**DEVOPS**

The core part of Git is written in C, which ignores runtime overheads associated with other high-level languages.

Git was developed to work on the **Linux kernel**; therefore, it is capable enough to handle large repositories effectively. From the beginning, **speed and performance** have been Git's primary goals.

Supports **non-linear development**.

Git supports seamless branching and merging, which helps in visualizing and navigating a non-linear development. A **branch in Git represents a single commit**. We can construct the full branch structure with the help of its parental commit.

Branching and Merging

Branching and merging are the great features of Git, which makes it different from the other SCM tools. Git allows the creation of multiple branches without affecting each other.

We can perform tasks like creation, deletion, and merging on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

We can create a separate branch for a new module of the project, commit and delete it whenever we want.

We can have a production branch, which always has what goes into production and can be merged for testing in the test branch.

We can create a demo branch for the experiment and check if it is working. We can also remove it if needed.

The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.

Data Assurance

The Git data model ensures the cryptographic integrity of every unit of our project. It provides a unique commit ID to every commit through a SHA algorithm. We can retrieve and update the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.

Staging Area

The Staging area is also a unique functionality of Git. It can be considered as a **preview of our next commit**, moreover, an intermediate area where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.

We can perform tasks like creation, deletion,and merging on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

o We can create a separate branch for a new module of the project,

commit and delete it whenever we want.

o We can have a production branch, which always has what goes into

production and can be merged for testing in the test branch.

o We can create a demo branch for the experiment and check if it is

working. We can also remove it if needed.

o The core benefit of branching is if we want to push something to a

remote repository, we do not have to push all of our branches. We can

select a few of our branches, or all of them together.

Another feature of Git that makes it apart from other SCM tools is that it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.

o Maintain the clean history

Git facilitates with Git Rebase; It is one of the most helpful features of Git. It

fetches the latest commits from the master branch and puts our code on top of

that. Thus, it maintains a clean history of the project.

**MILESTONE QUESTIONS**

GitHub is an immense platform for code hosting. It supports version controlling and collaboration and allows developers to work together on projects. It offers both **distributed version control and source code management (SCM)** functionality of Git. It also facilitates collaboration features such as bug tracking, feature requests, task management for every project.

Essential components of the GitHub are:

a)Repositories

b)Branches

c)Commits

d)Pull Requests

e)Git (the version control tool GitHub is built on.

Advantages of GitHub

GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

1)It is easy to contribute to open source projects via GitHub.

2)It helps to create an excellent document.

3)You can attract the recruiter by showing off your work.

4)If you have a profile on GitHub, you will have a higher chance of being recruited.

5)It allows your work to get out there in front of the public.

6)You can track changes in your code across versions.

>GitHub is a place where programmers and designers work together.

>They collaborate, contribute, and fix bugs together.

>It hosts plenty of open source projects and codes of various programming languages.

>Some of its significant features are as follows.

1.Collaboration

2.Integrated issue and bug tracking

3.Graphical representation of branches

4.Git repositories hosting

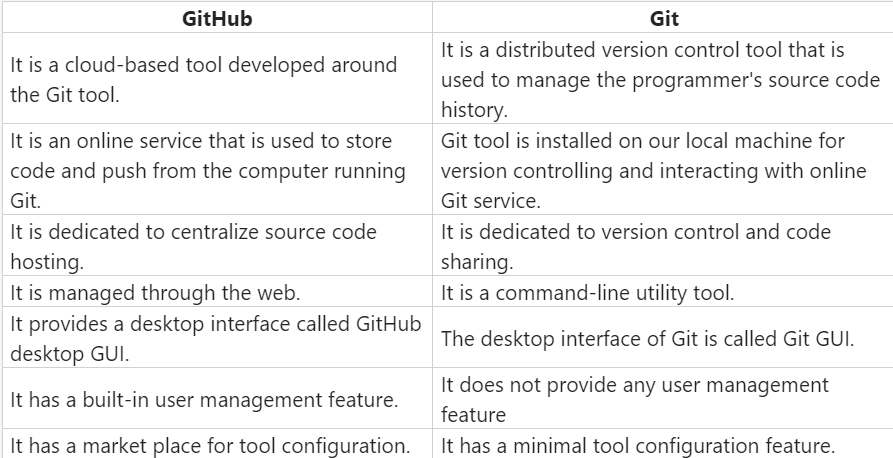
5.Project management

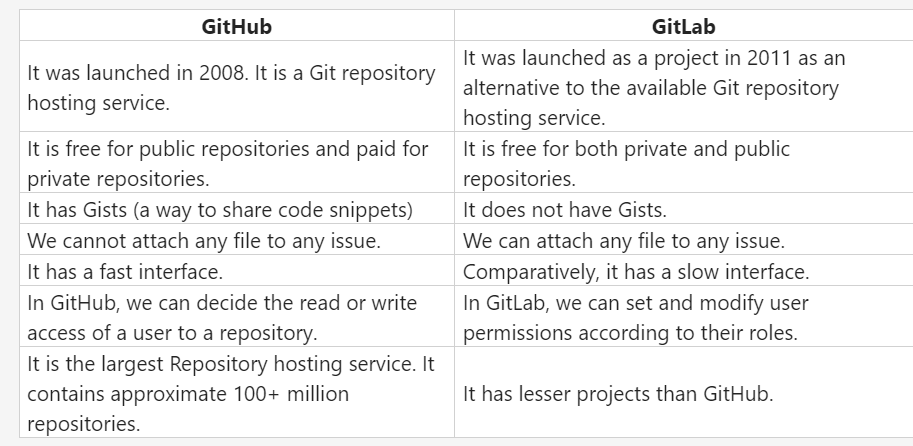
6.Team management

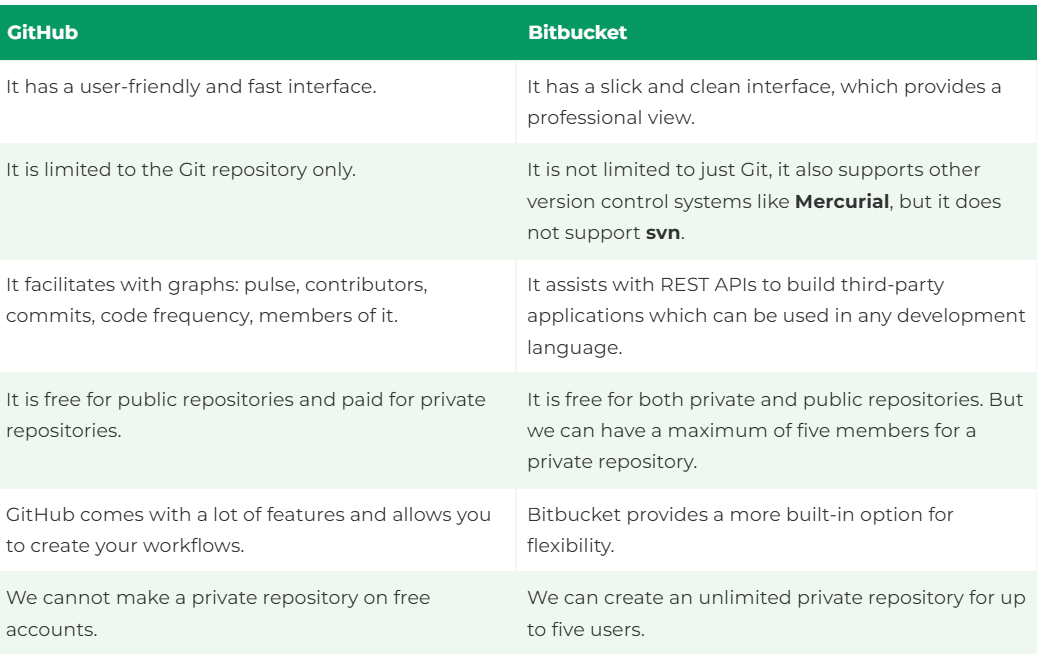
7.Code hosting

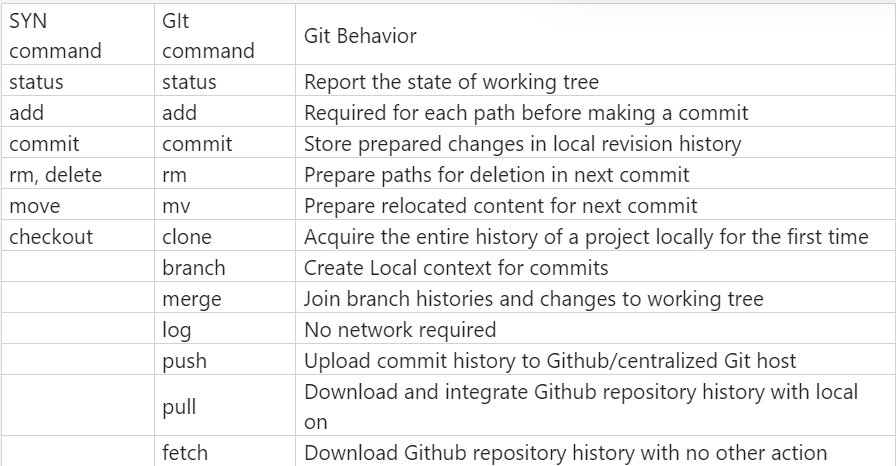
8.Track and assign tasks

9.Conversations









**Adding a New Subproject**

**Submodule**

git submodule add <https://github.com/githubraining/example-git>  
  
git commit -m "adding new submodule"

->The submodule add command adds a new file called gitmodules along with a subdirectory containing the files from example-submodule. Both are added to your index (staging area) and you simply need to commit them. The submodule’s history remains independent of the parent project.

**Subtree**

git subtree add --prefix=example-submodule <https://github.com>

->The subtree command adds a subdirectory containing the files from example-submodule. The most common practice is to use the —squash option to combine the subproject’s history into a single commit, which is then grafted onto the existing tree of the parent project. You can omit the —squash option to maintain all of the history from the designated branch of the subproject  
  
<https://github.com/githubtraining/example-submodule> main --squash

Viewing a Diff of the Subproject

To view a diff of the submodule:

# show changes to the submodule commit

git diff example-submodule

#show online log of new commits in the submodule

git diff --submodule example-submodule

#show changes to the files in the submodule

git diff --submodule=diff

## **Cloning a Repository with a subproject**

### Submodule

->To clone a repository along with its submodules:

git clone --recurse-submodules URL

->If you forgot —recurse-submodules. you can clone and initialize all submodules:

git submodule update --init --recursive

->Adding —recursive is only required if any submodule itself has submodules.

## **Pulling in Superproject Updates**

### Submodule

->By default, the submodule repository is fetched, but no updated when you run git pull in the superproject. You need to use git submodule update, or add the —recurse-submodules flag to pull:

git pull

git submodule update --init --recursive

# or , in one step (Git >= 2.14)

git pull --recurse-submodules

->—init is required if the super project added new submodules, and —recursive is needed if any submodule itself has submodules.

->If ever the superproject changes the URL of the submodule, a separate command is required.

# copy the new URL to your local config

git submodule sync --recursive

# update the submodule from the new URL

git submodule update --init --recursive

-> —recursive is only needed if any submodule itself has submodules.

## **Changing Branches**

### Submodule

->By default, the submodule working tree is not updated to match the commit recorded in the superproject when changing branches.

->You need to use git submodule update, or add the —recursive-submodules flag to switch

git swith <branch>

git submodule update --recursive

# or , in one step (Git >= 2.14)

git switch --recurse-submodules <branch>

## Pulling in subproject Update

### Submodule

# Update the submodule repository

git submodule update --remote

# Record the changes in the superproject

git commit -am "Update Submodule"

* If you have more than one submodule, you can add the path to the submodule at the end of the git submodule update —remote command to specify which subproject to update.
* By default, git submodule update —remote will update the submodule to the latest commit on the main branch of the submodule remote.
* You can change the default branch for further calls with:

# Git >= 2.22

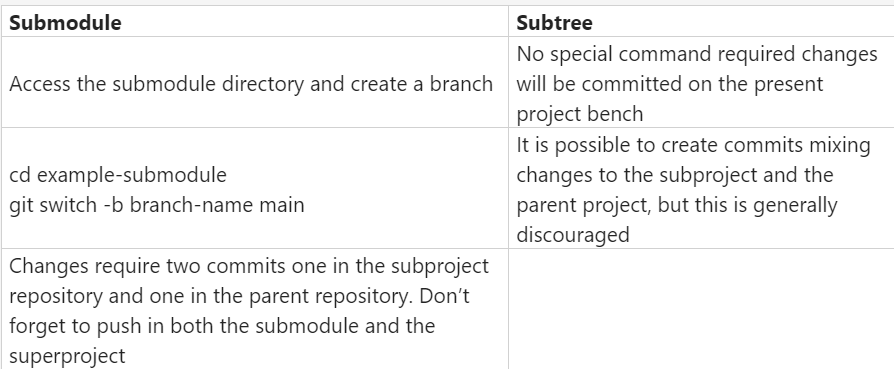
git submodule set-branch other-branch

# or

git config -f .gitmodules submodule.example-submodule.branch

## **Making Changes to a Subproject**

->In most cases, it is considered best practice to make changes in a separate clone of the subproject repository and pull them into the parent project. When this is not practical, follow these instructions.



# **Pushing Changes to the Subproject Repository**

### ****Submodule****

While in the submodule directory:

git push

git push --recurse-submodules=on-demand

**Subtree**

git subtree push --prefix=example-submodule <https://github.com/githubtraining>

# **Helpful Configs for Submodules**

Always show the submodule log when you diff:

git config --global diff.submodule log

Show a short summary of submodule changes in your git status message:

git config --global status.submoduleSummary true

Make push default to --recurse-submodules=on-demand:

git config --global push.recurseSubmodules on-demand

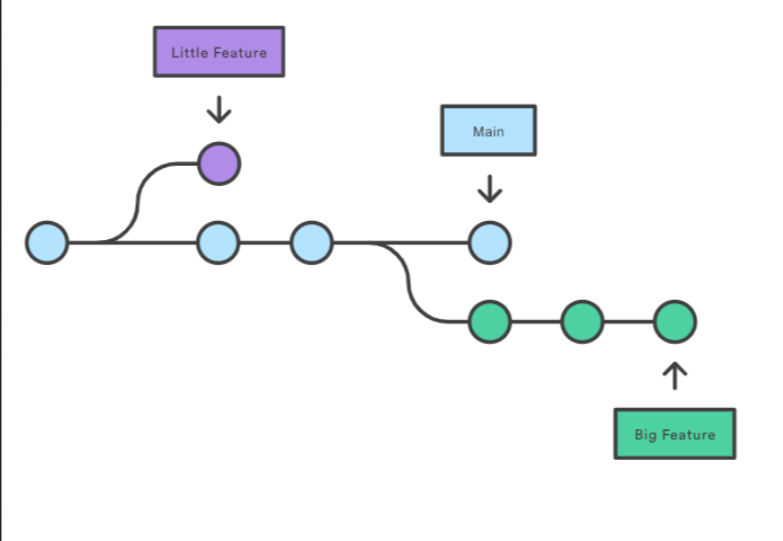
Make all commands (except clone) default to --recurse-submodules if they support the flag (this works for git pull since Git 2.15):

git config --global submodule.recurse true

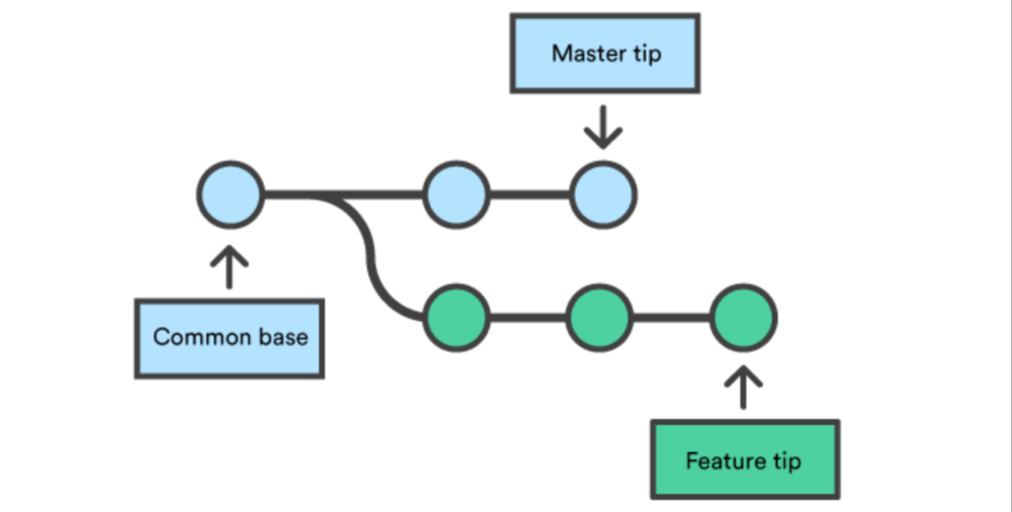
**Advanced Git Operations**

Branches

A branch is a separate version of a repo with its own history –it is not a separate copy; it exists within the repo where it was created.

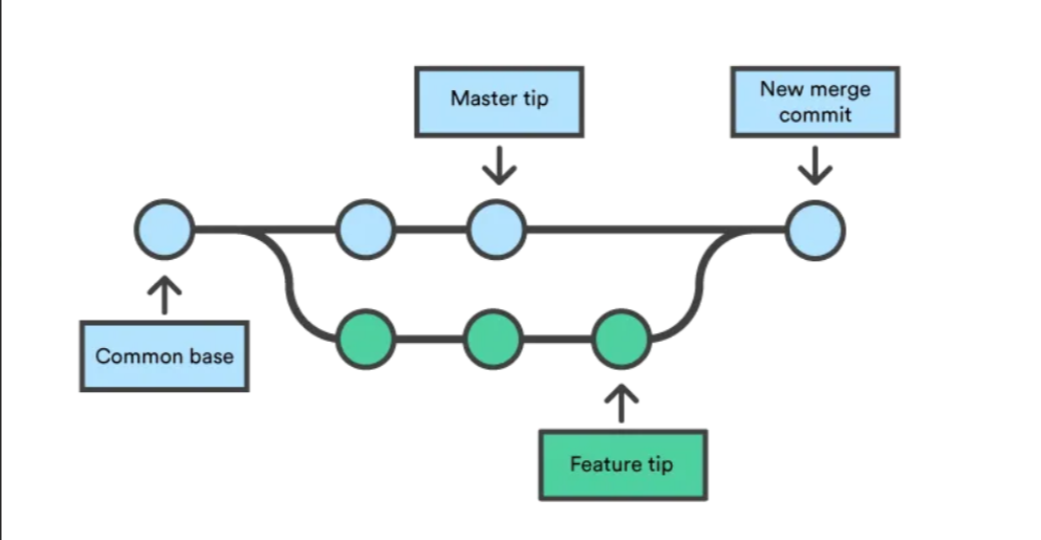


Two branches created from the main branch, each with their own commit history.



The main and feature branch have a common base but different tips(current states).

A merge commit joins the feature branch back to the main branch.



**Use cases of branches**

* Making a collaborative workflow smoother.
* Making pull request.
* Testing code in software and web development.

**Branch workflow**

* Create a new branch locally.
* switch to the new branch locally.
* Push the local branch to remote repo.
* Make some changes, stage, commit, and push (repeat as needed).
* Merge the branch back into the Main branch.

**Forking**

* To fork a repo is to copy a repo from someone else’s account into your account so that you can start your own project based on the existing repo, without having to push changes to the original’s repo. In git jargon, a fork is just a “clone with a different remote origin”.
* If you fork a repo, it is probably going to be one where you aren’t collaborating directly with its creator. The creator may continue to make updates, adding commits to their version's history. While you do the same. A good example would be if you are making a GitHub pages, and you want to use a premade template. You would fork the page template repo and add your own content; you would likely never push changes to the template itself.
* However, if you are working on a project that you forked from someone else’s repo, and you do want them to incorporate your changes into their version of the repo, you would make a pull request.

**Forking a Repo**

To fork a repo on GitHub, just go to the repo page and click “fork” in the upper right-hand corner.

**Pull Request**

* A pull request (PR for short) is just that a request that the owner of a repo pull your changes, incorporating them into his or her repo.
* You can make a pull request to any public repo on GitHub, even if your don’t have push access to the repo. In contrast, if you are a collaborator on a repo who has write access, you can push commits without making pull requests. However, it is often a good idea to make a pull request anyway, instead of pushing directly. This is a more respectful way to contribute to a project because it gives the repo owner a chance to review your changes and approve them.

**Basic pull request workflow**

* The “pull requester” forks someone else’s repository and clones it locally.
* S/he makes some changes, stages, commits, and pushes.
* On GitHub, s/he creates a pull request comparing the changes you just made with original owner’s repo.
* The original owner creates a new branch and pulls the PR into that branch.
* The owner tests the code, potentially adding additional changes and committing them.
* If the code is good, the owner accepts the PR by merging it into his or her main branch and pushing.

**GitHub Issues**

Issues are a useful feature of GitHub.com that can greatly improve your workflow. The user interface is fairly easy to figure out. This is just a plug to encourage to make them a part of their workflow.

**Uses For Issues**

* Pointing out a bug or feature request in someone else’s repository.
* Making a to-do list for yourself.
* Managing projects by assigning tasks to collaborate.
* Maintaining a record a rich information about your work
* Troubleshooting problems with code by searching existing issues.

**Formatting issues**

* You can reference users, commits, pull requests and other issues in the text of your issue, and they will automatically be linked.
* To reference users, use the @ sign followed by their name.
* To reference commits, use the first 7 digits of the commit’s hash(it will look something like 5d1001b).
* To reference pull requests and other issues, use the # sign followed by the number ID of the pull request of issue. They go up sequentially starting at #1.

**Issue Labels and assigns**

* You can put labels on issues to tag them by topic.
* You can also assign issues to collaborators on your project, to designate who is responsible for resolving the issue.
* Example: issues assigned to different collaborators.

**Closing resolved issues**

Once an issue is resolved, you can close it. It will no longer appear as an outstanding issue, but all closed issues are still archived as part of the repo on GitHub.

**Troubleshooting with issues**

If you run into an issue with an R package or other software. It is often helpful to go to the repo page for that software and search the (open and closed) issues there. This can be a more targeted way to find a solution for your problem, compared to searching Google or StackOverFlow.

**Merge Conflicts and how to Handle them in GIT**

Merge conflicts are a common challenge developers face when working with Git. Understanding what they are and how to resolve them effectively is important for smooth collaboration in any project.

Understanding how to handle merge conflicts is important for maintaining a smooth workflow. This article will explain what merge conflicts are, why they happen and how to resolve them effectively.

**What are Merge Conflicts?**

A merge conflict happens when Git is unable to automatically reconcile differences in code between two commits. This typically occurs during a merge operation, where changes from different branches are combined.

**Why Do Merge Conflicts Occur?**

1. **Simultaneous Edits:** Two developers modify the same line of code in different branches.
2. **Conflicting changes:** A file is deleted in one branch and modified in another.
3. **Complex Merges:** When multiple branches are being merged, with changes scattered across various files and lines.

**Types of Merge Conflicts**

**While starting the merge:** If there are changes in either the working directory or staging area, while merging, then Git will fall to start the merge. This happens because the pending changes could be overridden by the commits that are being merged. This is the error message provided by Git when this type of merge conflict happens:

* Error: Entry ‘<fileName>’ not uptodate. Cannot merge. (changes in working directory) or error: Entry ‘<fileName>’ would be overwritten by merge. Cannot merge. (Changes in staging area)

This type of conflict can be resolved either by doing **git stash save “any message to describe what is saved”** (Stashes away any changes in your staging area and working directory in a separate index) or **git checkout <file\_name> (throws out your changes),** and then the merge can be completed.

**During the merge**: This occurs because you have committed changes that are in conflict with someone else’s committed changes. Git will do its best to merge the files and will leave things for you to resolve manually in the files it lists. This is the error message provided by Git when this type of merge conflict happens:

**CONFLICT (content): Merge conflict in <fileName> Automatic merge failed; fix conflicts and then commit the result.**

**Creating a Merge Conflict**

To show a simple example of how a merge conflict can happen, we can manually trigger a merge conflict from the following set of commands in any UNIX terminal / GIT bash:

**Step 1:** Create a new directory using the **mkdir** command, and **cd** into it.

**Step 2:** initialize it as a new Git repository using the **git init** command and create a new text file using the **touch** command.

**Step 3:** Open the text file and add some content in it, then **add** the text file to the repo and **commit** it.

**Step 4**: Now, it's time to create a new branch to use it as the conflicting merge. Use **git checkout** create and checkout the new branch.

**Step 5**: Now overwrite some conflicting changes to the text file from this new branch.

**Step 6:** Add the changes to git and commit it from the new branch.

With this new branch: new\_branch\_for\_merge\_conflict we have created a commit that overrides the content of test\_file.txt.

**Step 7**: Again, checkout the master branch, and this time append some text to the test\_file.txt from the master branch.

**Step 8**: add these new changes to the staging area and commit them.

**Step 9**: Now for the last part, try merging the new branch to the master branch and you will encounter the second type of merge conflict.

**Handling the Merge Conflict**

* As we have experienced from the proceeding example, Git will produce some descriptive output letting us know that a CONFLICT has occurred. We can gain further insight by running the git status command. This is what we will get after running the git status command.
* On branch master

you have unmerged paths.

(fix conflicts and run “git commit”)

(use “git merge —abort” to abort the merge)

* Unmerged paths

(use “git add <file>…” to mark resolution)

no changes added to commit (use “git add” and/or “git commit -a”)

* On operating the test\_file.txt we see some “conflict dividers”. This is the content of our test\_file.txt:

<<<<<<<<<<<<<HEAD

Adding some content to mess with it later

Append this text to initial commit

============

Changing these contents of text file from new branch

new branch for merge conflict

* The ====== line is the “center” of the conflict. All the content between the center and the <<<<<<<<<< HEADLINE is content that exists in the current branch master which the HEAD ref is pointing to. Alternately, all content between the center and >>>>>>>> new branch for merge conflict is content that is present in our merging branch.
* To resolve our merge conflict, we can manually remove the unnecessary part from any one of the branches and only consider the content of the branch that is important for further use, along with removing the “conflict dividers” from our files. Once the conflict has been resolved we can use the git add command to move the new changes to the staging area, and then git commit to commit the changes.

**Database Fundamentals**

Database Management system or DBMS in short refers to the technology of storing and retrieving user’s data with utmost efficiency along with appropriate security measures.

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiency. It is also used to organise the data in the form of a table, schema, views, and reports etc.

For example: The college database organizes the data about the admin staff, students, and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.

* Database management system is a software which is used which is used to manage the database. For Example: MySQL, Oracle, etc. Are a very popular commercial database which is used in different applications.
* DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
* It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

**DBMS allows the users the following tasks**:

1.Data Definition: It is used for Creation, Modification, and removal of definition that defines the organization of data in the database.

2. Data Updation: It is used for the insertion, modification, and deletion of the actual data in the database.

3.Data Retrieval: It is used to retrieve the data from the database which can be used by applications for various purposes.

4. User Administration: It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

**Characteristics of DBMS**

1. It uses a digital repository established on a server to store and manage the information.
2. It can provide a clear and logical view of the process that manipulates data.
3. DBMS contains automatic backup and recovery procedures.
4. It contains ACID properties which maintain data in a healthy state in case of failure.
5. It can reduce the complex relationship between data.
6. It is used to support manipulation and processing of data.
7. It is used to provide security of data.
8. It can view the database from different viewpoints according to the requirements of the user.

**Advantages of DBMS**

1. **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

2. **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.

3. **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system

4. **Reduce time:** It reduces development time and maintenance need.

5. **Backup**: It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.

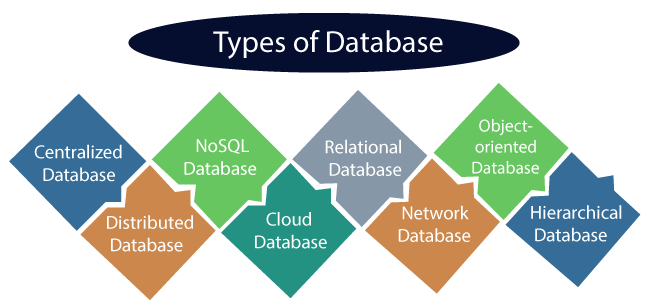
6.**multiple user interface**: It provides different types of user interface like graphical user interfaces, application program interfaces.

**Disadvantages of DBMS**

* **Cost of hardware and software:** It requires a high speed of data processor and large memory size to run DBMS software.
* **Size:** It occupies a large space of disks and large memory to run them efficiently
* **Complexity:** Database system creates additional complexity and requirements
* **Higher Impact of failure:** Failure is a highly impacted the database because in most of the organisation, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

**Concept of Database**

* **Database Schema:** It is a design of the database. Or we can say that it is a skeleton of the database that is used to represent the structure, types of data will be stored in the rows and columns, constraints, relationships between the tables.
* **Data constraints:** In a database, sometimes we put some restrictions on the table that what type of data can be stored in one or more columns of the table, it can be done by using constraints. Constraints are defined while we are creating a table.
* **Data Dictionary or Metadata:** Metadata is known as the data about the data. Or we can say that the database schema along with different types of constraints on the data is stored by DBMS in the dictionary is known as metadata.
* **Database instance:** In a database, a database instance is used to define the complete database environment and its components. Or we can say that it is a set of memory structures and background processes that are used to access the database files.
* **Query:** In a database, a query is used to access data from the database. So, users have to write queries to retrieve or manipulate data from the database.
* **Data manipulation:** In a database, we can easily manipulate data using the three main operations that is insertions, deletion and Updation.
* **Data Engine:** It is an underlying component that is used to create and manage various database queries.



**1)Centralized Database**

It is the type of database that stores data at a centralized database system. It comforts the users to access the stored data from different locations through several applications. These applications contain the authentication process to let users access data security. An example of a Centralized Database can be Central Library that carries a central database of each library in a college.

**Advantages of Centralized Database**

* It has decreased the risk of data management, ie, manipulation of data will not affect the core data.
* Data consistency is maintained as it manages data in a central repository.
* It provides better data quality, which enables organizations to establish data standards.
* It is less costly because fewer vendors are required to handle the data sets.

**Disadvantages of Centralized Database**

* The size of the centralized database is large, which increases the response time for fetching the data.
* It is not easy to update such an extensive database system.
* If any server failure occurs, entire data will lose, which could be a huge loss.

**2)Distributed Database**

* Unlike a centralized database system, in distributed systems, data is distributed systems, data is distributed among different database system of an organization.
* These database systems are connected via communication links, such links help the end-users to access the data easily.
* Examples of the Distributed database are Apache Cassandra, HBase, Ignite, etc.

**3)Relational Database**

* This database is based on the relational data mode, which stores data in the form of rows(tuple) and columns(attributes) and together forms a table(relation).
* A relational database uses SQL for storing, manipulating, as well as maintaining the data. E.F. Codd invented the database in 1970.
* Each table in the database carries a key that makes the data unique from others.
* Examples of Relational databases are MySQL, Microsoft SQL Server, Oracles, etc.
* There are following four commonly known properties of a relational model known as ACID properties, were:
  + A means Atomicity
  + C means Consistency
  + I means Isolation
  + D means Durability

**4)NoSQL Database**

* Non-SQL/Not only SQL is a type of database that is used for storing a wide range of data sets.
* It is not a relational database as it stores data not only in tabular form but in several different ways.
* It came into existence when the demand for building modern applications increased.
* Thus, NoSQL presented a wide variety of database technologies in response to the demands.
* We can further divide a NoSQL database into the following four types:
  + Key-values storage.
  + Document-oriented Database.
  + Graph Database.
  + Wide-column stores.

**5)Cloud Database**

A type of database where data is stored in a virtual environment and executes over the cloud computing platform. It provides users with various cloud computing services (SaaS, PaaS, IaaS, etc.) for accessing the database. There are numerous cloud platforms, but the best options are:

* Amazon Web Services(AWS)
* Microsoft Azure
* Kamatera
* PhonixNAP
* ScienceSoft
* Google Cloud SQL, etc.

**6)Object -Oriented Databases**

The type of database that uses the object-based data model approach for storing data in the database system. The data is represented and stored as objects which are similar to the objects used in the object-oriented programming language.

**7)Hierarchical Databases**

It is the type of database that stores data in the form of parent-children relationship nodes. Here, it organizes data in a tree-like structure.

Data get stored in the form of records that are connected via links. Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records.



**8)Network Databases**

It is the database that typically follows the network data model. Here, the representation of data is in the form of nodes connected via links between them. Unlike the hierarchical database, it allows each record to have multiple children and parent nodes to form a generalized graph structure.

**9)Personal Database**

Collecting and storing data on the user's system defines a Personal Database. This database is basically designed for a single user.

### Advantage of Personal Database

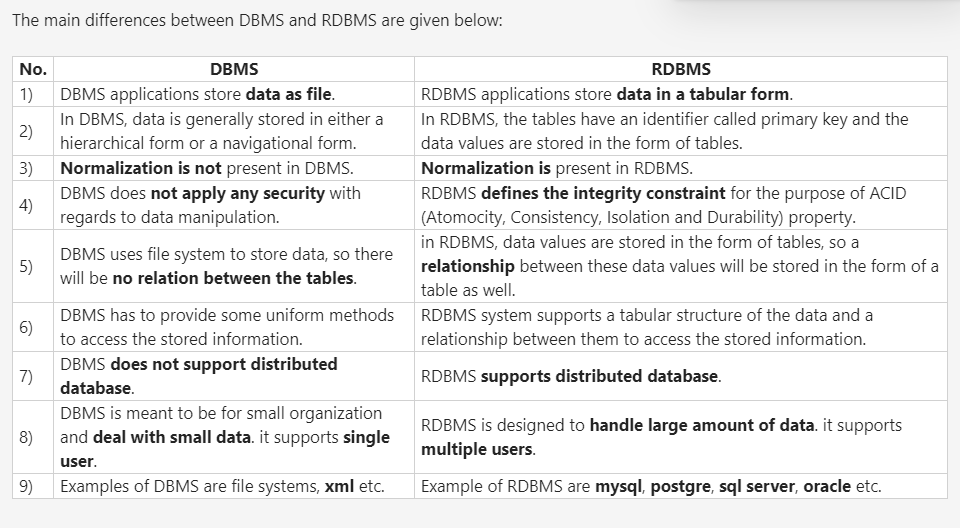
* It is simple and easy to handle.
* It occupies less storage space as it is small in size.

**10)Operational Database**

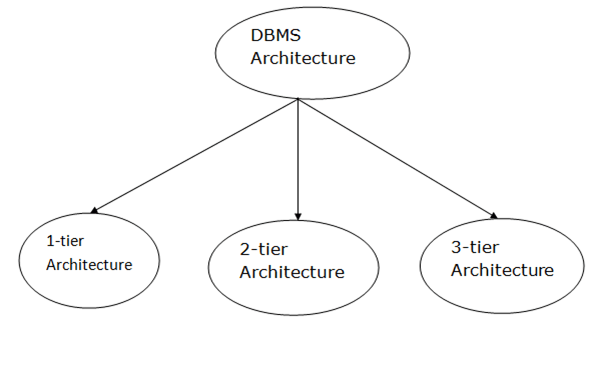
The type of database which creates and updates the database in real-time. It is basically designed for executing and handling the daily data operations in several businesses. For example, An organization uses operational databases for managing per day transactions.

**11)Enterprise Database**

Large organizations or enterprises use this database for managing a massive amount of data. It helps organizations to increase and improve their efficiency. Such a database allows simultaneous access to users.



**Types of DBMS Architecture**

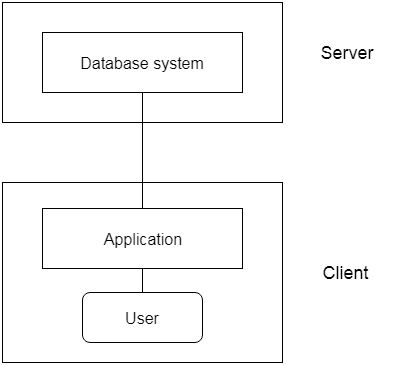


**1-Tier Architecture**

* In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
* Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
* The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

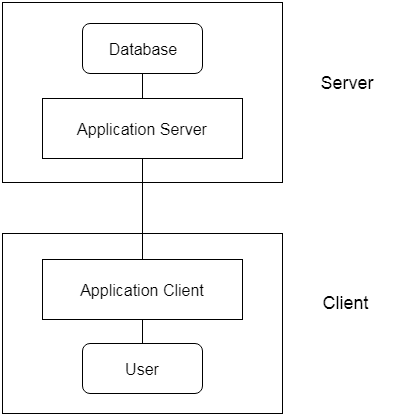
**2-Tier Architecture**

* The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
* The user interfaces and application programs are run on the client-side.
* The server side is responsible to provide the functionalities like: query processing and transaction management.
* To communicate with the DBMS, client-side application establishes a connection with the server side.

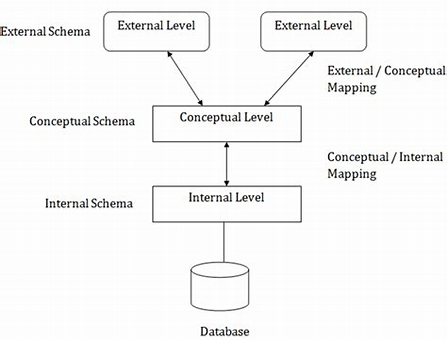


**3-Tier Architecture**

* The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
* The application on the client-end interacts with an application server which further communicates with the database system.
* End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
* The 3-Tier architecture is used in case of large web application.



The three-schema architecture is as follows:



**Normalization**

This is the process of organizing a database to minimize redundancy and dependency by breaking down complex tables into smaller, more manageable ones. It’s important to understand normalization because it helps you create efficient and scalable databases, reduces data inconsistency and duplication, and makes it easier to update and maintain the database over time. This information is often skipped over in introductory material because it can be technical and complex, but it is crucial for understanding how to properly design and maintain a database.

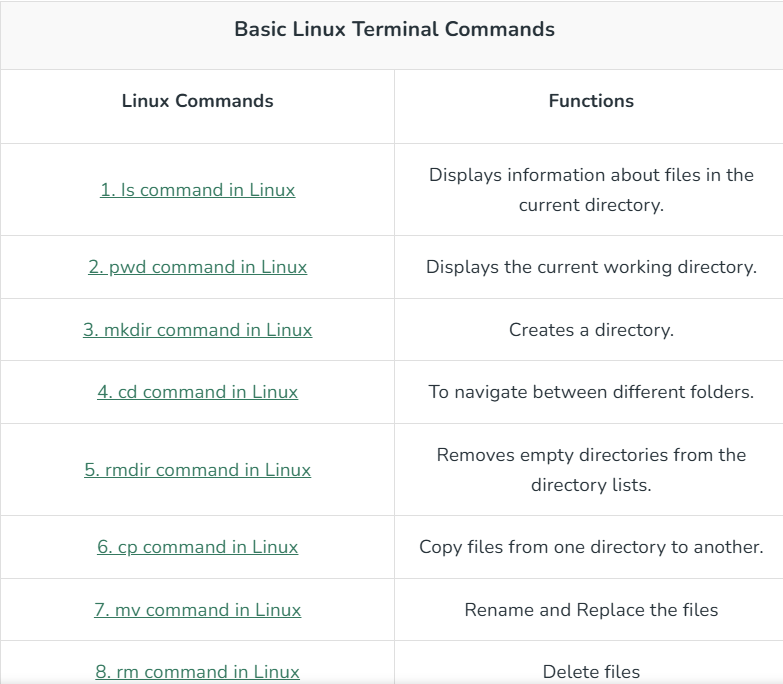
**Objectives of Three Schema Architecture**

The main objective of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once. Thus it separates the user's view from the physical structure of the database. This separation is desirable for the following reasons:

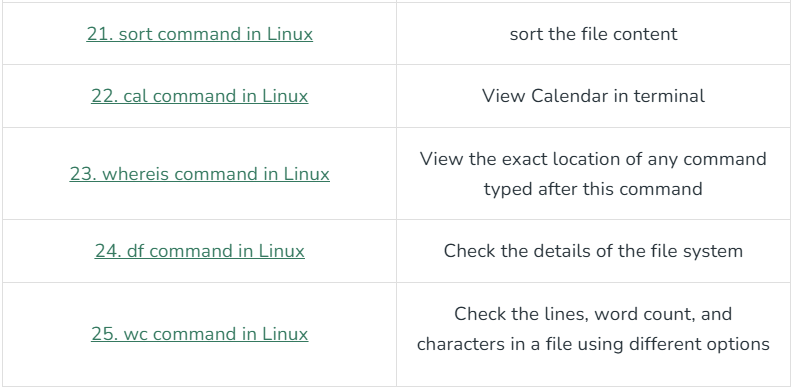
* Different users need different views of the same data.
* The approach in which a particular user needs to see the data may change over time.
* The users of the database should not worry about the physical implementation and internal workings of the database such as data compression and encryption techniques, hashing, optimization of the internal structures etc.
* All users should be able to access the same data according to their requirements.
* DBA should be able to change the conceptual structure of the database without affecting the users.
* Internal structure of the database should be unaffected by changes to physical aspects of the storage.

**LINUX**

* Linux is a widely used open-source operating system, similar to Windows, Mac and Android.
* It shares similarities with Unix, another operating system known for its commercial use.
* Unix and Linux have comparable components, including the kernel, shell and programs.
* Many commands in Unix and Linux exhibit similar behaviour and syntax.
* Learning the basics of Linux helps you understand how this powerful operating system works, why it’s widely used in various fields, and how it differs from Windows with its open-source nature, better security and flexibility in customization.



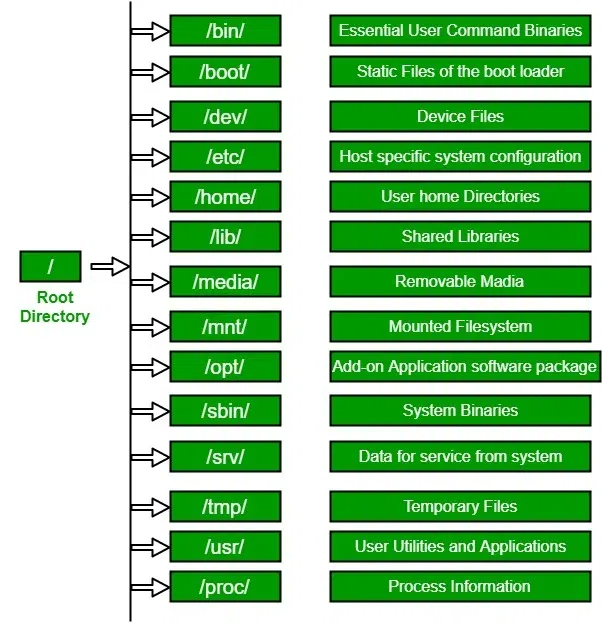




**LINUX File Hierarchy Structure**

The Linux File Hierarchy Structure or the Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems. It is maintained by the Linux Foundation.

* In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.
* Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.
* Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS and are not considered authoritative for platforms other than Linux.



1. /(Root):

Primary hierarchy root and root directory of the entire file system hierarchy

* Every single file and directory start from the root directory.
* The only root user had the right to write under this directory.
* /root is the root users home directory, which is not the same as /

2. /bin:

Essential command binaries that need to be available in single user mode for all users e.g.cat, ls, cp

* Contains binary executables.
* Common Linux commands you need to use in single user modes are located under this directory.
* Commands used by all the users of the system are located here e.g.ps, ls, ping, grep, cp.

3. /boot:

* Boot loader files, e.g., kernels, initrd
* Kernel initrd, vmlinux. grub files are located under /boot
* Example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic.

4. /dev:

* Essential device files, e.g., /dev/null
* These include terminal devices, usb or any device attached to the system.
* Example: /dev/tty1, /dev/usbmon0

5. /etc:

* Host specific system-wide configuration files.
* Contains configuration files required by all programs.
* This also contains startup and shutdown shell scripts used to start/stop individual programs.
* Example: /etc/resolv.conf

6. /home:

* Users home directories containing saved files, personal settings, etc.
* Home directories for all used to store their personal files
* example: /home/bhargav

7. /lib:

* Libraries essential for the binaries in /bin/ and /sbin/.
* Library filenames are either id\* or lib\*.so.\*
* Example:Id-2.11.1.so, libncurses.so.5.7

8. /media:

* Mount points for removable media such as CD-ROMs (appeared in FHS-2.3)
* Temporary mount directory for removable devices.
* Examples, /media/cdrecorder for CD Writer

9. /mnt:

* Temporarily mounted filesystems.
* Temporary mount directory where sysadmins can mount filesystems.

10. /opt:

Optional application software packages.

* Contains add-on applications from individual vendors.
* Add-on applications should be installed under either /opt/ or /opt/ sub-directory.

**11. /sbin:**

Essential system binaries, e.g., fsck, init, route.

* Just like /bin, /sbin also contains binary executables.
* The linux commands located under this directory are used typically by system administrators, for system maintenance purposes.
* Example: iptables, reboot, fdisk, ifconfig, swapon

**12. /srv:**

Site-specific data served by this system, such as data and scripts for web servers, data offered by FTP servers, and repositories for version control systems.

* srv stands for service.
* Contains server specific services related data.
* Example, /srv/cvs contains CVS related data.

**13. /tmp:**

Temporary files. Often not preserved between system reboots and may be severely size restricted.

* Directory that contains temporary files created by system and users.
* Files under this directory are deleted when the system is rebooted.

**14. /usr:**

Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications.

* Contains binaries, libraries, documentation, and source-code for second level programs.
* /usr/bin contains binary files for user programs. If you can’t find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
* /usr/sbin contains binary files for system administrators. If you can’t find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
* /usr/lib contains libraries for /usr/bin and /usr/sbin
* /usr/local contains user’s programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2
* /usr/src holds the Linux kernel sources, header-files and documentation.

**15. /proc:**

Virtual filesystem providing process and kernel information as files. In Linux, it corresponds to a procs mount. Generally, automatically generated and populated by the system, on the fly.

* Contains information about system process.
* This is a pseudo filesystem that contains information about running processes. For example: /proc/{pid} directory contains information about the process with that particular pid.
* This is a virtual filesystem with text information about system resources. For example: /proc/uptime.

**LINUX Directory Structure**

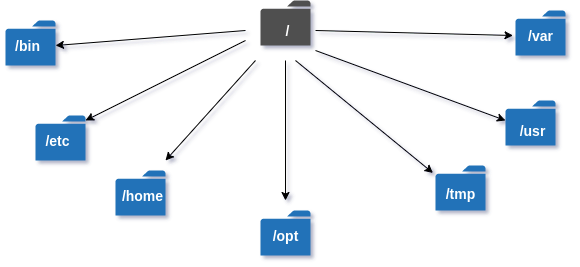
In Linux/Unix operating system everything is a file even directories are files, files are files, and devices like mouse, keyboard, printer, etc are also files. Here we are going to see the Directory Structure in Linux.

**Types of Files in the Linux System.**

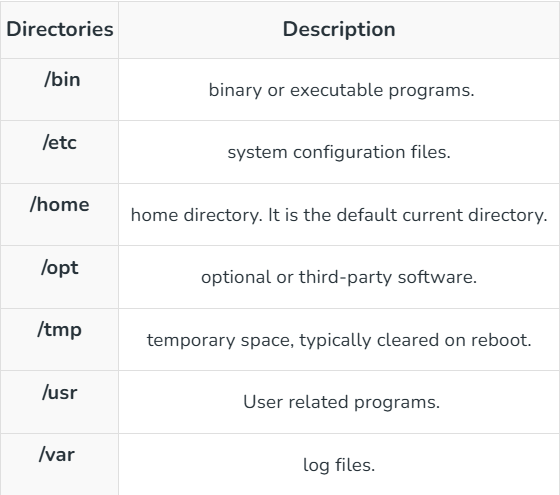
1. **General Files** – It is also called ordinary files. It may be an image, video, program, or simple text file. These types of files can be in ASCII or Binary format. It is the most commonly used file in the Linux system.
2. **Directory Files** – These types of files are a warehouse for other file types. It may be a directory file within a directory (subdirectory).
3. **Device Files –** In a Windows-like operating system, devices like CD-ROM, and hard drives are represented as drive letters like F: G: H whereas in the Linux system devices are represented as files. As for example, /dev/sda1, /dev/sda2, and so on.

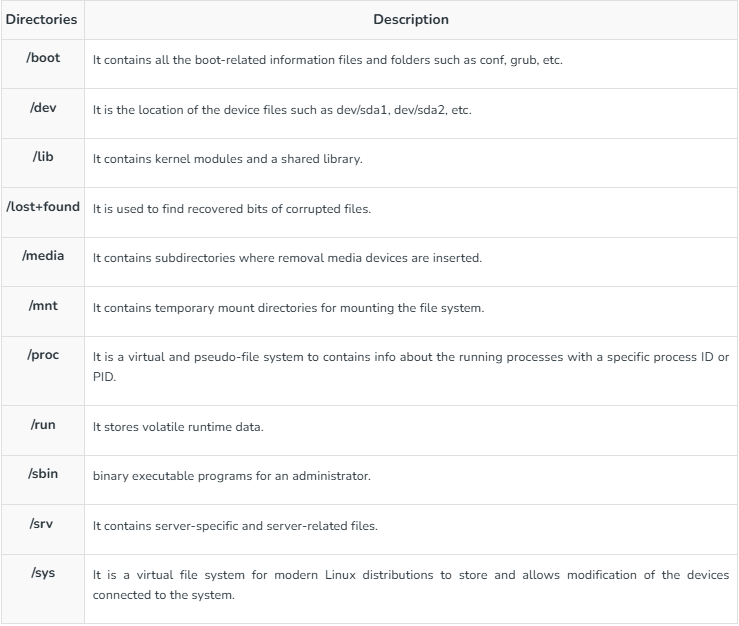
We know that in a Windows-like operating system, files are stored in different folders on different data drives like C: D: E: whereas in the Linux/Unix operating system files are stored in a tree-like structure starting with the root directory as shown in the below diagram.

The Linux/Unix file system hierarchy base begins at the root and everything starts with the root directory.



**These are the common top-level directories associated with the root directory:**

**Some other directories in the Linux system:** 

****

**Exploring directories and their usability:**

We know that Linux is a very complex system that requires an efficient way to start, stop, maintain and reboot a system, unlike Windows operating system. In the Linux system some well-defined configuration files, binaries, main pages information files are available for every process.

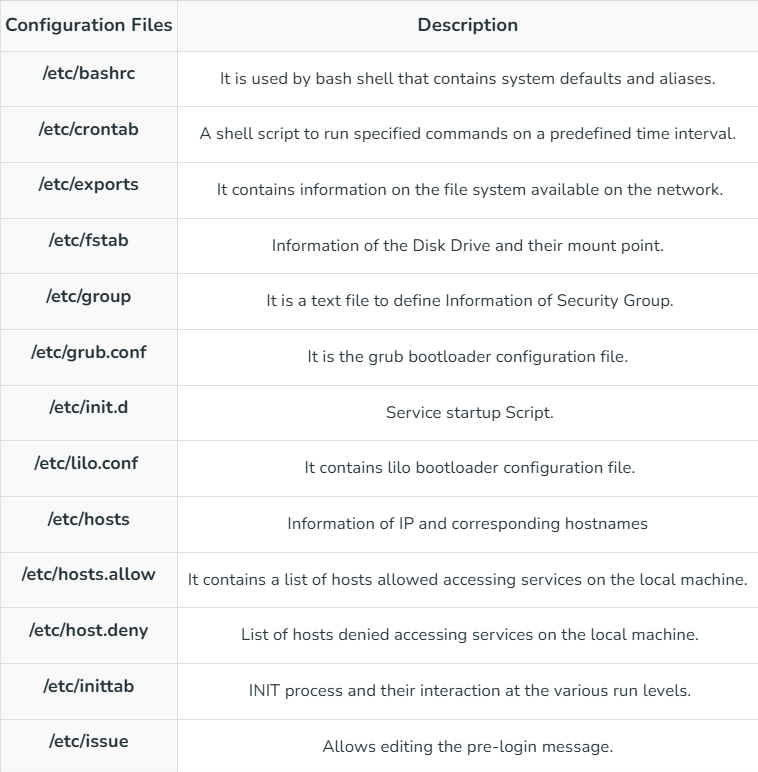
Linux Kernel File:

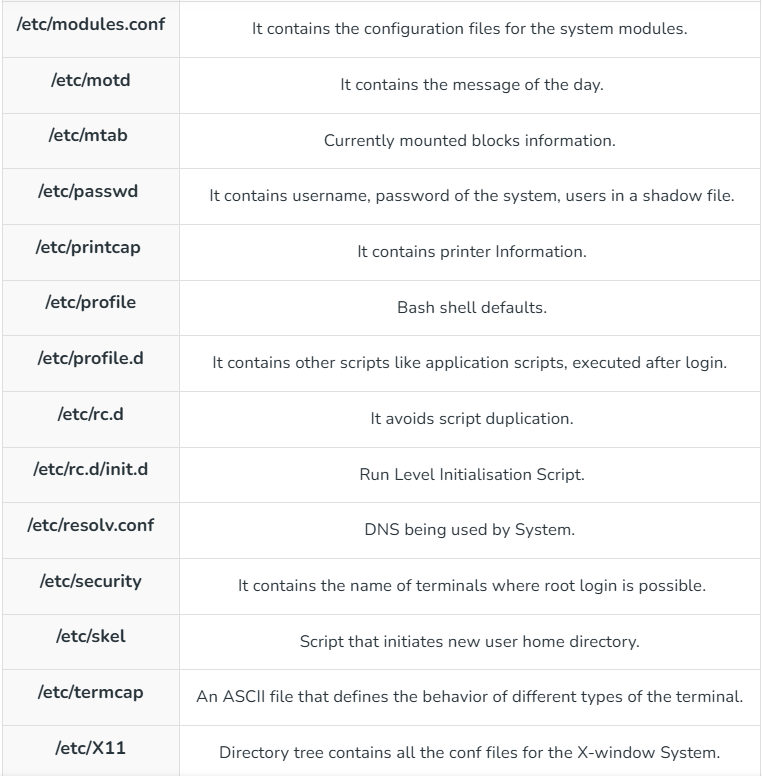
* /boot/vmlinux – The Linux kernel file.

Device Files:

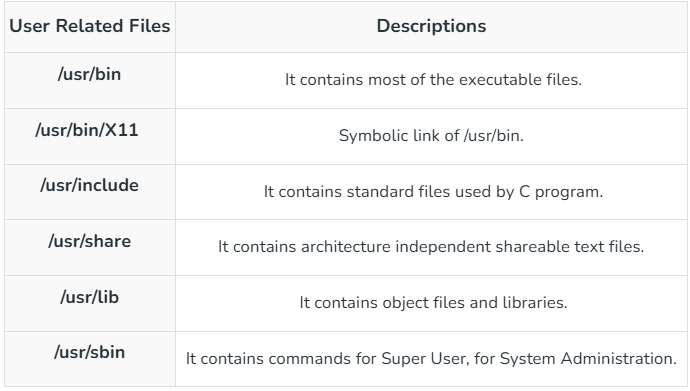
* /dev/hda – Device file for the first IDE HDD.
* /dev/hdc – A pseudo-device that output garbage output is redirected to /dev/null.

**System Configuration Files:**

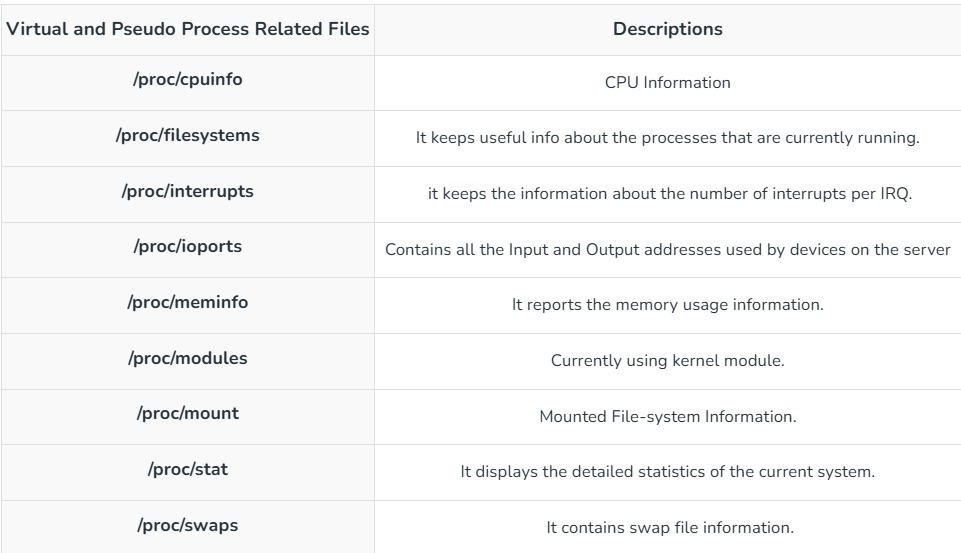




**User related files:**



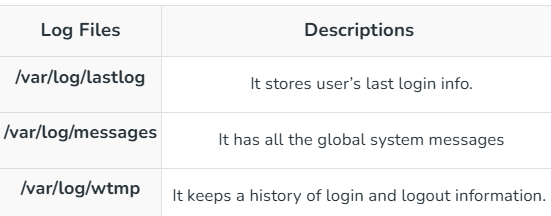
**Virtual and Pseudo Process Related Files:**

****

**Version Information File:**

* **/version – It displays the Linux version information.**

**Log Files:**

****

**Shell Scripting**

**Introduction to Linux Shell and Shell Scripting**

If we are using any major operating system, we are indirectly interacting with the shell. While running Ubuntu, Linux Mint, or any other Linux distribution, we are interacting with the shell by using the terminal. In this article we will discuss Linux shells and shell scripting so before understanding shell scripting we have to get familiar with the following terminologies:

* Kernel
* Shell
* Terminal

**Table of Content**

* [What is Kernel?](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#what-is-kernel)
* [What is Shell?](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#what-is-shell)
* [Command Line Shell](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#command-line-shell)
* [Graphical Shells](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#graphical-shells)
* [What is a terminal?](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#what-is-a-terminal)
* [Shell Scripting](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/#shell-scripting)

**What is Kernel?**

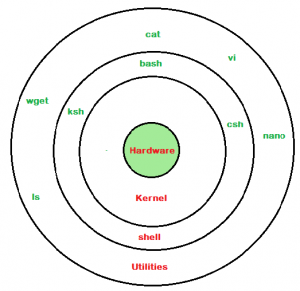
The kernel is a computer program that is the core of a computer’s operating system, with complete control over everything in the system. It manages the following resources of the Linux system –

* File management
* Process management
* I/O management
* Memory management
* Device management etc.

It is often mistaken that Linus Torvalds has developed Linux OS, but actually, he is only responsible for the development of the Linux kernel.

**What is Shell?**

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.



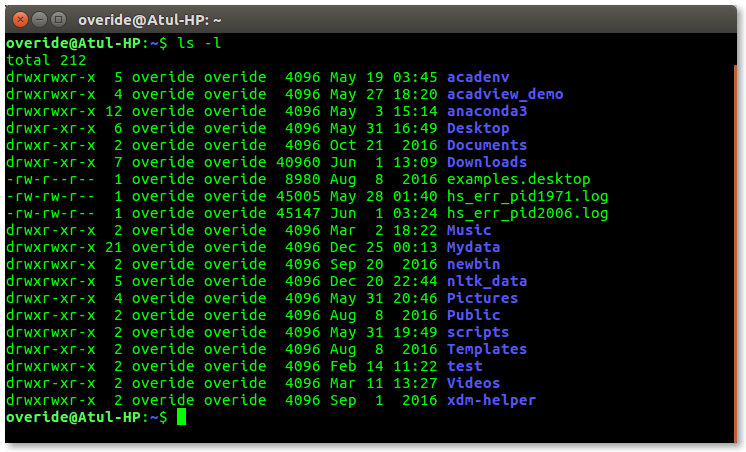
*Linux Shell*

Shell is broadly classified into two categories –

* Command Line Shell
* Graphical shell

**Command Line Shell**

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as “cat”, “ls” etc. and then it is being executed. The result is then displayed on the terminal to the user. A terminal in Ubuntu 16.4 system looks like this –



In the above screenshot “**ls**” command with “**-l**” option is executed. It will list all the files in the current working directory in a long listing format.  
Working with a command line shell is a bit difficult for beginners because it’s hard to memorize so many commands. It is very powerful; it allows users to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called batch files in Windows and **Shell**Scripts in Linux/macOS systems.

**Graphical Shells**

Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as a good example which provides GUI to the user for interacting with the program. Users do not need to type in commands for every action. A typical GUI in the Ubuntu system

There are several shells are available for Linux systems like –

* BASH (Bourne Again SHell) – It is the most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
* CSH (C SHell) – The C shell’s syntax and its usage are very similar to the C programming language.
* KSH (Korn SHell) – The Korn Shell was also the base for the POSIX Shell standard specifications etc.

Each shell does the same job but understands different commands and provides different built-in functions.

**What is a terminal?**

A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal.

To access the terminal, simply search in search box “terminal” and double-click it.

**Shell Scripting**

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts**or**Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with**`.sh`** file extension e.g., **myscript.sh.**

A shell script has syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

A shell script comprises the following elements –

* Shell Keywords – if, else, break etc.
* Shell commands – cd, ls, echo, pwd, touch etc.
* Functions
* Control flow – if..then..else, case and shell loops etc.

**Why do we need shell scripts?**

There are many reasons to write shell scripts:

* To avoid repetitive work and automation
* System admins use shell scripting for routine backups.
* System monitoring
* Adding new functionality to the shell etc.

**Some of the advantages of shell scripts**

* The command and syntax are exactly the same as those directly entered in the command line, so programmers do not need to switch to entirely different syntax
* Writing shell scripts are much quicker
* Quick start
* Interactive debugging etc.

**Some Disadvantages of Shell scripting**

* Prone to costly errors, a single mistake can change the command which might be harmful.
* Slow execution speed
* Design flaws within the language syntax or implementation
* Not well suited for large and complex task
* Provide minimal data structure unlike other scripting languages etc.

**Comparison Operators**

Integer Comparison

|  |  |
| --- | --- |
| Operator | Description |
| -eq | Is equal to |
| -ne | Is not equal to |
| -gt | Is greater than |
| -ge | Is greater than or equal to |
| -lt | Is less than |
| -le | Is less than or equal to |

**String Comparison**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Is equal to |
| != | Is not equal to |
| \< | Is less than, in ASCII Alphabetical order |
| \> | Is Greater than, in ASCII Alphabetic order |

**Conditional Statements**

**If statement**

It checks the condition, and if it is conditioned true, it executes the commands.

Syntax

If [ condition ]

then

#statements

fi

Lets see an example

#!/bin/sh

x=10

y=11

if [ $x -ne $y ]

then

echo “Not equal”

fi

if-else statement

in an if-else statement, you can specify a set of commands to run if the condition is not met.

Syntax

if [ condition ]

then

#set of statements if the condition is true

else

#set of statements if the condition is false

fi

**Lets see an example**

#!/Syntaxbin/sh

x=10

y=10

if [ $x -ne $y ]

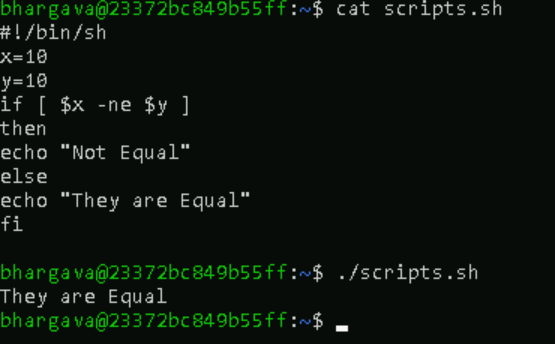
then

echo “Not equal”

else

echo “They are Equal”

fi



**Loops**

While Loop

It starts running the specified commands if the condition is true and repeats them until the condition is false

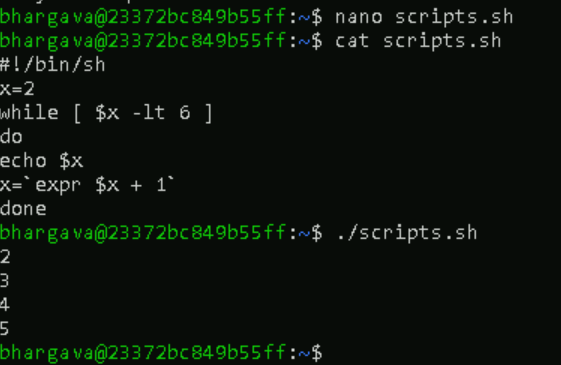
Syntax

while [ condition ]

do

#set of statements

done



**For loop**

In a for loop, the variable iterates over a list of values and ends when there are no more values to iterate over.

Syntax

for var in val1 val2 val3

do

#statements

done

Example

#!/bin/sh

for var in 2 4 5 8

do

echo $var

done

**Positional Arguments**

Positional arguments are the arguments or values which we pass to the shell script while executing the script. They are accessed by variables $0,$1,$2…..$9. Beyond that, they are referenced by ${10},${11} and so on. $# stores the no of passed arguments and $0 stores the script. Lets see an example to understand all this.

#!/bin/sh

echo “No of arguments is $#”

echo “Name of the script is $0”

echo “First argument is $1”

echo “Second argument is $2”

To pass the argument, just type them in the terminal after the script name as shown below.

**Storing the output of commands**

You can store the output of commands inside a variable in a shell script. There are two ways to do so.

Syntax

#Syntax 1

var=$(a valid linux command)

#Syntax 2

var2=`a valid linux command`

**Example**

#!/bin/sh

b=$(pwd)

c=`pwd`

echo $b

echo $c

d=$(ls /bin | grep bash)

echo $d

**Exit code of shell commands**

* Whenever a command ends and returns the control to the parent process, it returns exit codes between 0 and 255. Exit code 0 means the command was successful, and any other exit code means, the command was unsuccessful. You can view the exit code after running any command by accessing the $? variable.
* You can manually set an exit code for your shell script. This can be used with conditional statements to convey if the script’s purpose was achieved or not.

Example

#!/bin/sh

read x

if [ $x -ne 10 ]

then

echo failed

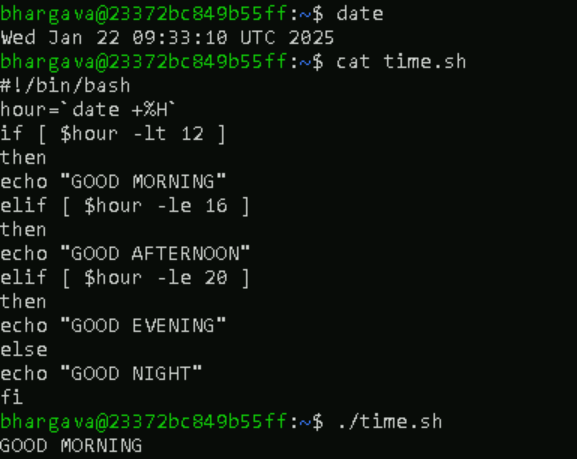
exit 1

else

echo passed

exit 0

fi

r

**Bash**

BASH is an acronym for Bourne Again Shell, a punning name, which is a tribute to Bourne Shell (i.e., invented by Steven Bourne).

Bash is a shell program written by Brian Fox as an upgraded version of Bourne Shell program '**sh**'. It is an open source GNU project. It was released in 1989 as one of the most popular shell distribution of GNU/Linux operating systems. It provides functional improvements over Bourne Shell for both programming and interactive uses. It includes command line editing, key bindings, command history with unlimited size, etc.

In basic terms, Bash is a command line interpreter that typically runs in a text window where user can interpret commands to carry out various actions. The combination of these commands as a series within a file is known as a Shell Script. Bash can read and execute the commands from a Shell Script.

Bash is the default login shell for most Linux distributions

**Shell:** A UNIX Shell is a program or a command line interpreter that interprets the user commands which are either entered by the user directly or which can be read from a file (i.e., Shall Script), and then pass them to the operating system for processing. It is important to note that Shall scripts are interpreted and not compiled, as the computer system interprets them and there is not any need to compile Shell Scripts in order of execution.

There are different types of shells available in Linux Operating Systems. Some of which are as follows:

1. Bourne Shell
2. C shell
3. Korn Shell
4. GNU Bourne Shell

To know, which shell types of your operating system supports, type the command into the terminal as given below:

**cat /etc/shells**

And to know where bash is in your OS, type the below command and you will get a specific location:

**which bash**

**Features of Bash**

1. Bash is **sh-compatible** as it derived from the original UNIX Bourne Shell. It is incorporated with the best and useful features of the Korn and C shell like directory manipulation, job control, aliases, etc.
2. Bash can be **invoked by** single-character command line options (**-a, -b, -c, -i, -l, -r, etc.** ) as well as by multi-character command line options also like --debugger, --help, --login, etc.
3. Bash **Start-up files** are the scripts that Bash reads and executes when it starts. Each file has its specific use, and the collection of these files is used to help create an environment.
4. Bash consists of **Key bindings** by which one can set up customized editing key sequences.
5. Bash contains **one-dimensional arrays** using which you can easily reference and manipulate the lists of data.
6. Bash comprised of **Control Structures** like the **select construct** that specially used for menu generation.
7. Directory Stack in Bash specifies the history of recently-visited directories within a list. Example: **pushd** builtin is used to add the directory to the stack, **popd** is to remove directory from the stack and **dirs** builtin is to display content of the directory stack.
8. Bash also comprised of restricted mode for the environment security. A shell gets restricted if bash starts with name **rbash**, or the bash --restricted, or bash -r option passed at invocation.

Advantages of Bash Scripting

* **Automation**: Shell scripts allow you to automate repetitive tasks and processes and saving time.
* **Portability**: Shell scripts can be run on various platforms and operating systems, including Unix, Linux, macOS, and even windows through the use of emulators or virtual machines.
* **Flexibility**: Shell scripts are highly customizable and can be easily modified to suit specific requirements. They can also be combined with other programming languages or utilities to create more powerful scripts.
* **Accessibility**: Shell scripts are easy to write and don't require any special tools or software. They can be edited using any text editor, and most operating systems have a built-in shell interpreter.
* **Integration**: Shell scripts can be integrated with other tools and applications, such as databases, web servers, and cloud services, allowing for more complex automation and system management tasks.
* **Debugging**: Shell scripts are easy to debug, and most shells have built-in debugging and error-reporting tools that can help identify and fix issues quickly.

**Bash script**

Bash script is a computer program written in the Bash Programming language.

**How to create and run a Bash Script?**

* To create an empty bash script , first change the directory in which you want to save your script using cd command
* Try to use text editor like gedit in which you want to type the shell command
* Use touch command to create the zero bytes sized script touch file\_name
* TO open the script in the text editor type gedit file\_name.sh
* Here .sh is suffixed as an extension that you have to provide for execution.
* Type the shell commands for your bash script in the newly opened text window or the text editor.
* Before typing bash shell commands, first, look at the base of any bash script.

Each Bash based Linux script starts by the line-

1. **#! /bin/bash**

* Where #! Is referred to as the shebang and rest of the line is the path to the interpreter specifying the location of bash shell in our operating system.
* Bash use # to comment any line.
* Bash use echo command to print the output.
* At the end, execute the bash script prefixing with ./.