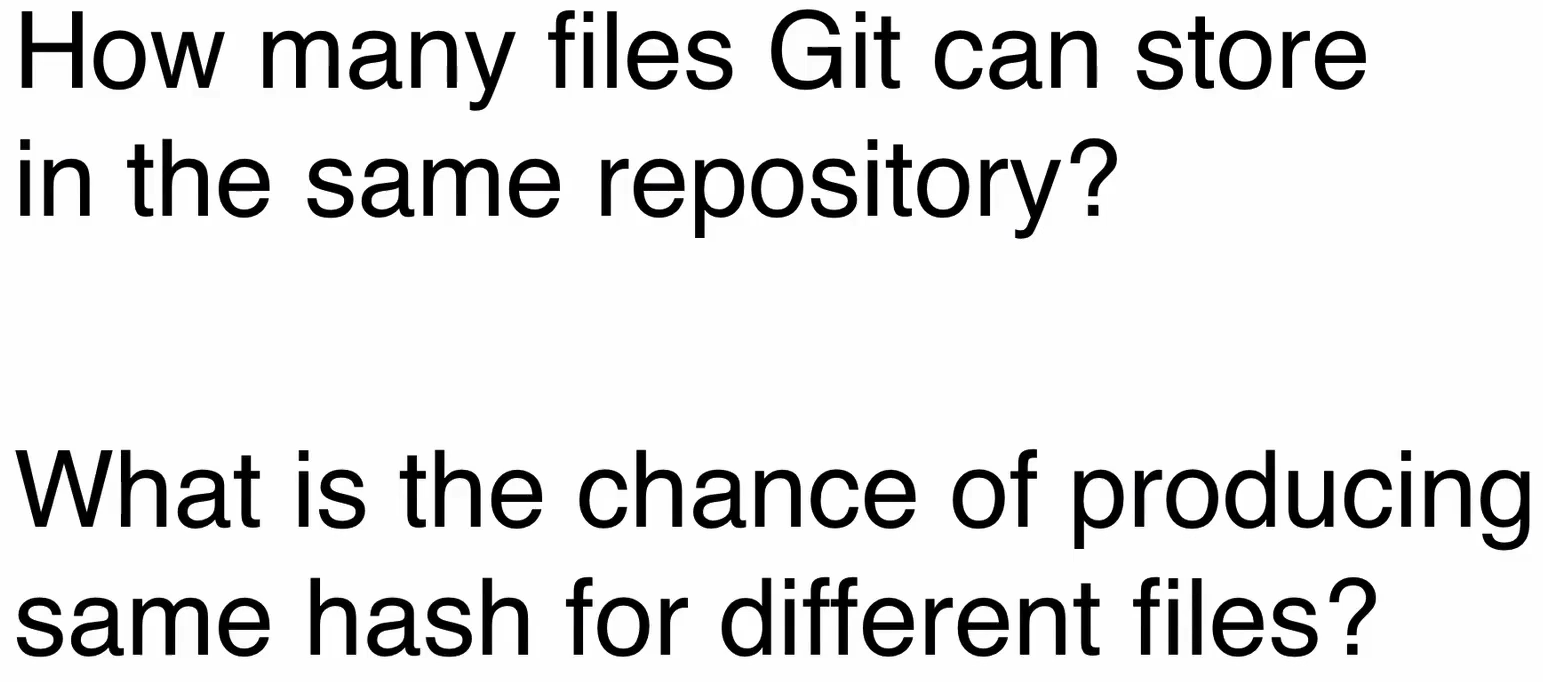
25. Hash functions overview

1. 
2. Variable Length input but Fixed length Hash.

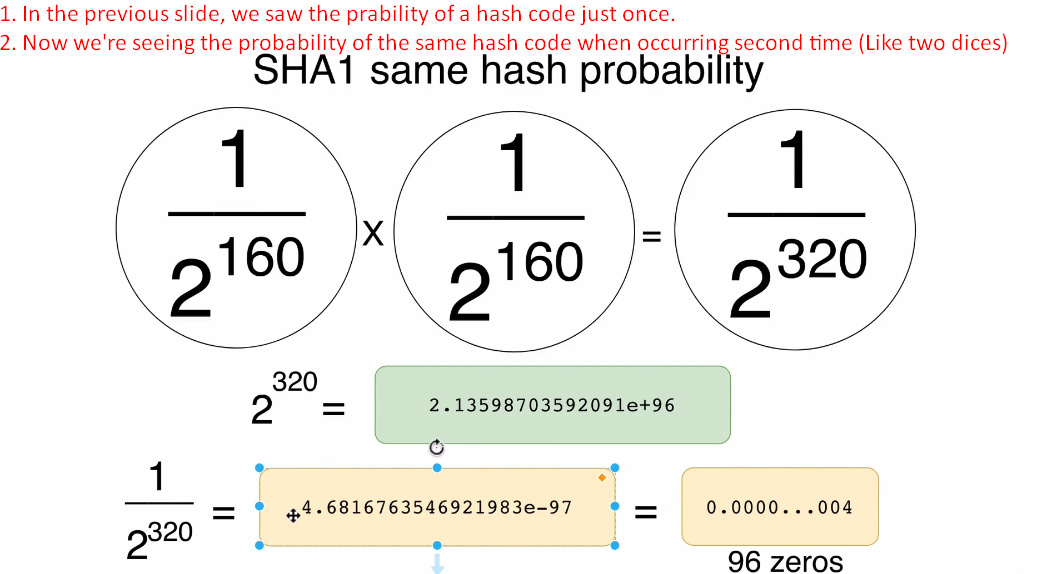
25. SHA1 Hash Function

1. SHA1: 160Bit = 40 Hexadecimal Characters, hash Function or algo.
2. **Commands**:

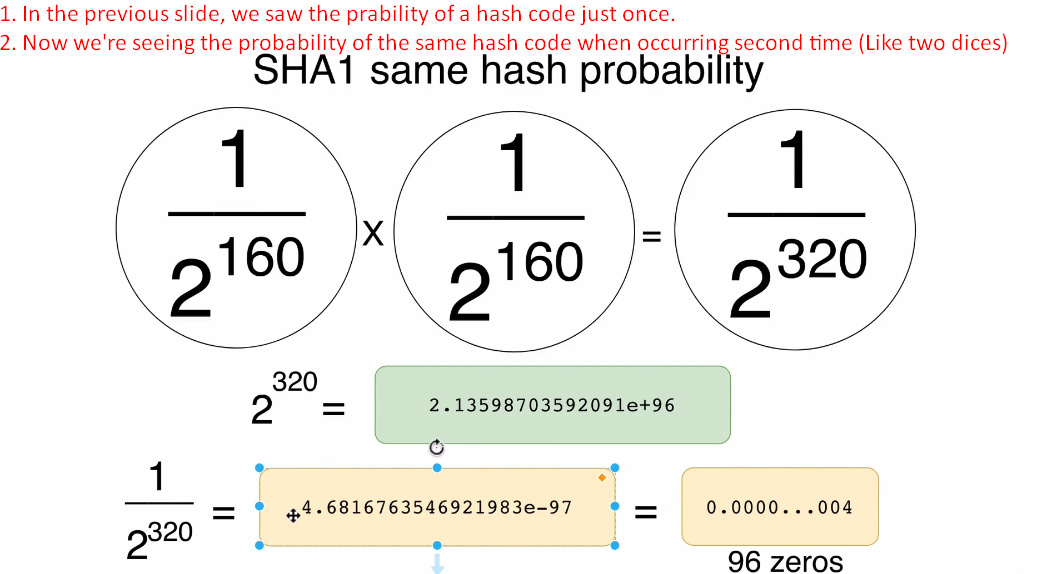
26. How many files Git can store

1.   
   Same hash for different file which is called **Hash Collision**.
2. Git uses SHA1 (160bit) hash function. So 2^160 different **hash codes** after that collision.

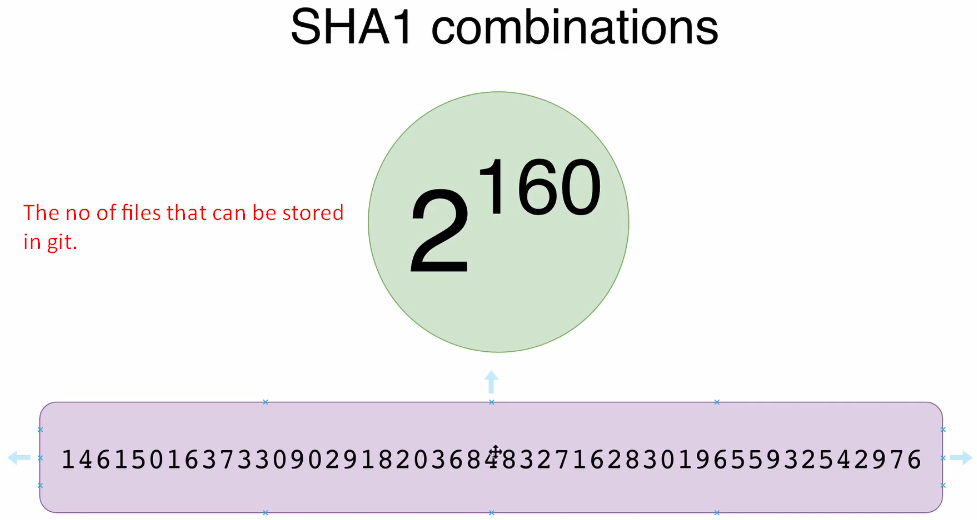
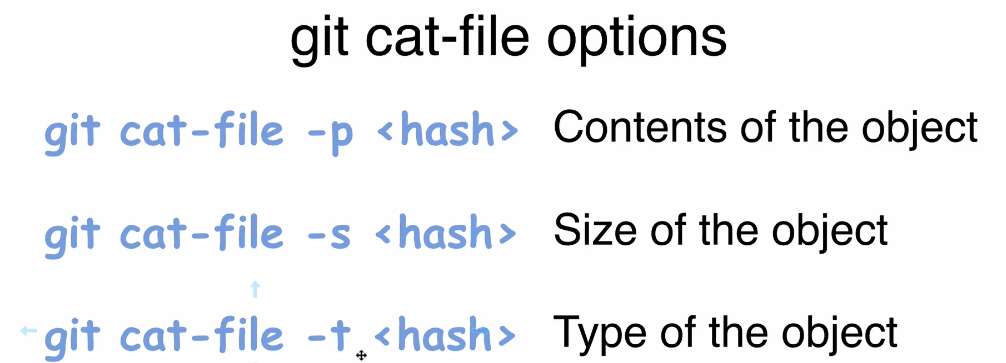
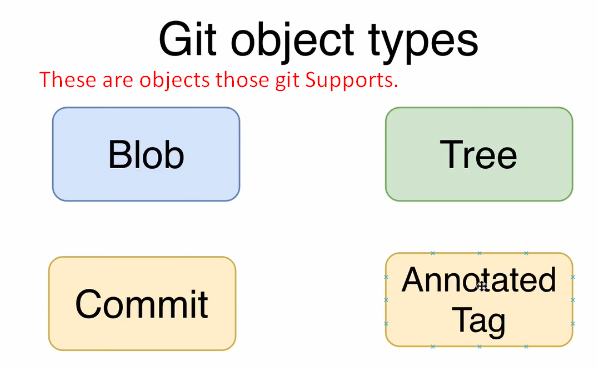
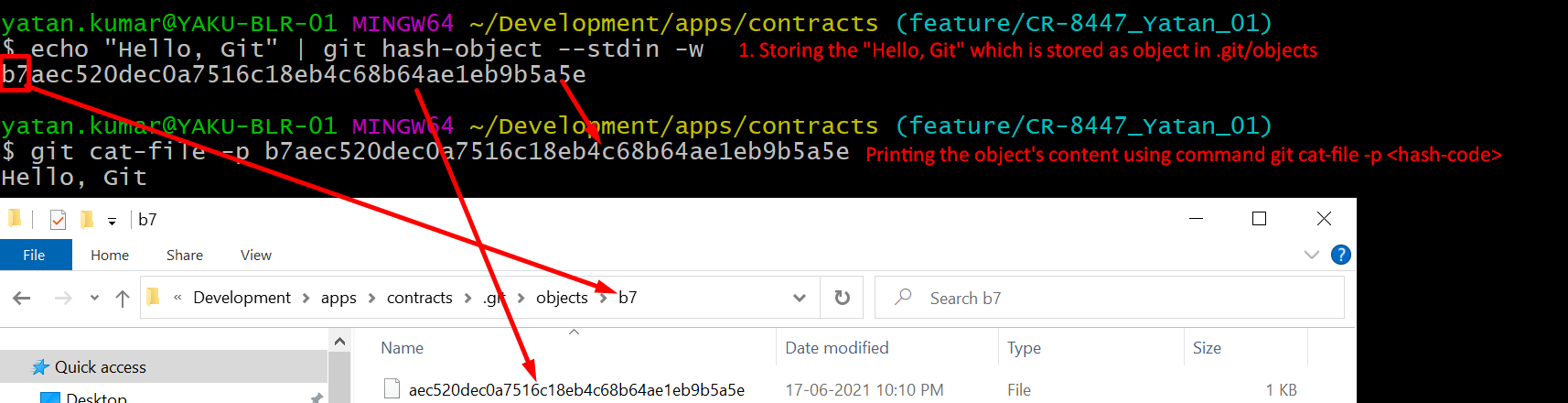
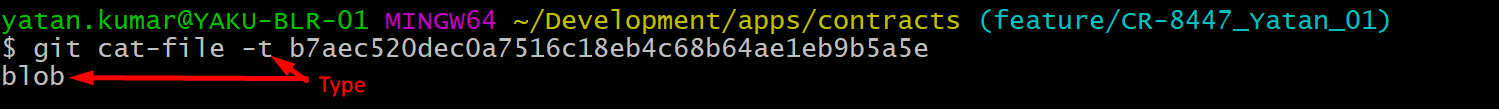
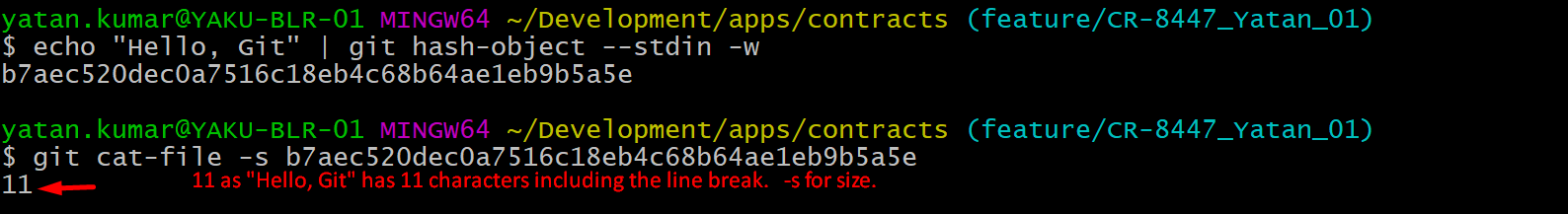
27. Probability Theory In Dice Game

1. 

28. Git Hash Collision Prabability

1. 

32. Exploring git objects with git cat-file commands

1. As git uses JSON similar format => Key, Value Pair where key is hash of the file and we can create 2^160 combinations so that many files can be stored in Git DB/Repo.
2. **Commands**:  
   
3. git cat-file -p <hash> Seeing the content of a blob.  
   
4. git cat-file -t <hash>  
   
5. git cat-file -s <hash>  
   

33. Create new Git Blob based on a file

1. 
2. hash-object calculates hashcode based on input, type, size whereas shasum uses only input. So hashcodes are different by both even though same input file and input, type, size are stored in blob itself.

35. Contents of Git Objects

1. Diagram

   Description automatically generated

36. What we have so far

37. Tree Objects in Git

1. A diagram of a git object

   Description automatically generated
2. A screenshot of a computer code

   Description automatically generated
   1. Tree contains a set of links to other Git Objects under it.
   2. **Every line has 4 fields.**
      1. Permission.
      2. Type of the Git Object.
      3. SHA1 HashCode.
      4. The file/Directory name (This is where file name is stored in tree object not in the blob object as we discussed in the previous lecture that blob doesn’t contain file name).

38. Git Object Permission

1. A close up of a web page

   Description automatically generated
2. A white background with blue text

   Description automatically generated

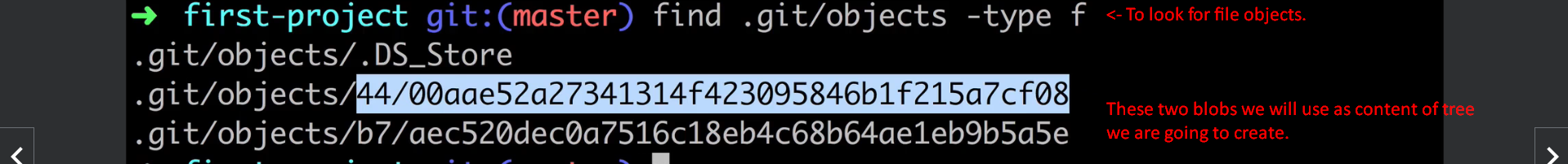
39. Creating Git Tree Object

1. A diagram of a computer

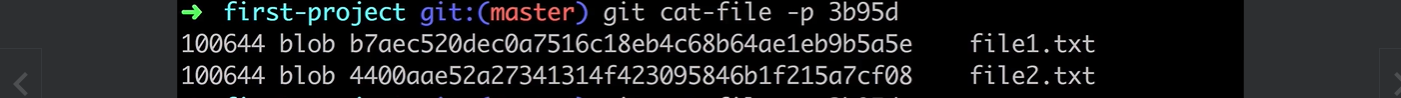
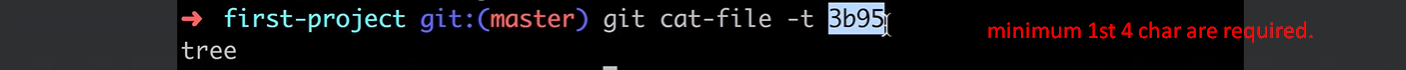
   Description automatically generated
2. A black text on a white background

   Description automatically generated
3. A screenshot of a computer

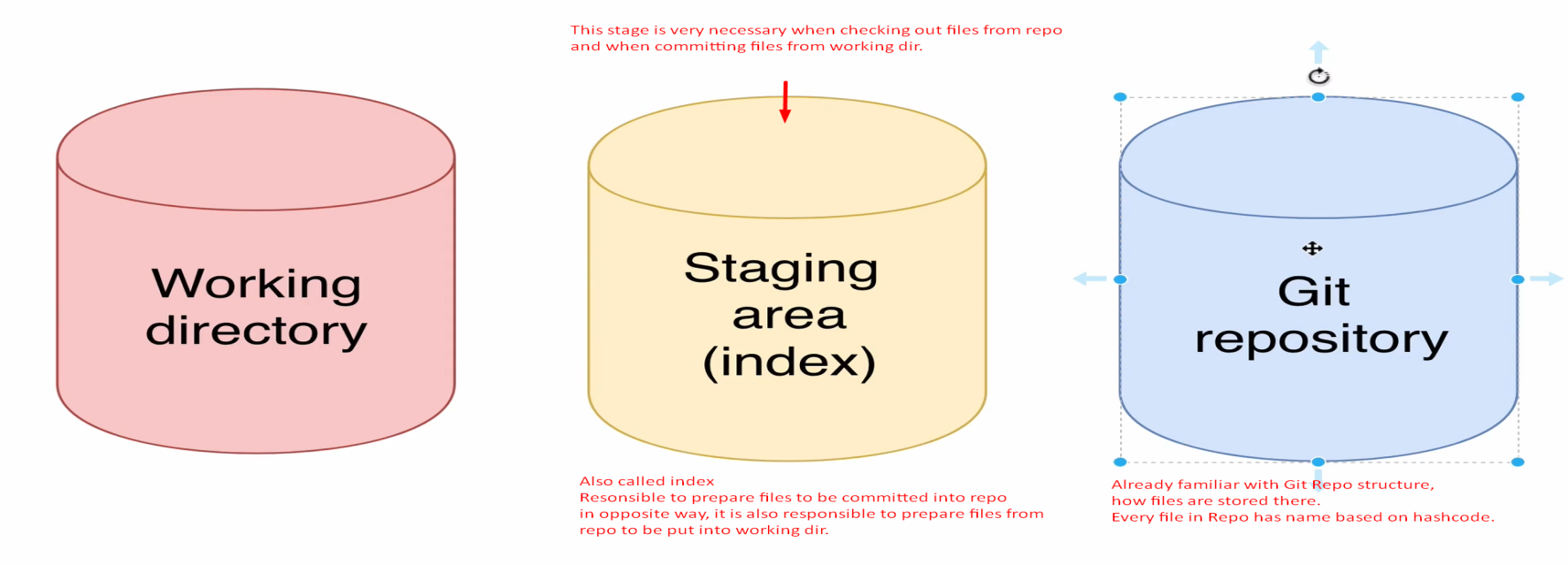
   Description automatically generated
4. A computer screen shot of a computer

   Description automatically generated
5. **Just Linux Command Info**:  
   

40. Examining Tree Object

1. Following SHA1 is SHA1 of the Git Tree Object Let us apply the following commands on the Git Tree Object.
2. git cat-file -p <sha1> 🡺 To see the content of a Git Object here Tree.  
   
3. git cat-file -s <sha1> 🡺 To see the size of Git Object here Tree.  
   
4. git cat-file -t <sha1> 🡺 To see the type of the Git Object.  
   

41. Working directory, Staging area and Git Repository

1. 
2. 
3. **Terms**: Working Directory, Staging Area (also called index), Git Repository.

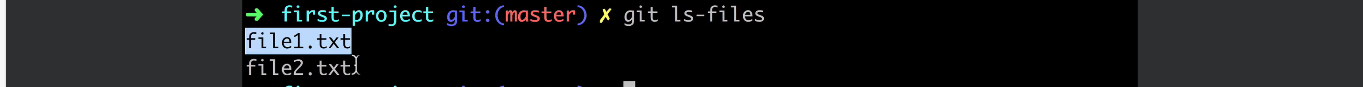
43. Git read-tree

1. A diagram of a diagram

   Description automatically generated

44. Read files in the Staging-Area Using git ls-files

1. A white cylinder with black text

   Description automatically generated
2.   
   A screenshot of a computer

   Description automatically generated

45. Git Checkout Index.

1. A diagram of a diagram

   Description automatically generated

46. How many Folders Can Be Created for Objects

1. 

48. Section 5 Introduction

49. Pdf Basic Git Operation

50. What is a commit

1. Git Object. So, it has its own SHA1. Used to store different versions/snapshots.  
   A diagram of a tree

   Description automatically generated  
   A black and white text

   Description automatically generated

51. Configure Git Author Name and Email

1. Username and email can be set globally and per project.
2. **Command**:  
   A screenshot of a computer

   Description automatically generated

A blue notepad with white text

Description automatically generated

52. Creating first Commit

1. **Always Remember**: Commit always points to Root Tree.A screenshot of a computer

   Description automatically generated