CS – 19 Open Source Tools

BCA Semester 3

Unit – 1

Open Source Software

Understanding Open Source Software

Definition: Software whose source code is freely distributed with a license to study, update and further distributed to anyone to fulfil a purpose is called open source software. Open source software is generally a team effort where dedicated programmers improve upon the source code and share the changes within the community. Some common examples of Open source software are Linux, Android, and ReactOS.

Open-source software is defined as software that is freely available for modification, usage, and dissemination. The copyright holder offers permission or control over the source code to anyone who wishes to modify it for next level like to add on additional functionalities. There are different compelling reasons to share something under an open source license, ranging from "more perspectives make better software" to "establishing a standard." It is critical when developing a sustainable project to consider your reasons for publishing as open source and utilize these as a guide for decision making.

Characteristics of Open Source Software: Some of the key Characteristics of Open Source Software are as follows

- **Flexibility** − Experts can make necessary changes in the software as per their needs.
- **Stability** − Availability of technical experts in the open source community to look after the software. Hence, users can make the software stable and can be used for the long term.
- **Security and Reliability** − Since the program is being developed and improved by a team of individuals. Thus, software is more secure and reliable.
- **Better Support** – Since the program is used by a large number of people, including developers, businesses, and end users. As a result, to get technical support is easier.
- **Absolute Transparency** – Open-source code visibility has several benefits. Users can view the code, enhancing trust in the software provider. Publicly available code fosters a sense of stability, ideal for long-term projects with reduced risk of abrupt discontinuation.
- **Agility** – Agility is vital in modern business. Firms must stay nimble to outshine competitors. Open-source software is suited for agility as it provides multiple solutions for a single problem.

Types of Open Source Software: Some common types of Open Source Software are as follows −

- **Freeware**: A software that is available free of cost for use and distribution but cannot be modified as its source code is not available is called freeware. Examples of freeware are Google Chrome, Adobe Acrobat PDF Reader, Skype, etc.
- **Shareware**: Software that is initially free and can be distributed to others as well, but needs to be paid for after a stipulated period of time is called shareware. Its source code is also not available and hence cannot be modified.
- **Proprietary Software**: Software that can be used only by obtaining license from its developer after paying for it is called proprietary software. An individual or a company can own such proprietary software. Its source code is often closely guarded secret and it can have major restrictions like −
    - No further distribution
    - Number of users that can use it
    - Type of computer it can be installed on, example multitasking or single user, etc.

For example, Microsoft Windows is proprietary operating software that comes in many editions for different types of clients like single-user, multi-user, professional, etc.

Advantages of Open Source Software: Some of the key advantages of Open Source Software are as follows

- Security
- Customization

- Affordability
- Scalability
- Interoperable on multiple platforms
- Powering the digital transformation
- The open-source community

Principles

- **Transparency**: Open source principles prioritize transparency in code, processes, and decision-making. This includes making source code available for anyone to view, modify, and distribute.
- **Participation**: Open source encourages active participation from a wide range of individuals, regardless of role or experience level. This fosters a sense of community and collaboration.
- **Collaboration**: Open source development involves working together to solve problems and create solutions. This leads to innovative and effective outcomes.
- **Sharing**: Open source principles promote the sharing of knowledge, code, and expertise. This enables rapid prototyping, iteration, and improvement.
- **Empowerment**: Open source communities empower individuals to contribute and make decisions, promoting a sense of ownership and accountability.
- **Meritocracy**: Decisions are made based on merit, expertise, and contributions, rather than hierarchical authority or personal relationships.
- **Inclusivity**: Open source communities strive to be inclusive, welcoming, and diverse, recognizing that diverse perspectives and skills lead to better outcomes.

History and evolution

The term Open Source was invented in 1998 when the Open Source Initiative (OSI) organization was created. The term was coined to make the movement more business-friendly than the already established “Free Software” movement which is more a political social movement. Open Source simply means that the source code that makes a program is included in the package distribution so anybody can make changes to such program. That is the difference between Open Source and commercial software where you only get the built product, and only the creator of such product can make changes to it. If you are not familiar with the process of how a computer program is created then here is a quick overview. A computer program is created in two phases, the first phase is when the program is coded by a computer programmer using his preferred programming language like JAVA, Python, C, C++ etc. That is the source code of the program, but computers can’t read the source code, so the program needs to be “compile” to a “binary” form. The binary code is normally represented with 1 and 0 which all computers can read. You can compile a program from its source code, but once the program is compiled you cannot ‘de-compile” it back to its original source code. This is the difference between commercial and open source software. In commercial software, you get the program in its compiled form, but not the source code, in an Open Source program, you get both.

Open-Source Licensing

Overview: Open-source licenses are software licenses that allow content to be used, modified, and shared. They facilitate free and open-source software (FOSS) development. Intellectual property (IP) laws restrict the modification and sharing of creative works. Free and open-source licenses use these existing legal structures for an inverse purpose. They grant the recipient the rights to use the software, examine the source code, modify it, and distribute the modifications. These criteria are outlined in the Open Source Definition.

After 1980, the United States began to treat software as a literary work covered by copyright law. Richard Stallman founded the free software movement in response to the rise of proprietary software. The term "open source" was used by the Open Source Initiative (OSI), founded by free

software developers Bruce Perens and Eric S. Raymond. "Open source" emphasizes the strengths of the open development model rather than software freedoms. While the goals behind the terms are different, open-source licenses and free software licenses describe the same type of licenses. The two main categories of open-source licenses are **permissive** and **copyleft**. Both grant permission to change and distribute software. Typically, they require attribution and disclaim liability. Permissive licenses come from academia. Copyleft licenses come from the free software movement. Copyleft licenses require derivative works to be distributed with the source code and under a similar license. Since the mid-2000s, courts in multiple countries have upheld the terms of both types of license. Software developers have filed cases as copyright infringement and as breaches of contract.

Rights and responsibilities of users and developers under open source licenses: The answer to this question depends on a few different factors, but the key consideration is what you want to allow other people to do with your code: Do you want to keep it open source or you agree to have your code as part of proprietary or commercial software? You also have to consider if you want to credit yourself as the author as part of the license (attribution). Here are a few things to consider when deciding what open source license to select for your open source software:

- Reflect on whether you'd feel at ease with someone else developing, hosting, selling, or distributing a version of your open source project. If you are comfortable with that scenario, a permissive license would be a good fit and then you can select one based on minor differences between MIT, Apache, or other permissive licenses.
- If you prefer to ensure that modifications to the source code remain open source, consider a copyleft license.

For open source at your workplace, you should consult with a lawyer or legal expert on copyright and open source licenses. A legal team should be able to recommend the license that is best suited for your organization.

Contracts & licenses and related issues: Organizations of all sizes and in all industries should establish a systematic approach for identifying and monitoring the presence of open source components in all software to maintain open source compliance. Organizations should define polices to stipulate that no open source component is integrated or used by other software without a prior identification of its origin, version, and open source license. As previously described, copyleft licenses can present restrictions if the software is going to be commercialized. Organizations should generate a software bill of materials (SBOM) that includes open source license information. This process ensures transparency and compliance with open source licensing requirements within your software development practices.

Application of Open sources

Some of the best open Source Applications are as follows:

- Linux is widely used for servers and devices.
- Mozilla Firefox and Chromium are open-source web browser choices.
- LibreOffice and Apache OpenOffice are free alternatives to Microsoft Office.
- WordPress and Joomla aid in website creation.
- GIMP is for image editing, and Blender is for 3D content creation.
- Postgre SQL and MySQL manage data in databases.
- Apache and Nginx host websites and distribute traffic.
- Eclipse and Git are development tools.
- QGIS assists with maps and geographic data

Open Sources Operating System:

FEDORA: Fedora Linux is a free and open-source operating system based on the Linux kernel and was developed by the community-supported Fedora Project. It is known for its fast release cycle, which keeps the operating system up to date with the latest software and technologies. Fedora Linux is a

free and open-source operating system based on the Linux kernel, which is a set of programs that allows you to interact with your computer.

It was created by the Fedora Linux Project and sponsored by Red Hat Linux, a leading provider of open-source solutions. Fedora Linux is known for its rapid release cycle, which means that new versions of the operating system are released regularly, typically every six months. This assists in keeping the operating system current with the latest software and technologies. It offers a suite of virus protection, system tools, office productivity services, media playback, and other desktop applications. Fedora Linux OS is integrated with applications and packaged software and enhances the ability of the software.

History of the Fedora Linux Operating System: Fedora's name derives from Fedora Linux, a volunteer project that provided extra software for Red Hat Linux Distribution. The Fedora Project began in 2003 as a Red Hat Linux community project to develop a free and open-source operating system suitable for both personal and professional use. Fedora Core 1, the first version of Fedora, was released in November 2003. The operating system has undergone numerous changes and improvements since then, with new versions being released regularly. Before Fedora 7, Fedora was called Fedora Core after the name of two main repositories, Core and Extra. Fedora Core contained all the packages that were required by the operating system. Since the release of Fedora 21, an effort to modularize Fedora distribution and make development more agile.

UBUNTU: Ubuntu is a popular free and open-source Linux-based operating system you can use on a computer or virtual private server. Ubuntu was introduced in 2004 by a British company Canonical. It was based on Debian – a popular distro back then – which was difficult to install. As a result, Ubuntu was proposed as a more user-friendly alternative. As the manager of Ubuntu, Canonical is responsible for releasing a new Ubuntu version every six months. Canonical also provides hosting servers for Ubuntu Community, allowing people worldwide to contribute to testing software bugs, answer questions, and give technical support for free. This article will discuss what Ubuntu is and several reasons why it is so popular. We will also explore the differences between Ubuntu and Linux.

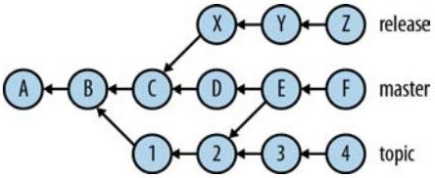
Unit – 2

Open Source Development and Collaboration

Version Control with Git

The general notion of branching, and the usual mechanism by which you share your work with others in Git. A Git project is represented by a "repository," which contains the complete history of the project from its inception. A repository in turn consists of a set of individual snapshots of project content collections of files and directories called "commits." A single commit comprises the following:

- A project content snapshot, called a "tree" A structure of nested files and directories representing a complete state of the project
- The "author" identification Name, email address, and date/time (or "timestamp") indicating who made the changes that resulted in this project state and when
- The "committer" identification The same information about the person who added this commit to the repository (which may be different from the author)
- A "commit message" Text used to comment on the changes made by this commit A list of zero or more "parent commits"
- References to other commits in the same repository, indicating immediately preceding states of the project content
- The set of all commits in a repository, connected by lines indicating their parent commits, forms a picture called the repository "commit graph," shown in Figure.



The letters and numbers here represent commits, and arrows point from a commit to its parents. Commit A has no parents and is called a "root commit"; it was the initial commit in this repository's history. Most commits have a single parent, indicating that they evolved in a straightforward way from a single previous state of the project, usually incorporating a set of related changes made by one person. Some commits, here just the one labeled E, have multiple parents and are called "merge commits." This indicates that the commit reconciles the changes made on distinct branches of the commit graph, often combining contributions made separately by different people. Since it is normally clear from context in which direction the history proceeds—usually, as here, parent commits appear to the left of their children—we will omit the arrow heads in such diagrams from now on.

Branches: The labels on the right side of this picture—master, topic, and release—denote "branches." The branch name refers to the latest commit on that branch; here, commits F, 4, and Z, respectively, are called the "tip" of the branch. The branch itself is defined as the collection of all commits in the graph that are reachable from the tip by following the parent arrows backward along the history. Here, the branches are:

release = {A, B, C, X, Y, Z}
master = {A, B, C, D, E, F, 1, 2}
topic = {A, B, 1, 2, 3, 4}

Note that branches can overlap; here, commits 1 and 2 are on both the master and topic branches, and commits A and B are on all three branches. Usually, you are "on" a branch, looking at the content corresponding to the tip commit on that branch. When you change some files and add a new commit containing the changes (called "committing to the repository"), the branch name advances to the new commit, which in turn points to the old commit as its sole parent; this is the way branches move forward. From time to time, you will tell Git to "merge" several branches (most often two, but there can be more), tying them together as at commit E in Figure. The same branches can be merged repeatedly over time, showing that they continued to progress separately while you periodically combined their contents.

The first branch in a new repository is named master by default, and it's customary to use that name if there is only one branch in the repository, or for the branch that contains the main line of development (if that makes sense for your project). You are not required to do so, however, and

there is nothing special about the name “master” apart from convention, and its use as a default by some commands.

Commit: A version control system manages content changes, and the commit is the fundamental unit of change in Git. A commit is a snapshot of the entire repository content, together with identifying information, and the relationship of this historical repository state to other recorded states as the content has evolved over time. Specifically, a commit consists of:

- A pointer to a tree containing the complete state of the repository content at one point in time.
- Ancillary information about this change: who was responsible for the content (the “author”); who introduced the change into the repository (the “committer”); and the time and date for both those things. The act of adding a commit object to the repository is called “making a commit,” or “committing (to the repository).”
- A list of zero or more other commit objects, called the “parents” of this commit. The parent relationship has no intrinsic meaning; however, the normal ways of making a commit are meant to indicate that the commit’s repository state was derived by the author from those of its parents in some meaningful way (e.g., by adding a feature or fixing a bug). A chain of commits, each having a single parent, indicates a simple evolution of repository state by discrete steps (and as we’ll see, this constitutes a branch). When a commit has more than one parent, this indicates a “merge,” in which the committer has incorporated the changes from multiple lines of development into a single commit. We’ll define branches and merges more precisely in a moment. Of course, at least one commit in the repository must have zero parents, or else the repository would either be infinitely large or have loops in the commit graph, which is not allowed (see the description of a “DAG” next). This is called a “root commit,” and most often, there is only one root commit in a repository—the initial one created when the repository was started. However, you can introduce multiple root commits if you want; the command git checkout --orphan does this. This incorporates multiple independent histories into a repository, perhaps in order to collect the contents of previously separate projects.

What’s a DAG?
Git, by design, will not ever produce a graph that contains a loop; that is, a way to follow the arrows from one commit to another so that you arrive at the same commit twice (think what that could possibly mean in terms of a history of changes!). This is called being “acyclic”: not having a cycle, or loop. Thus the commit graph is technically a “directed acyclic graph,” or DAG for short.

Branches: A Git branch is the simplest thing possible: a pointer to a commit, as a ref. Or rather, that is its implementation; the branch itself is defined as all points reachable in the commit graph from the named commit (the “tip” of the branch). The special ref HEAD determines what branch you are on; if HEAD is a symbolic ref for an existing branch, then you are “on” that branch. If, on the other hand, HEAD is a simple ref directly naming a commit by its SHA-1 ID, then you are not “on” any branch, but rather in “detached HEAD” mode, which happens when you check out some earlier commit to examine.

Merging: Merging is the complement of branching in version control: a branch allows you to work simultaneously with others on a particular set of files, whereas a merge allows you to later combine separate work on two or more branches that diverged earlier from a common ancestor commit. Here are two common merge scenarios:

1. You are working by yourself on a software project. You decide to explore refactoring your code in a certain way, so you make a branch named refactor off of the master branch. You can make any changes you like on the refactor branch without disturbing the main line of development. After a while, you’re happy with the refactoring you’ve done and want to keep it, so you switch to the master branch and run git merge refactor. Git applies the changes

you've made on both branches since they diverged, asking for your help in resolving any conflicts, then commits the result. You delete the refactor branch, and move on.

2. You have been working on the master branch of a cloned repository and have made several commits over a day or two. You then run git pull to update your clone with the latest work committed to the origin repository. It happens that others have also committed to the origin master branch in the meantime, so Git performs an automatic merge of master and origin/master and commits this to your master branch. You can then continue with your work or push to the origin repository now that you have incorporated its latest changes with your own.

There are two aspects to merging in Git: content and history

Merging Content: What it means to successfully "merge" two or more sets of changes to the same file depends on the nature of the contents. Git will try to merge automatically, and often call it a success if the two changesets altered non-overlapping portions of the file. Whether you will call that a success, however, is a different question. If the file is chapter three of your next novel, then perhaps such a merge would be fine if you were making minor grammar and style corrections. If you were reworking the plot line, on the other hand, the results could be less useful—perhaps you added a paragraph on one branch that depends on details contained in a later paragraph that was deleted on another branch. Even if the contents are programming source code, such a merge is not guaranteed to be useful. You could change two separate subroutines in a way that causes them to fail when actually used; they might now make incompatible assumptions about some shared data structure, for example. Git doesn't even check to see that your code still compiles; that's up to you.

Merging History: When Git has done what it can automatically, and you have resolved any remaining conflicts, it's time to commit the result. If we just make a commit to the current branch as usual, though, we've lost critical information: the fact that a merge occurred at all, and which branches were involved. You might remember to include this information in the commit message, but it's best not to depend on that; more importantly, Git needs to know about the merge in order to do a good job of merging in the future. Otherwise, the next time you merge the same branches (say, to periodically update one with continuing changes on the other), Git won't know which changes have already been merged and which are new. It may end up flagging as conflicts changes you have already considered and handled, or automatically applying changes you previously decided to discard.

Open Source Project Management

Project management tools are often specialized according to a specific project management approach: traditional (Waterfall), Agile, Scrum, Kanban, Lean, etc. The traditional project management approach is supported by the Project Management Institute (PMI) that proposes the Project Management Professional (PMP) and CAPM certifications. This approach uses sequential phases of different activities to deliver software. The features provided by traditional open source project management tools are the Work Breakdown Structure (WBS), the Gantt and PERT charts to describe the sequences of tasks, find the critical path, resource allocation graphs, mind maps and risk management. Some allows also to do some time tracking and document sharing.

Overview of Project Management Methodologies (Agile): Agile methodology in project management is a structured approach that segments projects into manageable phases, focusing on continuous improvement. It is an iterative process that involves planning, execution, and evaluation. We share everything you need to know about Agile methodologies, Agile project management, Agile methodology frameworks, and how to implement them in your team. We'll also share our Agile teamwork template to get started with Agile even faster.

Benefits of using Agile methodology: Agile is one of the most popular approaches to project management because it is flexible, it is adaptable to changes, and it encourages customer feedback. Many teams embrace this approach to achieve the following benefits of Agile:

- **Rapid progress:** By effectively reducing the time it takes to complete various stages of a project, teams can elicit feedback in real time and produce working prototypes or demos throughout the process
- **Customer and stakeholder alignment:** Through focusing on customer concerns and stakeholder feedback, the Agile team is well positioned to produce results that satisfy the right people
- **Continuous improvement:** As an iterative approach, the Agile project management methodology allows teams to chip away at tasks until they reach the best end result.

Types of Agile methodologies: Agile project management is not a singular framework but an umbrella term that includes a wide range of methodologies, including Scrum, Kanban, Extreme Programming (XP), and the Adaptive Project Framework (APF).

- **Scrum:** It is ideal for projects with rapidly changing requirements, using short sprints.
- **Kanban:** It visualizes project progress and is great for tasks requiring steady output.
- **Lean:** It streamlines processes, eliminating waste for customer value.
- **Extreme Programming (XP):** It enhances software quality and responsiveness to customer satisfaction.
- **Adaptive Project Framework (APF):** It works well for projects with unclear details, as it adapts to constantly evolving client needs.

Tools for Project Planning, Task Tracking and Team Collaboration (Trello)

Trello is a web-based, kanban-style, list-making application developed by Atlassian. Created in 2011 by Fog Creek Software, it was spun out to form the basis of a separate company in New York City in 2014 and sold to Atlassian in January 2017. The name Trello is derived from the word trellis, which had been a code name for the project at its early stages. Trello was released at a TechCrunch event by Fog Creek founder Joel Spolsky. In September 2011 Wired magazine named the application one of "The 7 Coolest Startups You Haven't Heard of Yet". Lifehacker said "it makes project collaboration simple and kind of enjoyable".

- **Uses:** Users can create task boards with different columns and move the tasks between them. Typically columns include task statuses such as To Do, In Progress, Done. The tool can be used for personal and business purposes including real estate management, software project management, school bulletin boards, lesson planning, accounting, web design, gaming, and law office case management.
- **Architecture:** According to a Fog Creek blog post in January 2012, the client was a thin web layer which downloads the main app, written in CoffeeScript and compiled to minified JavaScript, using Backbone.js, HTML5 .pushState(), and the Mustache templating language. The server was built on top of MongoDB, Node.js and a modified version of Socket.io.

Contributing to open source projects

Contributing to an open source project is a great way to become a better developer, improve your communication skills, and bolster your resume. However, it isn't always obvious how to contribute to open source. Luckily, there are many ways to get started! From bug reports, to documentation fixes, to code patches, it takes many different types of contributions to keep an open source project up and running. Do I need to write code to contribute to open source?

Many people assume that they must write code to contribute to an open source software project. However, this couldn't be farther from the truth! While code is important, there are many valuable ways to contribute to open source that require little or no coding ability.

Contributing Issues (Issue Tracking): Creating issues is one way to contribute to open source without having to write any code. An issue is a written description of something that could be improved about a project, such as a bug report or a feature request. Most open source repositories have an issue tracker, which is a site where people submit issues for that project. The project maintainers then use

the issue tracker to keep track of which issues are being worked on and by whom. There are a few common types of issues:

- Bug reports let the maintainers know about a bug or error in their project.
- Feature requests suggest an entirely new feature or behavior for the project.
- Clarifying questions ask a question about the project that isn't already answered by the project's documentation.

As an example, let's say you wanted to contribute to Pizza Please, an open source website that helps you find the best pizza near you. When you use the website, you see that one of the links in the dropdown menu is broken. To report this to the Pizza Please maintainers, you could open the following issue:

- Expected Behavior: I expected clicking the "User Profile" link in the dropdown menu to bring me to my profile page.
- Actual Behavior: Clicking the "User Profile" link did nothing.
- Browser: Safari 15.5

GitHub has an issue tracker built into every project, but some maintainers may choose to use different software such as Jira or Trello. Before submitting an issue to an open source project, be sure to check out the project's README for any specific instructions.

Contributing Documentation (Pull Requests): Another code-free way to contribute to open source is to write documentation. The quality of a project's documentation can have a huge impact on its success. Clear, detailed, and up-to-date documentation can make it easy for people to use and contribute to an open source project. On the other hand, confusing, out-of-date, or incomplete documentation might cause people to abandon an otherwise awesome project. There are different types of documentation, all of which can be important to a project's success:

- README: introduces people to the project and often contains links to the rest of the project's documentation.
- Getting started guide: a short demonstration of how to use the project for the most simple use cases.
- Usage guide: a more detailed explanation of how to use all of the project's features.
- Contribution guide: an explanation of how to set up the project on a local machine for the purpose of contributing code.

When contributing to a new open source project, it is often helpful to start by writing issues and documentation, and then move onto writing code once you're more familiar with the project. This will give you a strong understanding of how the project is supposed to work, which is a great foundation for building your technical knowledge.

That being said, many people contribute regularly to open source without writing a single line of code, preferring instead to focus on writing issues or documentation. Their work makes it easier for new users and contributors to join the project and ensures the project's longevity.

Contributing Code (Code Reviews): While writing code is the most common way that people think to contribute to open source, it can also be the most daunting. If you're writing your first code contribution to a project, it's a good idea to first read through the issues in the issue tracker. Many project maintainers will put the "good first issue" label on issues that they think are best suited for beginners.

The issue tracker will also show you which issues are assigned – meaning they have someone actively working on them. Checking whether an issue is assigned will prevent you from working on the same issue as somebody else. If you want to work on an issue, make sure to leave a comment saying you're working on it so the maintainers can assign it to you and save others the same trouble.

If you've found an issue you want to work on but don't know where to start, it's okay to ask for help! Many project maintainers and experienced contributors are willing to pair program with newcomers. Leave a comment on the issue asking if anybody would be willing to pair, and don't get discouraged if you don't hear back immediately. Open source maintainers are often busy, and it may take them some time to reply.

Unit – 3
Case Studies

Apache

The Apache Software Foundation (ASF) exists to provide software for the public good. We believe in the power of community over code, known as The Apache Way. Thousands of people around the world contribute to ASF open source projects every day.

Licenses: The Apache Software Foundation uses various licenses to distribute software and documentation, and to accept regular contributions from individuals and corporations and larger grants of existing software products. These licenses help us achieve our goal of providing reliable and long-lived software products through collaborative, open-source software development. In all cases, contributors retain full rights to use their original contributions for any other purpose outside of Apache while providing the ASF and its projects the right to distribute and build upon their work within Apache.

Incubator: The Apache Incubator provides services to projects which want to enter the Apache Software Foundation (ASF). It helps those incoming projects (called "podlings") adopt the Apache style of governance and operation and guides them to the ASF services available to our projects so they can become top-level ASF projects ("TLPs"). The Incubator delegates a few mentors for each podling to act as liaisons with the various ASF teams: Incubator PMC, Infrastructure team, etc., and facilitate the podling's growth and operations.

ASF Community Development : The ASF Community Development (ComDev) project creates and provides tools, processes, and advice to help open-source projects improve their own community health. We are primarily focused on ASF projects, but we believe that our governance principles and best practices apply to other projects as well. If you're new to the ASF and are looking for where to get involved, you can find relevant information here.

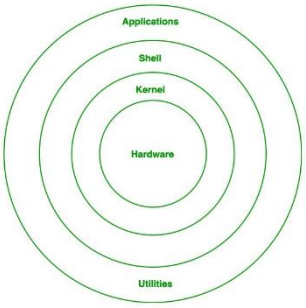
Linux Operating System

The Linux Operating System is a type of operating system that is similar to Unix, and it is built upon the Linux Kernel. The Linux Kernel is like the brain of the operating system because it manages how the computer interacts with its hardware and resources. It makes sure everything works smoothly and efficiently. But the Linux Kernel alone is not enough to make a complete operating system. To create a full and functional system, the Linux Kernel is combined with a collection of software packages and utilities, which are together called Linux distributions. These distributions make the Linux Operating System ready for users to run their applications and perform tasks on their computers securely and effectively. Linux distributions come in different flavors, each tailored to suit the specific needs and preferences of users.

History: Linus Torvalds designed the free and open-source Linux operating system kernel in 1991. Torvalds set out to develop a free and flexible system for personal computers, drawing ideas from the UNIX operating system and the MINIX operating system. Teamwork in development was encouraged with the initial release of the Linux kernel, which attracted developers and enthusiasts globally quickly. Various open-source software packages integrated with the Linux kernel created fully operational operating systems, occasionally referred to as Linux distributions. Over the years, Linux has become known as a key component of modern computing, powering everything from servers and personal computers to supercomputers and smartphones. Due to its flexibility, durability, and strong community support, developers, businesses, and educational institutions frequently opt for it.

Distribution: Linux distribution is an operating system that is made up of a collection of software based on Linux kernel or you can say distribution contains the Linux kernel and supporting libraries and software. And you can get Linux-based operating system by downloading one of the Linux distributions and these distributions are available for different types of devices like embedded devices, personal computers, etc. Around 600 + Linux Distributions are available and some of the popular Linux distributions are: Manjaro, Ubuntu, Debian, and Fedora.

Architecture of Linux: Linux architecture has the following components:



- Kernel: Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are: Monolithic Kernel, Hybrid kernels, Exo kernels, Micro kernels.
- System Library: Linux uses system libraries, also known as shared libraries, to implement various functionalities of the operating system. These libraries contain pre-written code that applications can use to perform specific tasks. By using these libraries, developers can save time and effort, as they don't need to write the same code repeatedly. System libraries act as an interface between applications and the kernel, providing a standardized and efficient way for applications to interact with the underlying system.
- Shell: The shell is the user interface of the Linux Operating System. It allows users to interact with the system by entering commands, which the shell interprets and executes. The shell serves as a bridge between the user and the kernel, forwarding the user's requests to the kernel for processing. It provides a convenient way for users to perform various tasks, such as running programs, managing files, and configuring the system.
- Hardware Layer: The hardware layer encompasses all the physical components of the computer, such as RAM (Random Access Memory), HDD (Hard Disk Drive), CPU (Central Processing Unit), and input/output devices. This layer is responsible for interacting with the Linux Operating System and providing the necessary resources for the system and applications to function properly. The Linux kernel and system libraries enable communication and control over these hardware components, ensuring that they work harmoniously together.
- System Utility: System utilities are essential tools and programs provided by the Linux Operating System to manage and configure various aspects of the system. These utilities perform tasks such as installing software, configuring network settings, monitoring system performance, managing users and permissions, and much more. System utilities simplify system administration tasks, making it easier for users to maintain their Linux systems efficiently.

Advantages of Linux

- The main advantage of Linux is it is an open-source operating system. This means the source code is easily available for everyone and you are allowed to contribute, modify and distribute the code to anyone without any permissions.
- In terms of security, Linux is more secure than any other operating system. It does not mean that Linux is 100 percent secure, it has some malware for it but is less vulnerable than any other operating system. So, it does not require any anti-virus software.
- The software updates in Linux are easy and frequent.
- Various Linux distributions are available so that you can use them according to your requirements or according to your taste.
- Linux is freely available to use on the internet.
- It has large community support.
- It provides high stability. It rarely slows down or freezes and there is no need to reboot it after a short time.
- It maintains the privacy of the user.

- The performance of the Linux system is much higher than other operating systems. It allows a large number of people to work at the same time and it handles them efficiently.
- It is network friendly.
- The flexibility of Linux is high. There is no need to install a complete Linux suite; you are allowed to install only the required components.
- Linux is compatible with a large number of file formats.
- It is fast and easy to install from the web. It can also install it on any hardware even on your old computer system.
- It performs all tasks properly even if it has limited space on the hard disk.

Disadvantages of Linux

- It is not very user-friendly. So, it may be confusing for beginners.
- It has small peripheral hardware drivers as compared to windows.