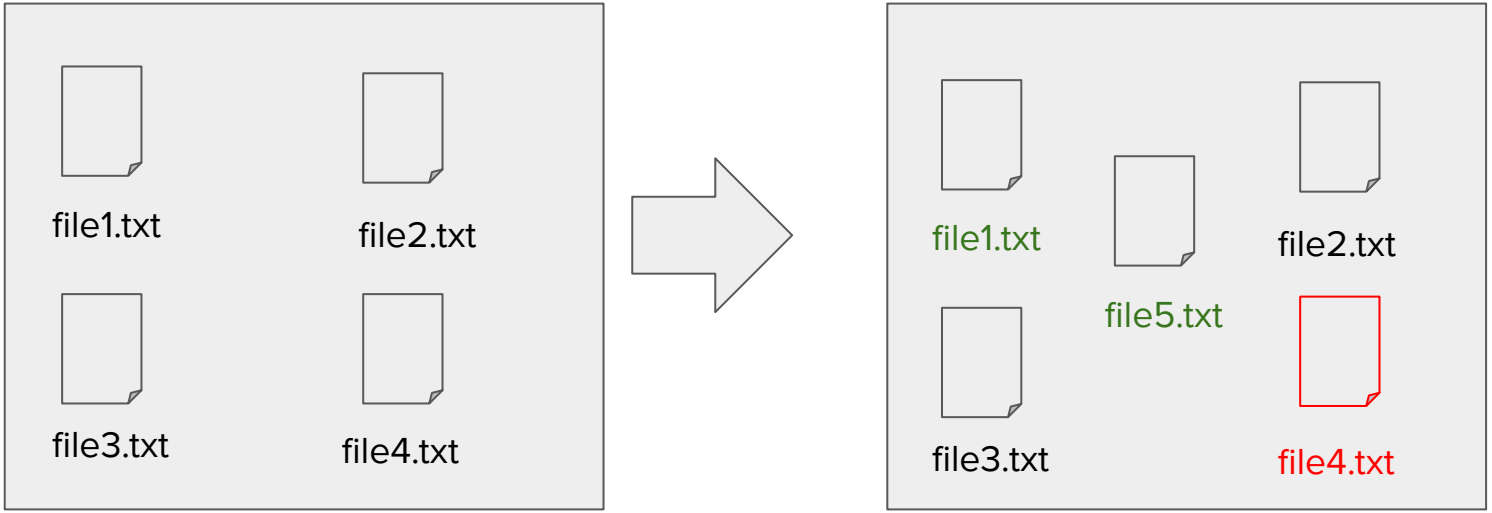
Introduction to version control

What does version mean?

- Version can be stated as state of project (group of files and directories)



Version 1

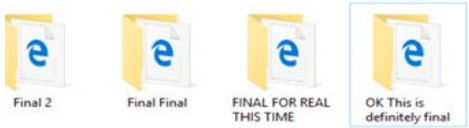
Version 2

Need for Version Control system

- VCSs track changes to a folder and its contents in a series of snapshots
- VCSs also maintain metadata like who created each snapshot, messages associated with each snapshot, and so on.
- They facilitate collaboration
  - Who wrote this module?
  - When was this particular line of this particular file edited? By whom? Why was it edited?

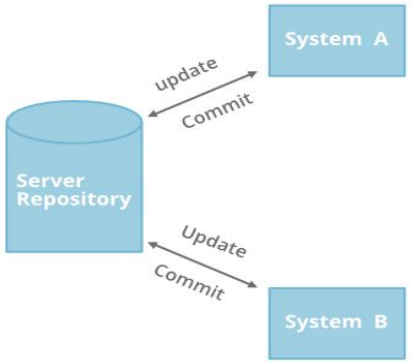
git

SUBVERSION

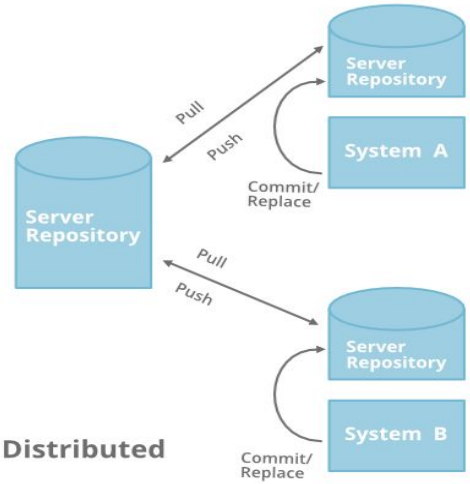


Centralized vs Decentralized Version Control system

- CVCS : Require you to contact the central database and hence require network access.
- DVCS: Each and every node has a copy of the database

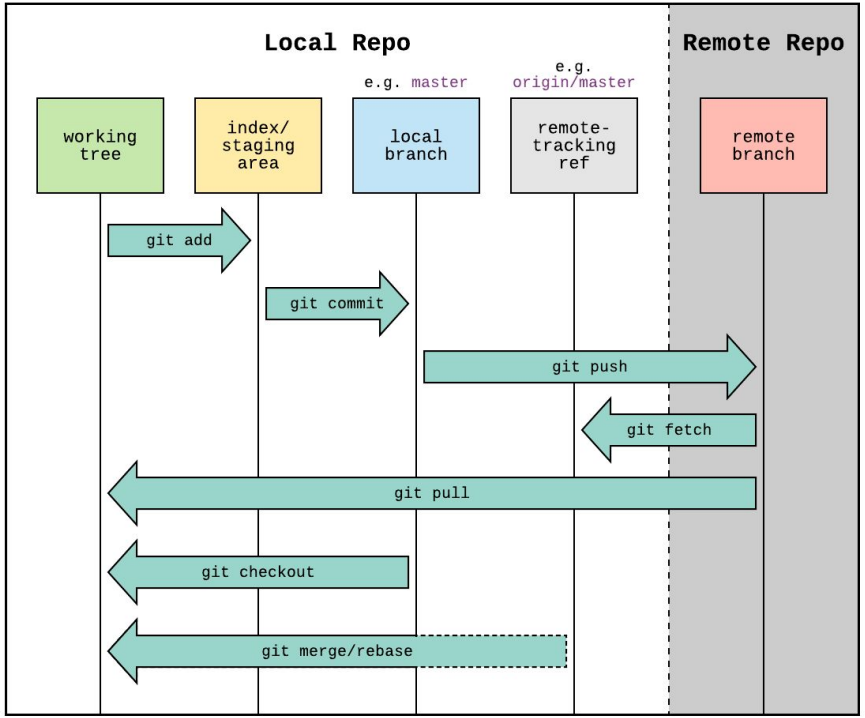


Centralized



Distributed

Different Stages In Git Workflow



GIT Workflow

- You modify files in your working tree.
- You selectively stage just those changes you want to be part of your next commit.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Any Questions??

States that file can reside in

- ***Untracked*** basically means that Git sees a file you didn't have in the previous snapshot (commit)
- ***Modified*** means that we have changed the file but have not committed it to our database yet.
- ***Staged*** means that we have marked a modified file in its current version to go into your next commit snapshot.
- ***Committed*** means that the data is safely stored in your local database.

```
~/git_workshop_2021/myFirstRepo main +2 !1 ?3 ❯ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file3.txt
        new file:   file5.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file5.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
        file4.txt
```

Configure user information for all local repositories

- There are config file for git that are applicable globally and also there are config file that are specific to single project repos.

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

$git init

- This command creates an empty Git repository. -
- a .git directory with subdirectories for objects, refs/heads, refs/tags, and template files.
- An initial HEAD file that references the master branch is also created.

```
~/git_workshop_2021 ❯ mkdir myFirstRepo

~/git_workshop_2021 ❯ ls
myFirstRepo

~/git_workshop_2021 ❯ cd myFirstRepo

~/git_workshop_2021/myFirstRepo ❯ git init
Initialized empty Git repository in /home/aayush57/git_workshop_2021/myFirstRepo/.git/

~/git_workshop_2021/myFirstRepo main ❯
```

$git status

Shows the working tree status

- Displays paths that have differences between the working area and the staging file
  - We need **`git add`** to add them to staging area
- Displays paths that have differences between the index file and the current HEAD commit,
  - we use **`git commit`** to commit or same them to local database

$git add <filepath>

This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit.

```
~/git_workshop_2021/myFirstRepo main ?3 ❯ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
        file4.txt

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

~/git_workshop_2021/myFirstRepo main ?3 ❯ git add file1.txt file2.txt

~/git_workshop_2021/myFirstRepo main +2 ?1 ❯ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4.txt


~/git_workshop_2021/myFirstRepo main +2 ?1 ❯
```

$git commit

- record staged changes to git repository
- commit object is created inside .git direcory

```
~/git_workshop_2021/myFirstRepo main +2 ?1 ❯ git commit file1.txt
[main f6eba4c] file1 added
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

~/git_workshop_2021/myFirstRepo main +1 ?1 ❯ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4.txt
```

$git diff

- Show changes between the working area and the staging area

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file2.txt
```

```
~/git_workshop_2021/myFirstRepo main +1 !1 ?1 ❯ git diff
diff --git a/file2.txt b/file2.txt
index 47c4cf4..a70e413 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,8 +1,8 @@
 When I started using Version Control in my projects, the concept was difficult
 to understand. I saw many people, including myself, running commands such as
+i added this line to the file
 git pull, git push and applying processes that I did not understand. Why did I
 need to both commit and push? Why did every new feature require a new branch?
-After working several months in a project using Git daily, the concept became
 very clear to me and I could finally understand the full potential for
 collaboration using Version Controlling, and more specifically, Git.
```

$git log

- shows a flattened log of history

```
~/git_workshop_2021/myFirstRepo main ❯ git log
commit 185476c637678556ebd87a50ff2dd6f8dae04336 (HEAD -> main)
Author: aayush <neupane0403@gmail.com>
Date:   Mon May 10 23:17:04 2021 +0545

    after death page added to table

commit 3353bbae8c2bb8467bc176a45d54068f9268eff5
Author: aayush <neupane0403@gmail.com>
Date:   Mon May 10 23:13:59 2021 +0545

    updated table of contents

commit d174e1211414f075870d6aee0cc452b118984744
Author: aayush <neupane0403@gmail.com>
Date:   Mon May 10 23:11:54 2021 +0545

    table of content initialized
```

$git log --all --graph --decorate

- visualizes history as a graph

```
~/git_workshop_2021/myFirstRepo main ❯ git log  --oneline --graph --all
* 185476c (HEAD -> main) after death page added to table
| * 0faa3ff (acknowledgement) acknowledge added
|/
* 3353bba updated table of contents
* d174e12 table of content initialized
```

$git config alias.[short_name] <command>

- shows a flattened log of history

```
~/git_workshop_2021/myFirstRepo main +1 !1 ?1 ❯ git config alias.st status

~/git_workshop_2021/myFirstRepo main +1 !1 ?1 ❯ git st
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4.txt


~/git_workshop_2021/myFirstRepo main +1 !1 ?1 ❯
```

.gitignore file

- the gitignore file is the file that tells git which files or folder to ignore in project

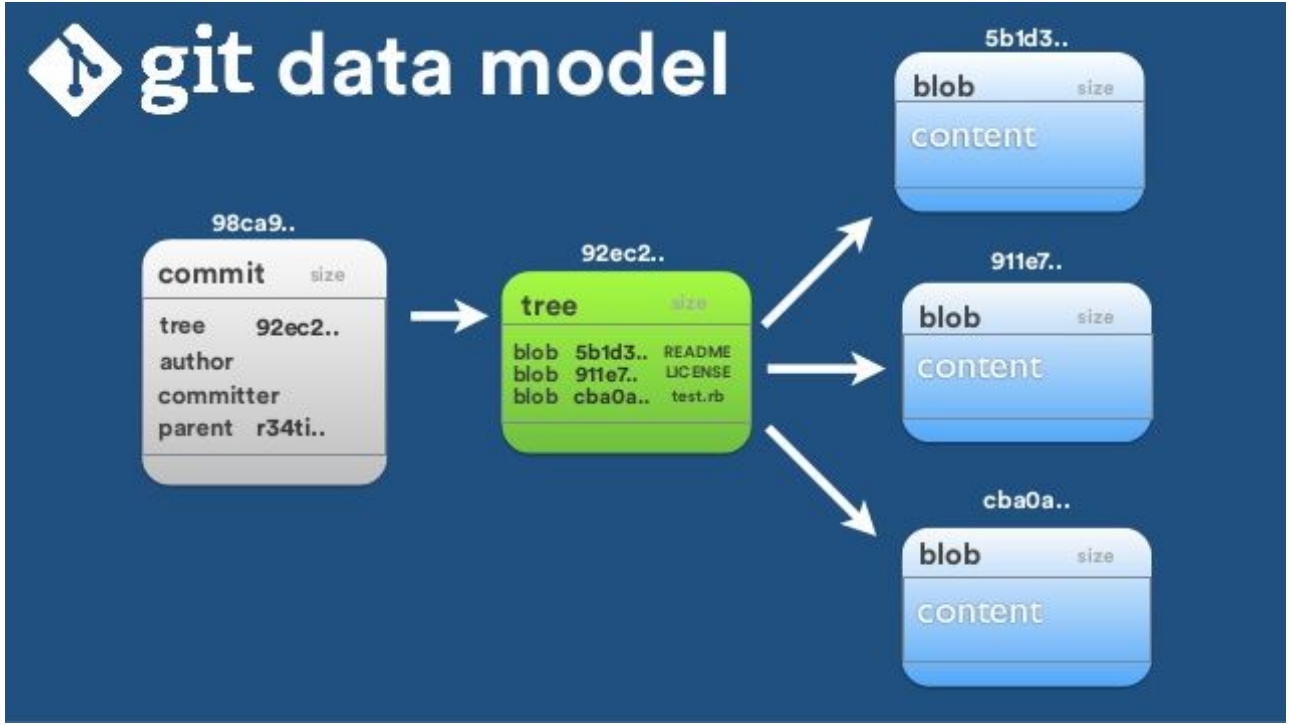
Open

*.gitignore
~/git_workshop_2021/myFirstRepo

```
1 bulids/
2 *.rough
3 *.html
```

Any Questions??

Git data model



Git data model

- A **“blob”** is used to store file data- it is generally a file.
- A **“tree”** is basically like a directory- it references a bunch of other trees and blobs (i.e. files and sub-directories).
- A **“commit”** object holds metadata for each change introduced in the repository, including the *author*, *committer*, *commit-data*, and *log-messages*.
- A **“tag”** object assigns an *arbitrary human-readable name* to a *specific object* usually a commit.

Git data model

```
~/git_workshop_2021/myFirstRepo death ❯ git cat-file -p 3353bbae8
tree 9dff0ed8d9810835d122b94b9e26e498ebb7e8a6
parent d174e1211414f075870d6aee0cc452b118984744
author aayush <neupane0403@gmail.com> 1620667739 +0545
committer aayush <neupane0403@gmail.com> 1620667739 +0545

updated table of contents

~/git_workshop_2021/myFirstRepo death ❯ git cat-file -p 9dff0ed
100644 blob 02571c22347fa8861614896e1fb4ea87a10c4e6c    table_of_content.txt

~/git_workshop_2021/myFirstRepo death ❯ git cat-file -p 02571c223
first chapter : age 0 to 9
second chapter : age 10 to 50
third chapter: age 51 to 100



~/git_workshop_2021/myFirstRepo death ❯
```

.git directory breakdown(major parts)

- When you run git init in a new or existing directory, Git creates the .git directory

```
~/git_workshop_2021/practice/.git main ❯ ls -F1
branches/
COMMIT_EDITMSG
config
description
HEAD
hooks/
index
info/
logs/
objects/
refs/
```

.git directory breakdown(major parts)

- The **objects** directory stores all the content for your database
- The **refs** directory stores pointers into commit objects in that data (branches, tags, remotes and more)
- **HEAD** file points to the branch you currently have checked out
- **Index** file is where Git stores your staging area information

Thank you