



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
7th January, 2023





Introduction to Open Source Software

Session 1 : Agenda

- About Course and Evaluation criteria
 - Open Source : what and why
 - Open Source Initiatives
 - Open Source software and examples
 - Proprietary software and examples
 - Advantages and Disadvantages of Open source software
 - Principle of Open source software
-

Open Source

- What is open source ?
 - Open
 - Collaboration is open
 - Source
 - Source is freely available
 - What is means by free ?
 - Share, Adapt, Modify and Collaborate



Open source is about philosophy and consumer's right, it's about free collaborations by community

Why open source

- Access to the best software
 - Greater Privacy and Control
 - Extended Hardware Life
 - Doing things your way
 - Comprehensive Support
 - Greater security
 - Quick Bug and Security Fixes
 - Instant Gratifications
 - No cost
-

Open Source Initiatives (OSI)

- Open-source initiatives (OSI) is a public benefits corporation
- For Education, Advocacy, and stewardship for collaboratives Development
- Founded in 1998 – actively engaged in building open-source community.
- Opens Source Definitions
- Open Standard Requirements
- Open Source Licenses



Open Source Initiative
[\(https://opensource.org/\)](https://opensource.org/)

Open Source Software (OSS)

- Open Source software (OSS) is a type of software in which Source code is released
 - Released under a **License** in which the **copyright** holder grants users the rights to **use, study, change, and distribute** the software and its source code to anyone and for any purpose.
 - Publicly available for anyone to **use, edit, inspect, modify and distribute**.
 - Open Source software is usually developed in a collaborative Public manner
-

Examples of Open source software

Categories of software	Open Source software
Programming Languages	PHP, Java, Python
Presentation Software	Libre Office's Impress, Apache Office's Impress
Communications Software	Free Switch, openPBX, Thunderbird
Content Management System	PHP-Nuke, WordPress, Joomla
Operating system	Linux, Ubuntu, Fedora, FreeBSD
Databases	MySQL, PostgreSQL
Web Design software	Adobe Dreamweaver, Brackets,
Web Browsers	Firefox, Mozilla, Arena, Chromium
Application Servers	JBoss, Tomcat, Glassfish, WebSphere
Source code Management/ Version control	Git, GitHub, Subversion

Proprietary Software

- Proprietary software, also known as **Protected or Restricted** software,
- Usually owned by an organization or a group of people;
- In order to use a proprietary software, end users must **accept** a License
- Proprietary software, also known as **Non Free software** or **Closed software** , is computer software for which the software's publisher or owner retains Intellectual property rights – usually copyrights of the software, and also sometimes patent rights.
- Non-free or Closed software :
 - It's not about cost of software; it's about source code unavailability

Examples of Proprietary Software

Categories of software	Open Source software examples
Programming Languages	MATLAB, VBScript
Presentation Software	MS–office
Content Management System	Adobe Experience Manager, Kentico, Sitecore
Operating system	Windows, Mac OS, Apple iOS
Databases	Oracle , DB2, Microsoft SQL Server, Informix Dynamic Server. SQL Anywhere
Web Browsers	Apple Safari, Browser , Internet Explorer
Application Servers	Weblogic, Sybase Enterprise Application server
Source code Management/ Version control	PerForce, Microsoft Team Foundations Server

Proprietary Software....

- Such Licenses restrict the right of the user, in any of the following manner :
 - Restricted right of redistribution
 - Restricted on reconstructions source code
 - Restricted ways in which the Product can be used or embedded within another Product
- Until recently, proprietary software was the only real model that was used by commercial software
- In proprietary software, the sources code is only available with the owner organization
- Some trusted partner may be given rights to use the same - under the judication of the non-disclosure agreement (NDA)

Advantage of OSS

Low Cost : OSS usually does not require a licensing fee, hence free of cost

Flexible : Can be modified by anyone and tailored to suit specific business needs

Quality and Reliability : One needs to identify and select an OSS that suits business needs and is of good quality . Usually, a mature OSS is generally viewed to be of good quality and considered to be more reliable

Vendor Independence : in case of use of proprietary software , there is usually a contract with a specific vendor (involving high cost, restricted usage). This is overcome in OSS

Advantage of OSS...

Availability of external Technical Support Services by vendors : For example

- Red Hat (now owned by IBM) provides support for many OSS
- MySQL supports provided by the patent company MYSQL AB – now owned by Oracle

- Several open source products have active Online community supports

Disadvantage of OSS

Lack of Personalised support services :

- Unlike proprietary software, OSS product or packages do not come with personalised support facility over phone or email.
 - However for Some mature OSS, there might be some commercial service provides that may provide support services
- **Limited Choices of OSS products**
- **Continuous changes** : continuous changes are made to open source software – difficult to get a compatible , and bug free version at a particular time
- **No warranty** : No warranty support is provided along with OSS since it is not owned by a single company

Principle of Open Source software...

Openness :

Publishing the design and source code of a software to public - with intent that

- Openly fixed or contributed to
- Openly scrutinised / criticized
- Open feedback obtained
- Analysed for bugs or defects
- Analysed for quality

Principle of Open Source software..

Transparency :

- Ability of the community to see the current progress and future plans :
 - projects roadmap is made available to the community
 - A defects tracking system is put in place - for reporting and reviewing defects
 - Publish design documents
- To make more effective decision and understand how decision affects us.

Collaboration : everyone free to participate, enhance each others work in unanticipated ways to unlock new possibilities

Principle of Open Source software..

Release Early and Often : Rapid prototypes and iterative approach

- Any changes proposed or made by any one are made public immediately.
- Contributions are expected to occur early – resolve errors early in development Lifecycle
- Contributions are expected to occur Often - changes shared with others immediately / regularly

Expectations of Community : Participants in an open source projects have expectations of a community - to be formed – that works together - to contribute to the development of the projects

References and further readings

Open Source Initiative (<https://opensource.org/>)

Open Source Resources (<https://opensource.com/>)

Open Source Guides (<https://opensource.guide/>)

Working with GitHub for Open Source Software Development (<https://github.com/>)



Open Source Software Engineering

SE ZG587

BITS Pilani
Pilani Campus

Kumar Manish
21st January, 2023

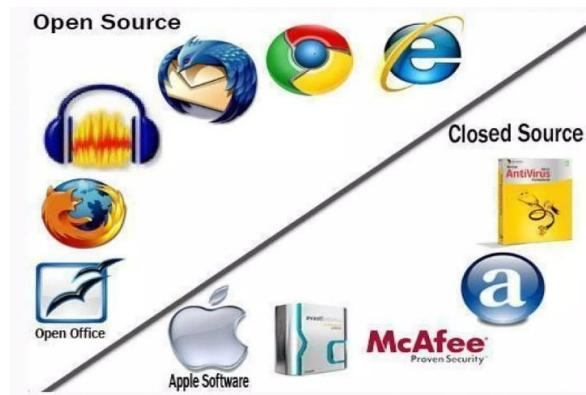




- **Introduction of open source software**
- **Understanding Free, Open Source, Freeware, and Public domain software**

Recap : Contact Session 1

- Open Source : What and why
- Open Source Initiatives
- Open Source Software and examples
- Proprietary Software and examples
- Advantages and Disadvantages of Open source software
- Principle of Open source software



Contact Session 2 : Agenda

- History of Open Source Software
 - Cost of Open Source Software
 - Understanding of Free Software
 - Understanding of Open Source Software
 - Understanding of Freeware
 - Understanding of Public domain software
-



History of Open Source Software

History of Open Source Software

1960

Hardware Expensive
but bundled Free
Software

1970

**Emergence of
Operating Systems,
Compilers**

• Separate selling

1980

**Free Software
Movement**

- 1985 : Free Software Foundations (FSF) GNU project
- Berkeley (BSD Unix)
- 1991 : Linus Torvalds – Launched GNU LINUX
- 1994 : Robert McCool - Apache HTTP server Open source Webserver

1990

**Open Source
Software Movement**

- 1998 : Bruce Perens and Eric S. Raymond - Open Source Initiatives (OSI) as Organisations
- Open source definitions (OSD)

Cost of Open source Sofware

Although OSS is free , there are some hidden cost :

1. Total cost of Ownership (TCO)

1. Cost associated with adopting and managing the software
2. Access to software updates, support services

2. Switching cost :

1. Migrating data from older systems
2. Training cost to Resources / users etc

3. Additional Cost :

1. POC/ POV : Evaluation and Selection
2. Integration cost
3. Fixing critical bugs



Understanding of Free Software

Free Software

- “Free software” means software that respects users' freedom and community. It refers to free use of software and not price
- **Definition** : Free software is the software that can be used , modified, studied, copied, changed and redistribute (with or without modifications) with no restrictions.
- **Philosophy** : Social Movement
- **Charge** : Free software is available free of charge , in the most cases , But , in principle, free software need not necessarily be free of cost;
 - Free software does not mean non commercial
 - One always has freedom to change or copy free software and then sell it.
 - You may even sell the original software
- **Copyright** : Yes
- Examples : 

Free Software – Social Movement

- The free software movement is a Social Movement
 - With the aim of gaining and assuring certain freedoms for software users
 - Movement was Founded by Richard Stallman , in 1983, by launching **GNU Project**
 - In 1985 established **The Free software Foundations**

“The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom. We defend the rights of all software users.”



- Four essential Freedoms of free software
 - Software which meets these freedom requirements is termed free software

Free software – Four essential freedoms

- A program is “free software” if its users have the following :
 - Freedom 0 : The freedom to run the program as you wish, for any purpose
 - Freedom 1 : The freedom to study how the program works, and change it as per your requirements
 - Freedom 2 : The freedom to redistribute copies so you can help others
 - Freedom 3 : The freedom to distribute copies of your modified versions to others
 - **Pre-requisite “ Access of the source code”**

Free Software - Legal considerations

- Legal considerations :
 - The owner of the free software does not have power to withdraw or invalidate the license , or add additional restrictions to its terms and conditions,
 - The license terms should be permanent and irrevocable

GNU in a Nutshell

- GNU was launched by Richard Stallman, as an operating system which would be put together by people working together for the freedom of all software users to control their computing.
- The name of the system, GNU, is a recursive acronym meaning GNU's Not Unix—a way of paying tribute to the technical ideas of Unix, while at the same time saying that GNU is something different.
- Technically, GNU is like Unix. But unlike Unix, GNU gives its users freedom.



Free Software - Licenses

A large number of licenses qualify as free software licenses and are fee compatible with GNU General Public licenses :

Examples :

- GNU General Public License (GPL) version 3
- GNU General Public Licenses (GPL) version 2
- GNU All – Permissive License
- Apache License version 2.0
- Clarified Artistic License

Refer a complete list of licenses at :

<https://www.gnu.org/licenses/license-list.html>



Understanding of Open source

Open Source Software

Definition : Open source software (OSS) is a software in which

- Source code is released under licenses, and
- The owner of the software (or copyright holder) permits the users the rights to use , modify and distribute the software to anyone and for any purpose
- Philosophy : Development methodology – open collaboration Model
- Rules : Governed by rules of Open source Initiative - open source Definitions (OSD) <https://opensource.org/osd>)
- Charge : Available free of charge, in most cases . But in principle , it need not necessarily be free of cost.
- Copyright : Yes
- Examples :



Open Source Software – Open collaborations

The Main Principle behind software development model are :

- Decentralised Software Development
- Open collaboration, and
- Peer Production

As per Wikipedia

- Open collaboration is “a System of innovation or Production or Development that relies on goal-oriented, yet loosely coordinated, participants who interact to create a product (or service) of economic value, and make it available to contributors and non contributors alike”.

Rules for Distributions of open source Software



- Free Redistributions
 - Source code
 - Derived works
 - Preserving Integrity of Original software
 - No discriminations based on Person or groups
 - No discrimination based on Field of Endeavours
 - Distribution of license
 - License should not be product specific
 - License Must not restrict other
-

Open Source Software – License

License must not restrict other software :

No restrictions should be placed on the license for the other software that would be distributed along with the licensed software .

For example : the license must not insist that all other programs distributed along with this software and through the same channel, should also be open-source

License must be technology neutral :

License should not be grouped based on any specific technology or interface style

Open Source Software – Licenses

Open source Licenses examples :

- Apache Licenses 2.0 (Apache 2.0)
- 3 clause BSD licenses (BSD -3 clause)
- 2 clause BSD licenses (BSD 2-clasue)
- GNU General Public Licenses (GPL)
- GNU Lesser General public License (LGPL)
- MIT License (MIT)
- Mozilla Public License 2.0 (MPL 2.0)
- Eclipse Public License 2.0 (EPL 2.0)



Freeware

Freeware Software

- Free refer to price ,
- freedom to use is restricted by owner
- No source code Available
- Copyright law : yes
- Philosophy :
 - Marketing Goals – Intended to benefit the owner
 - Make profit end of day

Examples : WhatsApp, Skype, Adobe, Gmail etc



Public domain Software

Public Domain Software

- Belong to Public ; Can be modified , distributed or sold even without any attribution by anyone
- Rules : Creative common organisation
 - <https://creativecommons.org>
- Charge : No cost
- License : Creative Common Licenses
- Copyright law : No
- Example : SQL Lite

Creative Common

- **Creative Commons** is a nonprofit organization that helps overcome legal obstacles to the sharing of knowledge and creativity to address the world's most pressing challenges.
- Provide [Creative Commons licenses](#) and [public domain tools](#) that give every person and organization in the world a free, simple, and standardized way to grant copyright permissions for creative and academic works; ensure proper attribution; and allow others to copy, distribute, and make use of those works
- Work closely with major institutions and governments to create, adopt and implement open licensing and ensure the correct use of CC licenses and CC-licensed content



Summary

	Free software	Open source Software	Freeware	Public domain software
Definition	“FREE” is a matter of liberty , not price	“OPEN” does not just mean access to the source code; more about collaboration	Free refers to price , while freedom of the use is restricted by creator	PUBLIC domain belongs to the public as a whole
Philosophy	Social movement	Development methodology	Marketing goals	Copyright disclamation
Rules	Four freedoms	Open source Initiatives		Creative common Org
Free of charge	Not necessary	Not Necessary	YES	YES
Copyright law	YES	YES	YES	NO
Examples	Linux, Ubuntu , MySQL, Apache	Linux, Ubuntu , MySQL, Apache	Skype, Adobe acrobat	SQLite

References and further readings

Open Source Initiative <https://opensource.org/>

Open Source Resources <https://opensource.com/>

Open Source Guides (<https://opensource.guide/>)

Working with GitHub for Open Source Software Development (<https://github.com/>)

[How To Pronounce GNU - GNU Project - Free Software Foundation](#)

[GNU in a Nutshell - GNU Project - Free Software Foundation](#)

GNU Licenses <https://www.gnu.org/licenses/license-list.html>

[What We Do - Creative Commons](#)



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
28th January, 2023





Session 3: Understanding Intellectual property Rights and Software Licenses

Recap : previous sessions

- Open Source : Why, What and Examples
 - Open Source Initiatives (OSI) and Free software Foundations (FSF)
 - Advantages and Disadvantages of Open source software
 - Principle of Open source software
 - History of Open Source Software
 - Cost of Open Source Software
 - Understanding of Free Software
 - Understanding of Open Source Software
 - Understanding of Freeware
 - Understanding of Public domain software
-

Recap : Previous sessions

	Free software	Open source Software	Freeware	Public domain software
Definition	“FREE” is a matter of liberty , not price	“OPEN” does not just mean access to the source code; more about collaboration	Free refers to price , while freedom of the use is restricted by creator	PUBLIC domain belongs to the public as a whole
Philosophy	Social movement	Development methodology	Marketing goals	Copyright disclamation
Rules	Four freedoms	Open source Initiatives		Creative common Org
Free of charge	Not necessary	Not Necessary	YES	YES
Copyright law	YES	YES	YES	NO
Examples	Linux, Ubuntu , MySQL, Apache	Linux, Ubuntu , MySQL, Apache	Skype, Adobe acrobat	SQLite

Agenda : session 3

- Understanding Intellectual Property Rights
- Understanding Software Licenses
 - **Licensing Models in OSS:**
 - Copyright,
 - Copyleft,
 - Permissive,
 - Creative Commons

Intellectual Property Rights (IPR)

- The creator of an artefact is considered to be the owner of the artefact, This implies that the owner who writes the code owns it – unless there exists a written contract that states differently
- In House software development :
 - The entire ownership or the copy right of the software remains with the Organisation
- Outsource software development :
 - The issues related to Intellectual Property (IP) rights are correctly Managed - By written contracts
 - Rights to artefacts like business ideas , images , diagram, source code and documentations remain under the sole ownership of the client company .

Intellectual Property Rights (IPR)

Types of Intellectual Property Rights : relevant to the software industry

- **Patents** – used to protect functional features , like hardware configurations etc
- **Copyrights** – used to protect works of authorship , like source code , diagram etc
- **Trade secrets** – used to protect internal business secrets, like business and pricing models
- **Trademarks** - used to protect brand recognition, through logo etc
 - Patents , copyright and trade secrets are used to protect the technology itself
 - Trademarks do not protect technology, but help in distinguishing a product in the marketplace

IPR in Software Domain

Who can claim the ownership of the IPR ?

All the people involved in a development project, may claim for the ownership of the IPR for the various artefacts developed as a part of the project.

- **Employee involved** – are the first who have the IP rights to all the artefacts developed during employment ; however they are usually restricted by the employment contracts.
- **Consultant or contractors (Organisation or individual)**
 - unless a written contract says otherwise , they also claim for the ownership of the projects artefacts
- **Vendor company developing the software**
 - unless documented , although the clients pay for the services provided by the vendor , they are not necessarily the owners

IPR in Software Domain

Category of Software source code in Projects :

- **Unique code**
- **Open source code**
- **Existing / Third- party code**
- **Unique code** : refers to the code specifically developed for a particular project
 - source code and related artefacts of the software may be limited use to the developer or vendor organisations
 - Vendor organisation may be ready to assign ownership of the same or grant exclusive license to the client company
- **Open Source code** – refers to use of open source code or technologies which are Publicly available
 - Neither the client, nor the vendor owns the IP rights to the Open sources code or technology ; nor does anyone maintain exclave control over it.
 - Violation of the Open source license could lead to significant risks of the entire projects e.g. as per GPL – one must publish source code of the complete projects
 - Ensure open source are used legally and all Compliance requirements are met.

IPR in software domain

Existing / Third party code – Written by the developing company for other projects – reuse the same for the current projects

- **Existing code** : Vendor is not willing to give the ownership of this type of code, since they would want to continue using it for other clients
- **Third party code or technology** : Pay a license for the code in a way that is compatible with client's projects
 - Other projects artefacts like image , audio and video files etc .. Could also be included



Licensing Models in OSS:

Software Licenses

- A software License is a **Legal instrument** which describes the manner in which a software can be used or redistributed
 - in both source code or object code forms.
- A typical software license grants the **Licensee** (end-user) , the permission to the software in a manner where such a use would **otherwise potentially constitute copyright Infringement of the software** owner's exclusive rights.
- Software Licensing Models :
 - Copyrights
 - Copyleft Licenses - protective
 - Permissive - more free licenses
 - Creative Commons Licenses - Public copyrights licenses

Copyrights

- Copyright is an important intellectual property license that gives the owner an exclusive right on the work created by him / her. The owner is allowed to create copies of the creative work
- The creative work may be of any type - including literary, artistic, educational, or musical form
- As a general rule , for works created after January 1, 1978 , copy right protection lasts for the life of the author plus an additional 70 years.
- In case of a corporate authors , the protection is for the shorter of 95 years from publication or 120 years from creation

Copyleft

- Copyleft is a licensing method,
 - used to make a work (or program) free to use, modify , adapt or extend
 - The freedom to carry out all activities , as aligned with the **four essential freedoms** (prerequisite – source code available)
 - Copy left licenses includes the GNU general public license (GPL) – which was originally written by Richard Stallman
 - the work or program should be **made available to the recipients - Mostly in the form of source code**
- No re-licensing allowed, No commercial usage allowed
- All modified and extended version of the work) or program) should be free as well - hence called **protective licenses**
- Rules for copyleft –
 - all derivative works should be attributed to the creator, open sourced and copyleft
- Copyleft is a generic concept , and can't be sued directly ; one needs to use a specific implementations of the concepts

Variants of Copyleft

- **Strong and Weak copyleft** : The strength of the copyleft license is decided based on the extent its provision are imposed on the derived works

Examples :

- Strong Copyleft :
 - GNU general public License
- Weak copyleft
 - GNU lesser General Public License –
 - Mozilla public license

Weak copyleft licensing :

- Most commonly, weak copyleft licenses are used to create **Software libraries**
- Help software to link to the library and redistributed without requirement for the linking software to also be copyleft –licensed

Copyleft – share alike condition

Share-alike :

- Imposes the requirement that any freedom that is granted regarding the original work must be granted on exactly the same or compatible terms in all derived work
- However , in some cases the author may be willing to share only a certain part of the work, but the share –alike agreement require that the whole body of the work is shared
- The positive side – for an author of source code – any modification to the code will not only benefit the original creator , but that the author will be attributed and recognised and hold equal claim over the changed code.

Permissive Licenses

- Free software licenses with only minimal restrictions on how the software can be used , modified and redistributed (also called Berkeley software distribution : BSD-like or style licenses)
- Rules for usage - what ever user wants , but with few restrictions – derived works must be attributed to the creator
- Source code need not be open or made available in the public domain
- Re-licensing allowed - derivative works can be release under any other licenses or used as proprietary products
- Allows commercial usage
- Examples :
 - GNU All permissive License
 - MIT License
 - BSD licenses
 - Apple Public source licenses
 - Apache license

Comparing Permissive and copyleft

Copy left (Protective Licenses)	Permissive Licenses
<p>Publication of software code of all modified versions or derived works under the original copyleft licenses - protective licenses</p>	<p>Provides No guarantee that derived works of the software will remain free and publicly available ; generally requiring only that the original copyrights notice be remained</p>
	<ul style="list-style-type: none"> - Hence derived works , or future versions , of permissively – licenses software can be released as proprietary software - Permissive licenses offer highly extensive license compatibility as compared to copyleft licenses - Wider adaptability in the open source community

Creative Common Licenses (CCL)

- A public copyright licenses that enable free distribution of copyright work – allowing the users the right to share, use , and build upon a work that – someone lese (the author) has created
- **Rules for usage** - whatever user wants without any restrictions - derived works must be attributed to the creator
- Availability of the source code : No specific terms about the distribution of source code
- Re-licensing allowed
- Commercial usage allowed , however author can decide to allow uses of given work as non commercial
- CCL applied all works : books , plays, movies, photograph , music, articles, blogs and websites
- CCL were initially released in 2002 by creative commons , Non profit Org founded in 2001
- Five version of licenses , latest in 2013 - version 4

Creative Commons Licenses

Compatibility with software :

- **Not recommended for use for software** - since they do not contain specific terms about the distributions of source code, which is important in order to ensure the free reuse and modifiability of software
- Most of the CC licenses are not compatible with the major software licenses , so it would be difficult to integrate CC licenses work with other free software
- However, they may be used for software documentations , or other artistic elements embedded within documentation.
- **Free Software Foundations (FSF)** : In 2011 added “CC 0 version”
- **Open Source Initiative (OSI)** : Not approved under OSI because of its clause which excluded from the scope of the licenses any relevant patents held by the copyrights holder.

How to choose an open source License ?

- Open source licenses help in protecting works contributed in the open source domain
- It also help protect contributors and users from any copyright infringements
- Interested developers or contributors or business will not touch a project without a license protection.

Which license should I choose :

- If I wish to work with a community
- If I wish to keep the licenses simple and permissive
- If I wish to share the improvements made
- If I wish to work without license

References and further readings

Open Source Initiative <https://opensource.org/>

Open Source Resources <https://opensource.com/>

Open Source Guides (<https://opensource.guide/>)

Creative commons <https://creativecommons.org>

GNU <https://www.gnu.org/>

Copyleft <https://copyleft.org/>



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
4th Feb, 2023

Recap

- Intellectual Property Rights and Software Licenses
- Licensing Models in OSS:
 - Copyright,
 - Copyleft,
 - Permissive,
 - Creative Commons



Session 4 :

Understanding and Choosing Open Source Licensing Models

Agenda

Choosing Open Source Licensing Models

- Option 1: Work with a community
- Option 2: Keep it simple and permissive
- Option 3: Need to share improvements
- Option 4: Work without a license

How to choose an open source License ?

Why to choose ?

- Open source licenses help in protecting works contributed in the open source domain
- It also help protect contributors and users from any copyright infringements
- Interested developers or contributors or business will not touch a project without a license protection.

Which License should I choose :

- If I wish to work with a community
- If I wish to keep the license simple and permissive
- If I wish to share the improvements made
- If I wish to work without a license

Work with a community

- To contribute to or modify or adapt an existing projects,
 - Continue contributions using original license available with the project
- Moreover, using the same license might also be a requirement
 - as per stated in the original license terms of the project
- Look at LICENSE or COPYING or README file for more info
- Certain Open source software projects require contributors to sign the [Contributors License Agreement \(CLA\)](#)
 - Simple CLAs vs Detailed CLAs
 - Individual CLAs vs Corporate CLAs
- No centralised body of knowledge engaged in standardizations of CLAs
 - Different organisations write their own versions
 - Larger project need formal CLAs
 - Generally baked by one or more corporations

Ex – Apache, Django, Eclipse Foundations etc

Work with a community

- Normally, a CLA require the contributors to make certain classifications (In terms of legal proofs) , which may comprise of one or more of the following :
 - the contributors is the author of the contributions;
 - the contribution is an original work;
 - the contribution is not subject to third – party license , claims etc;
 - the contribution has the legal right to grant the copyright license;
 - the contributors does not have an employer that can claim rights in the copyright;
 - Benefits of CLA : protection against copyright infringement
 - Disadvantages : discourage contributions
-

Work with a community

How to manage the ownership and licensing of copyrights for individual software contributors to the project ?

Option 1 : The author (who is also, by default, the copyright holder) of the software contributions retains the software 's copyright and ownership and contributes it under the same open source license as used by the project.

Option 2 : The contributors assigns the ownership of the software copyright to The “**Project Maintainer**”. Who then releases the software under the projects’ Open Source License.

Option 3 : the last option is not to define any ownership policy

Keep it simple and permissive

- Choose the MIT License :
 - It is a short and simple permissive license with condition only requiring preservation of copyright and license notices.
 - Licensed works, modification and larger works may be distributed under different terms and without source code.
 - The MIT license allows users the permission to reuse code for any purpose , or embed it as a part of proprietary software or make any changes or modifications to the code to suit their needs.
 - Restrictions - users are required to include the original copy of the MIT licenses

Keep it simple and permissive

- **Permissions :**
 - Commercial Use : the licensed materials and derivatives may be used for commercial purposes
 - Distribution : The licensed materials may be distributed
 - Modification : The licensed materials may be modified
 - Private use : The licensed materials may be used and modified in private
- **Conditions :**
 - A copy of the license and copyright notice MUST be included with the license materials.
- **Limitations :**
 - Liability : The license includes a limitation of liability
 - Warranty : The license explicitly states that it does NOT provide any warranty

Need to share improvements

- Choose the GNU general Public Licenses v3.0 – copyleft license - protective license
 - This strong copyleft license provisions on making available complete source code of licensed works and modifications, including larger works under the same license.
 - Copyright and license notice Must be preserved.
 - Contributors provides an express grant of patent rights.
 - GPL assures that patent can not be used to render the program non free.
-

Need to share improvements

- **Permissions :**
 - Commercial use : The licensed materials and derivates may be used for commercial purposes.
 - Distributions : the licensed materials may be distributed.
 - Modification : The licensed materials may be modified
 - Private use: The license materials may be used and modified in private
 - Patent use : the license provides an express grants of patent rights from contributors
- **Limitation :**
 - Liability : The license includes a limitation of liability
 - Warranty : The license explicitly states that it does NOT provide any warranty

Need to share improvements

Conditions :

- Disclose source : Source code MUST be made available when the licensed materials is distributed.
- License and copyright notice : A copy of the license and copyright notice Must be include with the license material.
- Same license : Modifications Must be released under the same license when distributing the licensed materials . In some cases a similar or released license may be used.
- State Changes : Changes made to the licensed materials Must be documented

Work without a license

- By default, all works are under exclusive copyrights of the creator – be it creative work or code
- Unless a license is associated with the copyrighted works, that states otherwise , no one can copy , distribute, or modify the work without being at risk of copyright infringement
- However , in the absence of a license file , one may still grant some rights – this holds in cases where one publishes the source code on a site that requires accepting terms of service
- For example , if the source code is published in a public repository on GitHub, and you have accepted their terms of service , by this you allow others to view and fork your repository

Work without a license

- You do not have to do anything to work without a license
 - However, in the case , one may wish to add a copyright notice and statement - indicating that no license has been included - in a prominent place – e.g. the project READADME
- Moreover, disallowing the use of your code might not be what you actually intent by “ no license”
 - An open source license allows reuse of your code while retaining copyright.
 - Add a [contributors license agreement \(CLA\)](#) to your non-licensed project, so that you maintain copyright permission from contributors, even though you are not granting the same.

Work without a license

To use or contribute to project that has no license associated with it

- Do not use the software : Find or create an alternative that is under an open source license
- Request the Project Maintainers to add a license :
 - Unless the software included strong indications to the contrary, lack of a license is probably an oversight.
 - If the software is hosted on a site like GitHub, open an issue requesting a license
 - If you are sure what license is most appropriate, open a pull request to add a license - with a suggestive license

References and further readings

- [Choose a license](https://choosealicense.com/) <https://choosealicense.com/>
- **Open Source Initiative** <https://opensource.org/>
- **Open Source Resources** <https://opensource.com/>
- **Open Source Guides** (<https://opensource.guide/>)
- **Creative commons** <https://creativecommons.org>
- **GNU** <https://www.gnu.org/>
- **Copyleft** <https://copyleft.org/>



Open Source Software Engineering

SE ZG587

BITS Pilani
Pilani Campus

Kumar Manish
11th Feb, 2023





Session 5 : Open Source Business Model

Recap Previous sessions

- **Intellectual Property Rights and Software Licenses**
 - **Licensing Models in OSS: Copyright, Copyleft, Permissive, Creative Commons**
 - **Choosing Open Source Licensing Models**
 - Option 1: Work with a community
 - Option 2: Keep it simple and permissive
 - Option 3: Need to share improvements
 - Option 4: Work without a license
-

Agenda session 5

- Open Source Business Model
 - By Intellectual Property :
 - Dual Licensing Model
 - Open Core Model
 - By Services and other Business model
 - Open source SaaS model
 - Selling Users advertisement, Services and Merchandise
 - Donation, Crowd Funding and Crowd Sourcing
 - Freemium Business Model
-

Open Source Business Model

- In traditional commercial business model for companies :
 - Sale of Software Products - Product Companies
 - Sale of Software development - Services companies
 - Sale of Support services
- Companies that create and promote Open source software referred to as :
 - **Commercial Open source Software companies (COSS , or)**
 - **Professional Open source Software companies (POSS)**
- **Why ? To solve the challenge of how to make money providing software that is by definition licensed free of charge.**

Open source Business model

- **Business strategies rests on the premise that users of open-source technologies are willing to purchase additional software features under proprietary licenses, or purchase other services or elements of value that complement the open-source software that is core to the business.**
- This additional value can be, but not limited to, enterprise-grade features and up-time guarantees to satisfy business or compliance requirements, performance and efficiency gains by features not yet available in the open source version, legal protection (e.g., indemnification from copyright or patent infringement), or professional support/training/consulting that are typical of proprietary software applications.

Open source Business Model

For Financial viable and successful, POSS companies
Business Model :

1. Selling Intellectual property - the code itself

- Dual Licensing model
- Open – core model

2. Open Software as a Service (SaaS)

3. Selling Services

- Selling Support and Consultancy service – professional services

4. Others :

- Advertising
- Partnership with funding organisations
- Selling branded merchandise
- Freemium

Dual License Model

Dual license products are generally sold as :

- Community version – under GPL
- Enterprise version – under commercial license

Examples : Oracle's MySQL database – dual licensed

- Under GPL : MySQL Classic edition , and MySQL community edition
- Under commercial license :
 - MySQL Enterprise Edition
 - MySQL Cluster CGE, an
 - MySQL Standard Edition

Dual Licensing Model

In this model, the OSS licensed by the POSS company under two licenses :

- **An open source license - Using GPL licenses, and**
- **A commercial license**
- The company generates revenue by selling the OSS under commercial license.

Why would a consumer of OSS buy a license from a company by paying a heavy amount for a software which is already available freely and at a no cost ?

This is necessary when the consumer wants to link his own proprietary software to the OSS, but does not want this to cause its proprietary software to become open source ; as it would under the GPL license

Dual Licensing Model

- According to the GPL license, if someone uses the GPL licensed source code and links it with some other code (dynamic linking or static linking), the linked software also becomes open source.
 - Thus the only way proprietary software vendor can link GPL software without causing their own software to become GPL is by buying a commercial license from the company
- The POSS company provides the same software to the proprietary software company under a commercial license **which saves the proprietary software from becoming open source**

Open- Core Model

- In this model , a core portion of the software, which provides the basic functionality - is made available as open source
- The outer or the other portion or extensions are designed to provide extended or additional functionality and are built over the core.
- The extended portion is licensed under commercial license and sold as a proprietary software.
- Also called Split open source software

Examples : Open-Core Model

Example : Used by - Apache and Mozilla

- GitLab CE (Community Edition) is under a MIT style open source license , while GIT Lab EE is under commercial license
- Initially, Elastic core , which includes Elastic search, Kibana , Logstash and Beats , is under an Apache 2.0 license , while additional plugins are distributed under Elastic's own proprietary license

Open Software as a Service (Open SaaS)

Another way to commercialise an open source project is by using the Open SaaS business model

- **WordPress** and **Sharetribe** are popular examples products that make successful business through the open SaaS model
- software stored in cloud and accessible using a web browser
- Purchase via subscriptions, offering varying levels of services
- No vendor locking
- OpenSaaS' : Coined in 2011 by Dris Buytaert, creator of Drupal.

Selling users : Professional services

- Sell services – such as consulting service , technical support service, or training services.
- Make available only the source code of OSS application or product :
 - Sell executable binaries to the buyers or customers on demand
- Offers the commercial services of compiling and packaging the software
- Example : companies like RedHat, IBM that have successfully used this model
- Additionally , selling good like physical installation media (e.g. DVDs) is another frequently used business model

Selling Users : Advertising – supported software

- Another POSS Business model :
 - For e.g. Google and Mozilla
- Example : Google pay Open source application for allowing whitelisted acceptable Ads display.
 - This is carried out by bypassing browser Ad remover.
- SourceForge, have a revenue model of advertising banner sales on their model

Selling Branded Merchandise

- Some POSS companies like Wikimedia foundation and Mozilla foundation, attract customers by selling branded merchandise articles like T-shirt and coffee mugs.
-

Freemium Business Model

- Feature limited - basic Features vs. Extended features
- Time limited – fixed duration after that paid versions
- Example : widely used in Gaming Industry, Anti-Virus, LinkedIn

References and Recommended Reading:

- How Developers Can Make money with Open Source Projects

<https://rubygarage.org/blog/how-make-money-with-open-source-projects>

- What Motivates a Developer to Contribute to Open-Source Software?

<https://clearcode.cc/blog/why-developers-contribute-open-source-software/>

- How do Open Source Programmers make money

<https://www.thewindowsclub.com/open-source-companies-programmers-make-money>

- https://en.wikipedia.org/wiki/Business_models_for_open-source_software

References and further readings

- [Choose a license](https://choosealicense.com/) <https://choosealicense.com/>
- [Open Source Initiative](https://opensource.org/) <https://opensource.org/>
- [Open Source Resources](https://opensource.com/) <https://opensource.com/>
- [Open Source Guides](https://opensource.guide/) (<https://opensource.guide/>)
- [Creative commons](https://creativecommons.org) <https://creativecommons.org>
- [GNU](https://www.gnu.org/) <https://www.gnu.org/>
- [Copyleft](https://copyleft.org/) <https://copyleft.org/>



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
18th Feb, 2023



Session 6 : Open Source Business Model

- Donations, funding and Crowd - sourcing

Recap Previous sessions

- **How individual / company Can Make money with Open Source Projects ?**
- **How do Open Source Companies and Programmers make money?**

1. Selling Intellectual property - the code itself

- Dual Licensing model
- Open – core model

2. Open Software as a Service (OpenSaaS)

3. Selling Services

- Selling Support and Consultancy service – professional services

4. Others Business Model :

- Advertising, Selling branded merchandise
- Partnership with funding organisations
- Freemium

Dual License Model

Dual license products are generally sold as :

- Community version – under GPL
- Enterprise version – under commercial license
- According to the GPL license, if someone uses the GPL licensed source code and links it with some other code (dynamic linking or static linking), the linked software also becomes open source.
- Thus the only way proprietary software vendor can link GPL software without causing their own software to become GPL is by buying a commercial license from the company

Examples : Oracle's MySQL database – dual licensed

- Under GPL : MySQL Classic edition , and MySQL community edition
- Under commercial license : MySQL Enterprise Edition , MySQL Cluster CGE, and MySQL Standard Edition

Open - Core Model

- In this model , a core portion of the software, which provides the basic functionality - is made available as open source
- The outer or the other portion or extensions are designed to provide extended or additional functionality and are built over the core.
- The extended portion is licensed under commercial license and sold as a proprietary software.
- Also called **Split open source software**

Example : Used by - Apache and Mozilla

- GitLab CE (Community Edition) is under a MIT style open source license , while GIT Lab EE is under commercial license
- Initially, Elastic core , which includes Elastic search, Kibana , Logstash and Beats , is under an Apache 2.0 license , while additional plugins are distributed under Elastic's own proprietary license

Open Software as a Service (Open SaaS)

Another way to commercialise an open source project is by using the Open SaaS business model

- **WordPress** and **Sharetribe** are popular examples products that make successful business through the open SaaS model
- software stored in cloud and accessible using a web browser
- Purchase via subscriptions, offering varying levels of services
- No vendor locking
- OpenSaaS' : Coined in 2011 by Dris Buytaert, creator of Drupal.

Other Business Models ...

- **Sell Services** – such as consulting service , technical support service, or training services. Example : companies like RedHat, IBM that have successfully used this model
- **Selling Users : Advertising** – supported software, Example : Google pay Open source application for allowing whitelisted acceptable Ads display. This is carried out by bypassing browser Ad remover. SourceForge, have a revenue model of advertising banner sales on their model
- **Selling Branded Merchandise** : like Wikimedia foundation and Mozilla foundation, attract customers by selling branded merchandise articles like T-shirt and coffee mugs.

Agenda : session 6

Open Source Business Model

- **Freemium Business Model**
- **Re-licensing under a proprietary license**
- **Donations and funding**
 - **Crowd Funding for OSS**
 - **Crowd Sourcing for OSS**

Freemium Business Model

- **Freemium is combination of two words – free and premium**
- Feature limited - basic Features vs. Extended features
- Time limited – fixed duration after that paid versions
 - In this pricing strategy , the basic version of the software product or service is made available for free, but a premium amount is charged for additional or enhanced features or services that enhance or expand the functional of the free versions
- Source code not available
- Model extensively used in gaming industry

In OSS :

- Companies raises money by selling additional , but optional features , modules , extensions , or plugins to an open source software product

Re-licensing under a proprietary license

- In this model, a software company combines some portion of its proprietary software product, with an existing Open source software (**under a permissive free software license**),
- And relicenses the resulting software Product under proprietary licenses and sells it without the source code or associated freedoms.
- For example : Apple Inc, uses BSD Unix operating system kernel (Under BSD licenses) in the Apple's mac PCs, and sells it as proprietary product.

Revenue model followed by the version control system

- Different version control system provide different revenue models :
- SourceForge : making money Through advertising banner sales on their website
- **Making money by GitHub:**
 - Get funding for your GitHub Repository - interested company might pay you – they would may be using your code or need to promote themselves using your repository
 - Solving open issues in a GitHub repository
 - Finding bugs in GitHub

Donations and funding

- Voluntary Donations and Crowd – funding - other way of getting money for developers

Crowd Funding is the practice of raising funds for a project or event or a venture in small or large amounts - typically, from a large number of people - mostly via the internet.

Actors Involved :

- **Initiator** - Responsible for proposing idea of the project/ venture or event to be funded
- **Individual or group who support the idea**
- And **Moderating Organisation** that provides necessary infrastructure to launch and execute the idea
- Ex : for film making, education, medical expenses, NGOs, travel etc.

Crowd funding for OSS Projects

Open source projects seek support for financial assistance through crowd funding; platform examples

- BountySource <https://bountysource.com/>
 - Bountysource is the funding platform for open-source software. Users can improve the open-source projects they love by creating/collecting bounties and pledging to fundraisers.
 - Snowdrift <https://snowdrift.coop/>
 - Snowdrift.coop is a nonprofit cooperative run by an international team driven by a common goal: To dramatically improve the ability of ordinary people to fund public goods – things like software, music, journalism, and research – that everyone can use and share without limitations.
 - Gunio <https://gun.io/>
-

Crowd - Sourcing

- Crowd sourcing is a participative and online activity
 - In crowd sourcing , an individual or an institution reached out to an individual or group of individual , requesting them to voluntarily undertake the proposed task through a flexible open call.
 - Expectations :
 - The crowd is expected to participate by bringing their **work, money , knowledge and or experiences**
 - **Mutual benefit**
 - **Example for crowdsourcing are Linux, Google Android, Wikipedia**
-

What Motivates a Developer to Contribute to Open-Source Software?



- Improve Coding Skills
- Gain Early Experience
- Increase Community and Peer Recognition
- Greater Job Prospects
- Improve Software on a User and Business Level

How do Open Source Programmers make money



- Companies Pay Open Source Programmers
- Earning By Creating Special Plugins, Etc.
- Earning by Customization of Code
- Earning By Providing Support

References and Recommended Reading:

- https://en.wikipedia.org/wiki/Business_models_for_open-source_software

- **How Developers Can Make money with Open Source Projects**

<https://rubygarage.org/blog/how-make-money-with-open-source-projects>

- **What Motivates a Developer to Contribute to Open-Source Software?**

<https://clearcode.cc/blog/why-developers-contribute-open-source-software/>

- **How do Open Source Programmers make money**

<https://www.thewindowsclub.com/open-source-companies-programmers-make-money>

References and further readings

- [Choose a license](https://choosealicense.com/) <https://choosealicense.com/>
- Open Source Initiative <https://opensource.org/>
- Open Source Resources <https://opensource.com/>
- Open Source Guides (<https://opensource.guide/>)
- Creative commons <https://creativecommons.org>
- GNU <https://www.gnu.org/>
- Copyleft <https://copyleft.org/>



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
24th Feb, 2023



Session 7 : Consumption of Open source in Enterprise

Recap Previous sessions

- **Open Source Business Model :**
 - **Selling Intellectual property - the code itself**
 - Dual Licensing model (community and enterprise version)
 - Open – core model
 - **Open Software as a Service (OpenSaaS)**
 - **Selling Services** : Selling Support and Consultancy service – professional services
 - **Others Business Model** : Advertising, Selling branded merchandise ; Partnership with funding organisations; Freemium

In summary :

How individual / company Can Make money with Open Source Projects ?
How do Open Source Companies and Programmers make money?

Open source in Enterprise :

- How to consume open source software in enterprise ?
 - Is there a proper way to consume open source software in an enterprise ?
 - **WHAT ARE BEST PRACTICES FOR OPEN SOURCE SOFTWARE CONSUMPTION IN ENTERPRISE ?**
-

SOFTWARE CONSUMPTION IN ENTERPRISE

COMMUNITY OPEN SOURCE

Key Characteristics:

- Fast moving with new features released regularly.
- Security and bug/fixes are applied to the most recent stable version.
- Bug fixes or features may be out of sync with your needs if community does not prioritize them.
- Mostly community supported.
- Requires some DIY to bridge functional gaps.

COMMUNITY OPEN SOURCE

:LIMITING FACTOR



The most limiting factors in the wider adoption of community open source system from two reasons:

- Mostly supported by the community and thus require a good amount of DIY, patching and fixing capabilities within enterprise teams.
- Community decides which versions to patch and bug fix, which is mostly the latest version.
- This requires a forced upgrade to latest version in most cases and may incur significant technical debt in order to upgrade the enterprise software built using it. Sometimes this debt is not significant and easy to manage.
- But at scale it can present quite a challenge to review, patch, test and release thousands of binaries without a significant downtime or impact.

: FOR ADOPTION IN ENTERPRISE

DOs

Experiment and evaluate new capabilities and use cases using community open source. For example, evaluate MySQL or Postgres as possible replacement of expensive proprietary databases.

Build initial capability and momentum for a new business case. For example, use API frameworks to evaluate, build and provision REST APIs for partner or B2B integration.

Release as part of a pilot or controlled testing. Might be okay for low-visibility or low-impact production which does not increase security or data corruption risk.

DONT's

Expose to production data in production environment without having a bullet-proof rollback and recovery strategy.

Deploy at scale if you cannot upgrade at community's pace.

Deploy to production if you don't have capabilities to (a) detect security vulnerabilities in the new OSS and (b) patch it in-house.

Deploy to business critical environments, where service, support and recovery guarantees require vendor support.

Key characteristics:

- Supported by a vendor.
- Stable with long supported versions.
- One off bug fixes and feature enhancements are usually supported.
- Feature rich and complete package provided by vendor, to implement and run in production.
- Performance tuned to typical production workloads.

DOs

Deploy to production after thorough testing, performance benchmarking and integrating with support and operational procedures.

Make sure all Dev, Test, QA and UAT environments have the same enterprise versions running as production.

Adopt an enterprise roadmap to map out where the open source version might be able to displace proprietary stacks and still match or exceed feature + performance metrics. This could result in significant cost savings.

DONT's

Select a vendor without understanding what they will and will not support. For example, if the vendor does not provide at least three years of support, be very deliberate about it.

Select a vendor that does not significantly contribute of the community open source equivalent. This is important for various reasons. Notably, vendor's ability to support issues on time and be able to drive the product's direction and provide innovation.



Session 7 : Life cycle and methodologies in Open source software

Open Collaboration Model

- Open collaboration is “a system of innovation or production or development that relies on goal oriented, yet loosely coordinated, participants who interact to create a product (or service) of economic value, and make it available to contributors and non contributors alike.”
 - Additionally, Riehle et al defined open collaboration as collaboration that is based on the following three principles of :
 - Egalitarianism or equalitarianism
 - Meritocracy, and
 - Self –organization
-

Open Collaboration Model

Egalitarianism

- Egalitarianism (from French *égal* 'equal') or equalitarianism is a school of thoughts within political philosophy that builds from the concepts of **social equality**, prioritizing it for **all people**.
 - In the open collaboration model as applied to open source software domain , egalitarianism is used to specify that **everyone can contribute**.
 - Open source software projects are made available over the internet and accessible to the entire human race, equally.
-

Open Collaboration Model

Meritocracy

- Meritocracy advocates a political system in which political power and value of economic goods or individual are measured **on the basis of effort, talent, and achievement**, rather than **the social class to which one belongs to, or the amount of wealth possesses by him/her**
- In the Open Collaboration Model as applied to open source software domain, meritocracy is used **to judge contributions, in a transparent manner, based on the merit or the quality of the contribution.**
- Additionally, all decisions are also made publicly available for evaluation of the judgement taken.

Open Collaboration Model

Self-organization

- Self – organization, in social science , is a process in which some system of order or arrangement emerges through interactions between individuals or parts of an initially disordered system.
- This process may be impulsive and usually not controlled by external agents
- The resulting system of order is
 - Decentralised
 - With all the components of the system distributed , yet connected
 - Robust and self repairing
- In the Open collaboration Model as applied to open source software domain , project communities organise themselves without any external control or influence from a person or process.

Community driven development Model

- Open source software are developed by a large number of developers around the globe some of these developers are independent while some are supported by companies
 - These group of people are often termed as “Community”
 - A community can also be defined as a group of people
 - Which are diverse in nature
 - And engage in sharing ideas , work and experiences
 - Through a common platform for a common cause
 - In the software industry, the word community driven development is broadly used to describe an initiatives in which a group of contributors or developers align their activities together and engage toward the development of open source software
-

Community – driven software development



- Mostly of the open source software development communities structure themselves into groups based on job roles.
 - This is similar to the manner in which professional software organisation organise their employee
 - Structures OSS community comprises of various jobs , roles and multiple sub teams.
 - Developer's group
 - Builders group
 - Testers group
 - Release management group
-

Community – driven software development



- Self driven OSS development communities , often provide flexibility to learn members to shuffle between job roles.
- These communities are open to customers or end users , inviting them to join the community and contribute to the project
- Agile mindset, self motivated, self organised

Developers' group in OSSE

- Software Design – define the architecture and behaviour of the product
- Coding – implement the design
- A community driven software development project typically starts with the aim to address a typical or selected problem
- These problem (or pain points) specially the requirements of the software
- The designers group is responsible for creating software design which should be effective and appropriate
- The design should be self explanatory, easy to understand and created using standard notations like UML

Developers' Group in OSSE

- The next steps is to choose an appropriate language for working the source code
- Like in many other software development projects , in a community driven development also , the transition between design phase and coding phase typically tends to overlap with each other
- Developers usually begins implementing sub parts of a module or system, while other modules are still in their deign phase
- While this approach of running parallel phases (both design and coding) saves time, it might result in duplication of efforts in the event of a change in design.

Builders' group in OSSE

- The builders' group is responsible for taking care of the software build process which comprises of the following two tasks :
 - Compiling all source code in to object modules
 - Linking the object module code in to one whole
 - Software building is continuous process , since developers continuously modify , add or delete software artefacts , which are placed in a common projects repository
 - As a result , the build process also takes up a continuous form – since it is necessary to reflect these changes in the final build.
 - The latest code needs to be recompiled and linked regularly
-

Builders' group in OSSE

- In order to speed –up this process , the underlying build process is usually designed to recompile only those source code files which were modified since the previous build was released
- For the remaining unchanged artifacts , the older compiled versions are picked and released.
- A large number of continuous integration and continuous deployment (CI/CD) tools exits that help to speed up to the build and release process .
 - Jenkins, TeamCity, Gitlab, CicleCI, TravisCI

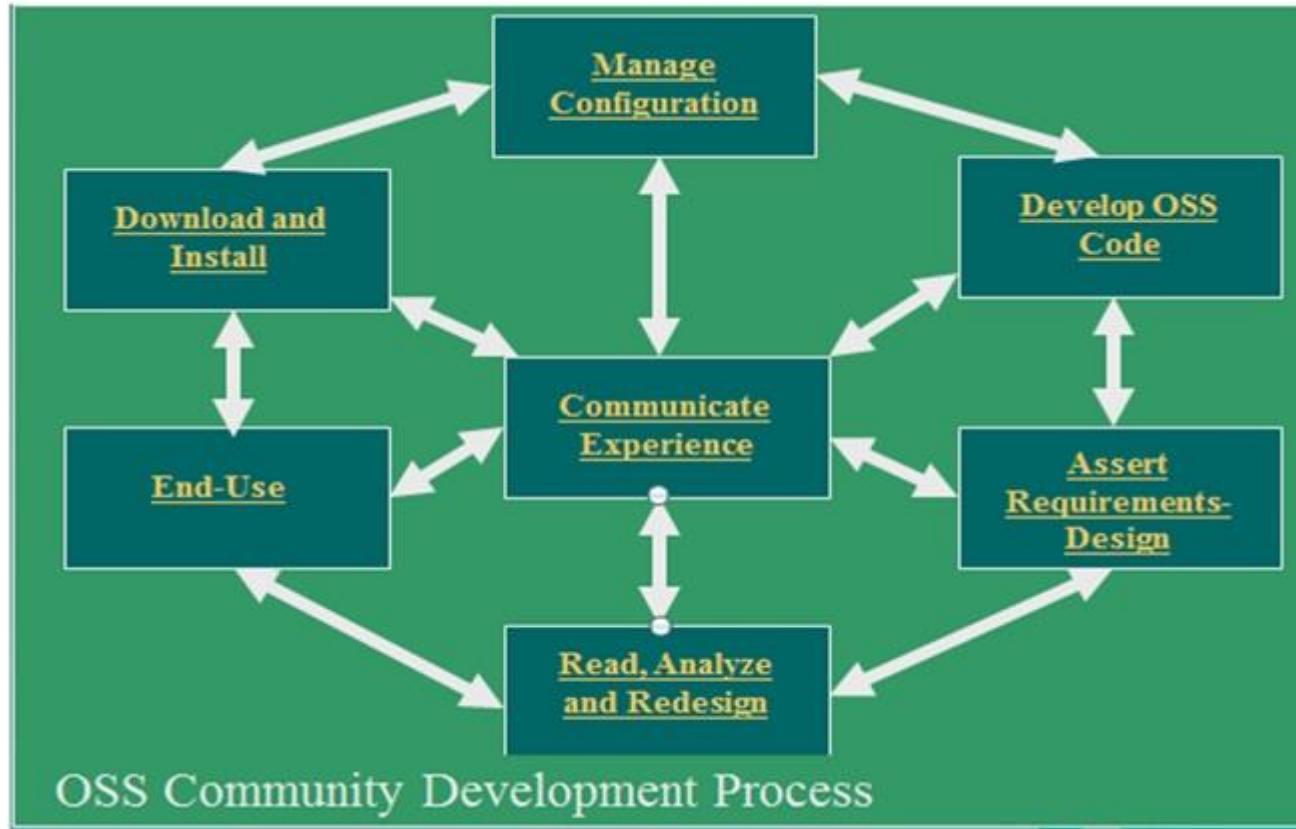
Testers' group in OSSE

- Testing in the OSSE model is carried out by the testers' group of the community

May support :

- Mailing list, Discussion boards, Bug reports
- The testers group carries out testing , generate bugs reports which are shared with the developers / community – fixes are provided
- This cyclic process continues until the project community feels that the implementations is stable enough - and then it is released - development still continues

Community Development process



Community Development process

- **Assert Requirements Design**

This phase is all about the need for the software that is being proposed to the community for development and also the requirements during the development process are kept in mind in this phase. Along with it the design or the approach to be followed during the development is also a part of this phase.

- **Develop OSS Code**

The requirements are fulfilled; the need has been understood and the design is ready. Now, it's time for the contributors to start developing the backbone or the skeleton of the software by the means of Coding.

Community Development process

- **Manage Configuration**

Once the basic OSS Code is developed it is necessary to provide an initial configuration too is that its integration works perfectly fine along with all the features provided with that particular version.

- **Download and Install**

Once the initial version is ready and configured properly it is ready to go in the markets for general use.

- **End-Use**

The users who require the software to fall into this phase and use the OSS daily to provide experience and feedback to the developers or for personal benefits.

Community Development process

- **Communicate Experience**

Once it is deployed in the market, the users share their experience with the OSS and give feedback, suggestions, and reviews on the functions that are good to be intact and also on the features that could be enhanced or added in the later versions of the software.

- **Read, Analyze and Redesign**

Once the feedbacks are registered, it all comes back to the developers to work on the feedbacks, keep updating their Software and also track control of version.

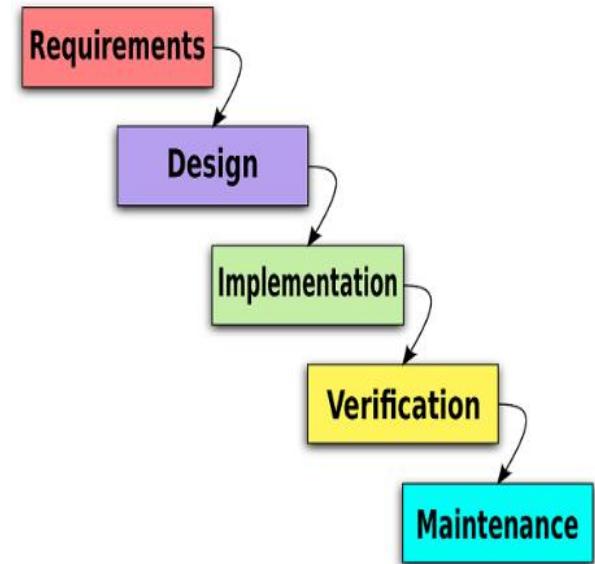
Open Source Development Model **process**



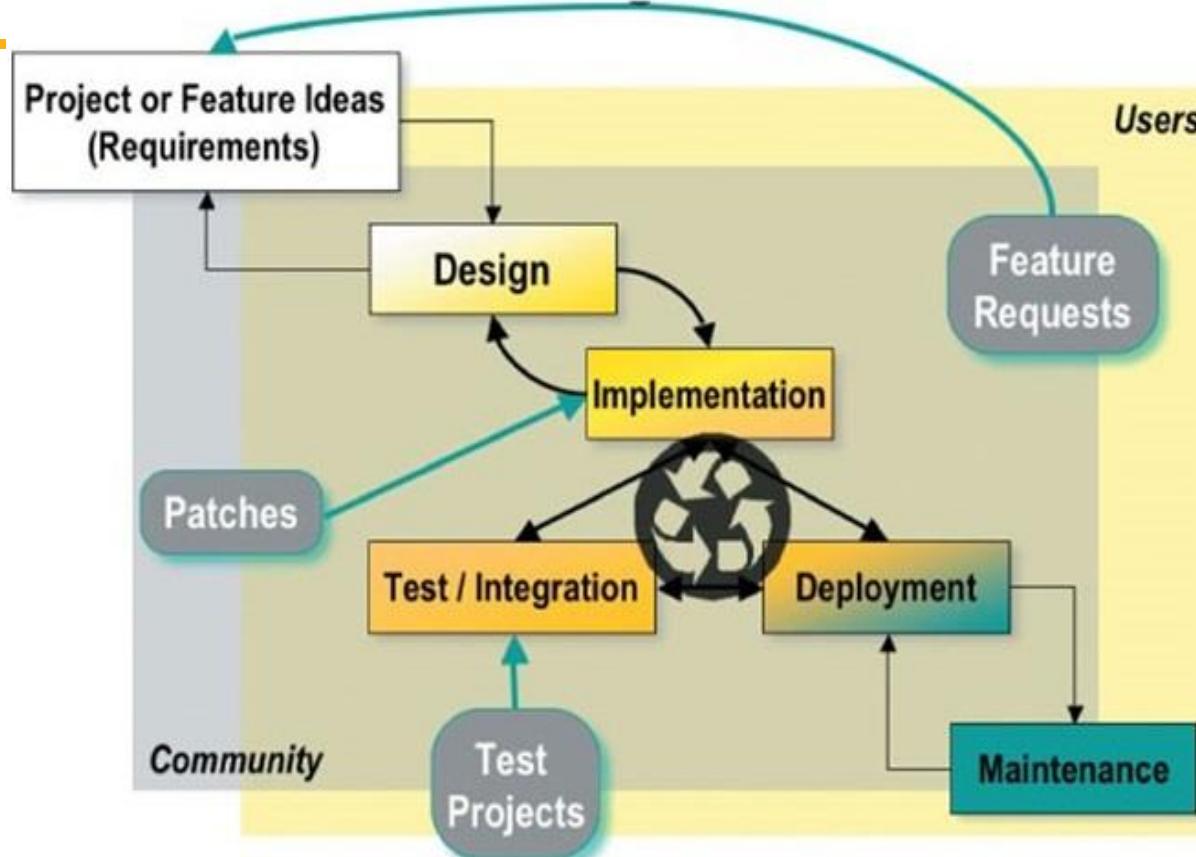
- **Open Source Development Model** Involves an interconnected OSS Community Development Process in which each stage or phase plays a vital role in building the ethics of the community, keeping contributions of each developer in mind, working with the latest technologies, keeping a track on the version control system and fixing the bugs in the software.

Open source software development : Process Model

- Waterfall Model / Linear development model
- OSSD : idea for a new project , or new features → design for proposed solutions – POC → implementations
- The moment the software runs (partially , mostly), it is released as development release (even though it may contain known and unknown bugs)
- In alignment of Philosophy : **“Release early, release often”**

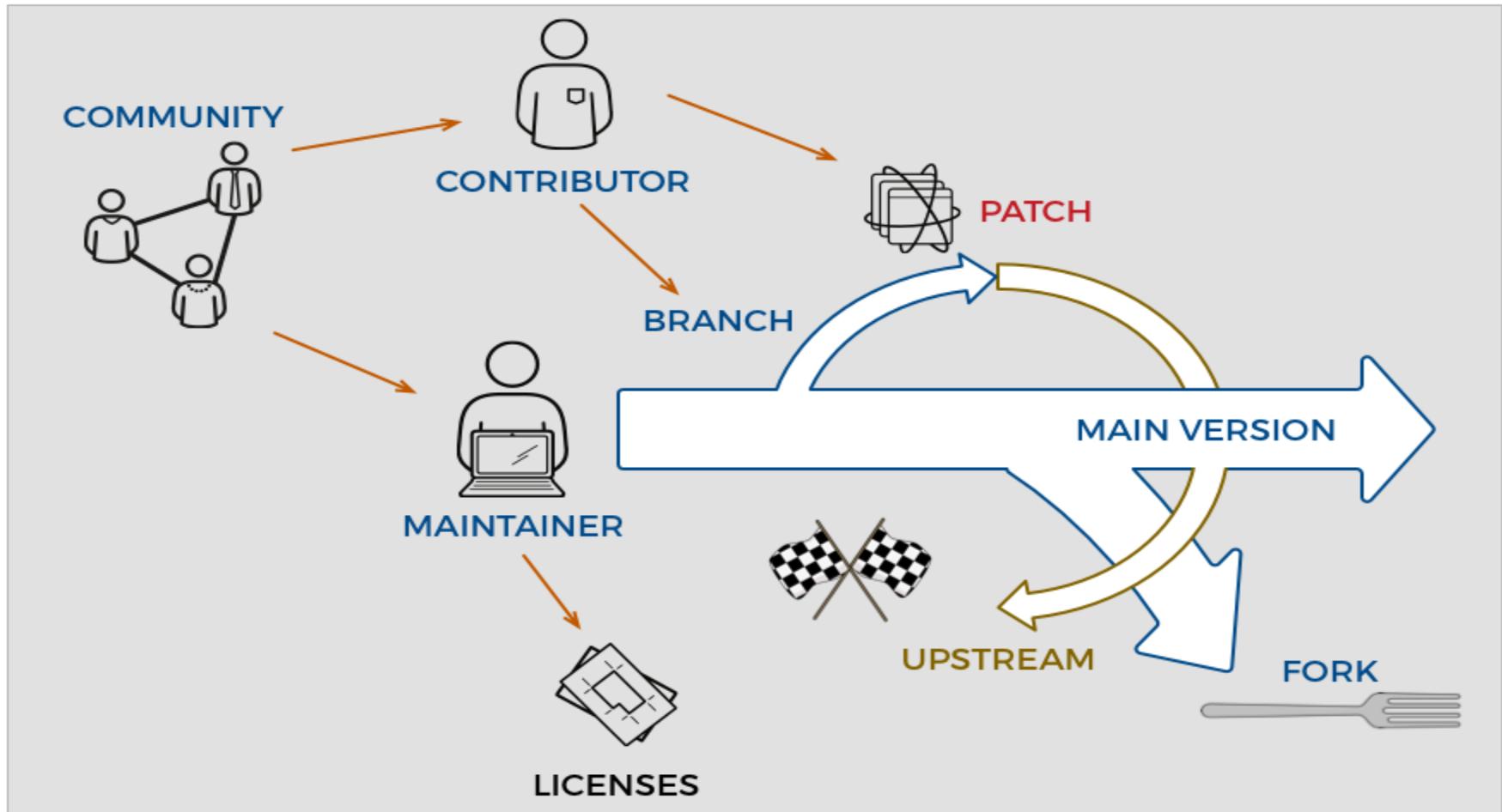


Open source software development : Process Model



- This model is followed when a request is made to the developers to build a product to best suit their needs to which the developers keep in mind certain factors and phases to deliver a fully functional and a product with no compromises to the client.

Open source software



References

<https://www.stackinnovator.com/blog/what-is-opensource-anyway/>

HOW TO CONSUME OPENSOURCE SOFTWARE IN ENTERPRISE?



Open Source Software Engineering

SE ZG587

BITS Pilani
Pilani Campus

Kumar Manish
4th March, 2023

Quick check

- Can open source software be used for commercial purposes ?
 - Can I restrict how people use an open source licenses programs ?
 - Can I call my programs : open source “ even if I don't use an approved licenses ?
-

Recap :

- **Open source** : Collaboration open and Source freely available
 - Freely : Share, Adapt, Modify and Collaborate
- **OSS** : Source code Released under a **License** in which the **copyright** holder grants users the rights to **use, study, change, and distribute** the software and its source code to anyone and for any purpose.
- **Key Principle of OSS** : Openness, Transparency, Collaboration, Expectations of Community
 - Release Early and Often : Rapid prototypes and iterative approach
- **Free software**” means software that respects users' freedom and community. It refers to free use of software and not price
- **OSI** : Definitions, Standard Requirements, Licenses
- **FSF** : To promote computer user freedom and defend the rights of all software users
 - Richard Stallman in 1983, by launching **GNU Project**

Recap :

Free software Four essential freedoms

- **A program is “free software” if its users have the following :**
 - Freedom 0 : The freedom to run the program as you wish, for any purpose
 - Freedom 1 : The freedom to study how the program works, and change it as per your requirements
 - Freedom 2 : The freedom to redistribute copies so you can help others
 - Freedom 3 : The freedom to distribute copies of your modified versions to others
 - **Pre-requisite “Access of the source code”**
- **Legal considerations :**
 - The owner of the free software does not have power to withdraw or invalidate the license, or add additional restrictions to its terms and conditions,
 - **License terms should be permanent and irrevocable**
 - **License must not restrict other software**
 - **License must be technology neutral**

Recap :

	Free software	Open source Software	Freeware	Public domain software
Definition	“FREE” is a matter of liberty , not price	“OPEN” does not just mean access to the source code; more about collaboration	Free refers to price , while freedom of the use is restricted by creator	PUBLIC domain belongs to the public as a whole
Philosophy	Social movement	Development methodology	Marketing goals	Copyright disclamation
Rules	Four freedoms	Open source Initiatives		Creative common Org
Free of charge	Not necessary	Not Necessary	YES	YES
Copyright law	YES	YES	YES	NO
Examples	Linux, Ubuntu , MySQL, Apache	Linux, Ubuntu , MySQL, Apache	Skype, Adobe acrobat	SQLite

Recap :

- Understanding Intellectual property Rights (**IPR**) to the software industry
 - **Patents** – used to protect functional features , like hardware configurations etc
 - **Copyrights** – used to protect works of authorship, like source code , diagram etc
 - **Trade secrets** – used to protect internal business secrets, like business and pricing models
 - **Trademarks** - used to protect brand recognition, through logo etc
- All the people involved in a development project, may claim for the ownership of the IPR for the various artefacts developed as a part of the project.
- **Understanding Software Licenses**
 - **Licensing Models in OSS:**
 - Copyright, Copyleft, Permissive, Creative Commons

Recap :

Copyleft or Protective Licensing Model :

- To make a work (or program) free to use, modify , adapt or extend.
- Aligned with the **four essential freedoms**
- No re-licensing allowed, No commercial usage allowed
- All modified and extended version of the work (or program) should be free as well - hence called **protective licenses**
- All derivative works should be attributed to the creator, open sourced and copyleft

Variants of Copyleft : **Strong and Weak copyleft** :

- The strength of the copyleft license is decided based on the extent its provision are imposed on the derived works
- Most commonly, weak copyleft licenses are used to create **Software libraries**
- Help software to link to the library and redistributed without requirement for the linking software to also be copyleft –licensed

Examples :

Strong Copyleft :

GNU general public License

Weak copyleft

GNU lesser General Public License –

Mozilla public license

Recap

Permissive Licenses

- Free software licenses with only minimal restrictions on how the software can be used , modified and redistributed (also called Berkeley software distribution : BSD - like or style licenses)
- Rules for usage - what ever user wants, but with few restrictions – derived works must be attributed to the creator
- Source code need not be open or made available in the public domain
- Re-licensing allowed - derivative works can be release under any other licenses or used as proprietary products
- Allows commercial usage
- Examples : GNU All permissive License, MIT License, BSD licenses, Apple Public source licenses. Apache license

Recap

Copy left (Protective Licenses)	Permissive Licenses
<p>Publication of software code of all modified versions or derived works under the original copyleft licenses / protective licenses</p>	<p>Provides No guarantee that derived works of the software will remain free and publicly available ; generally requiring only that the original copyrights notice be remained</p>
	<ul style="list-style-type: none"> - Hence derived works , or future versions , of permissively – licenses software can be released as proprietary software - Permissive licenses offer highly extensive license compatibility as compared to copyleft licenses - Wider adaptability in the open source community

Recap

Choosing Open Source Licensing Models : Based on Permissions, Distribution, Modification, Use.

- Option 1: Work with a community –
 - Contributors License Agreement (CLA)
 - Simple CLAs vs Detailed CLAs
 - Individual CLAs vs Corporate CLAs
- Option 2: Keep it simple and permissive - Choose the MIT License ,
- Option 3: Need to share improvements
- Option 4: Work without a license

Why to choose Open Source Licensing Models?

- To protect works contributed in the open source domain
- To protect contributors and users from any copyright infringements
- To Interested developers or contributors or business will not touch a project without a license protection.

Recap

Open Source Business Model

1. By Intellectual Property :

- Dual Licensing Model –
 - Community version – under GPL
 - Enterprise version – under commercial license
- Open Core Model : Core portion of the software, which provides the basic functionality is made available as open source

2. By Services and other Business model

- **Open source SaaS model** – example WordPress, Sharetribe...
- **Selling services** : such as consulting service , technical support service, or training
- **Donation, Users advertisement, and Merchandise**
- **Crowd Funding and Crowd Sourcing**
- **Freemium Business Model** - Feature limited - basic Features vs. Extended features Time limited – fixed duration after that paid versions

References and further readings

- [Free software foundations https://www.fsf.org/](https://www.fsf.org/)
- [GNU https://www.gnu.org/](https://www.gnu.org/)
- [Open Source Initiative https://opensource.org/](https://opensource.org/)
- [Open Source Resources https://opensource.com/](https://opensource.com/)
- [Open Source Guides https://opensource.guide/](https://opensource.guide/)
- [Creative commons https://creativecommons.org](https://creativecommons.org)
- [Choose a license https://choosealicense.com/](https://choosealicense.com/)
- [Copyleft https://copyleft.org/](https://copyleft.org/)

References and Recommended Reading:

- How Developers Can Make money with Open Source Projects

<https://rubygarage.org/blog/how-make-money-with-open-source-projects>

- What Motivates a Developer to Contribute to Open-Source Software?

<https://clearcode.cc/blog/why-developers-contribute-open-source-software/>

- How do Open Source Programmers make money

<https://www.thewindowsclub.com/open-source-companies-programmers-make-money>

- https://en.wikipedia.org/wiki/Business_models_for_open-source_software

- Bounty Source <https://bountysource.com/> for Cloud Funding

- Gunio <https://gun.io/>



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
18 March, 2023





Session 9 : Life cycle and Methodologies in Open Source Software

Agenda

Lifecycle and methodologies in Open Source Software

- Open Collaboration Model
- Community Driven Development Model
- Open Source Software Development Process Model
 - Unique characteristics of the Open Source Software Development Process Model
 - Comparing OSS development methodologies with traditional methodologies

Open Collaboration Model

- **Open collaboration** is “a system of innovation or production or development that relies on **goal oriented**,
- yet loosely coordinated,
- participants who interact to create a product (or service) of economic value,
- and make it available to contributors and non contributors alike.”

Open Collaboration Model : principles



- **Equalitarianism :**
 - To specify that everyone can contribute.
 - social equality, prioritizing it for all people.
- **Meritocracy :**
 - To judge contributions, in a transparent manner, based on the merit or the quality of the contribution.
- **Self –organization :**
 - communities organise themselves without any external control or influence from a person or process.

Community Driven Development Model

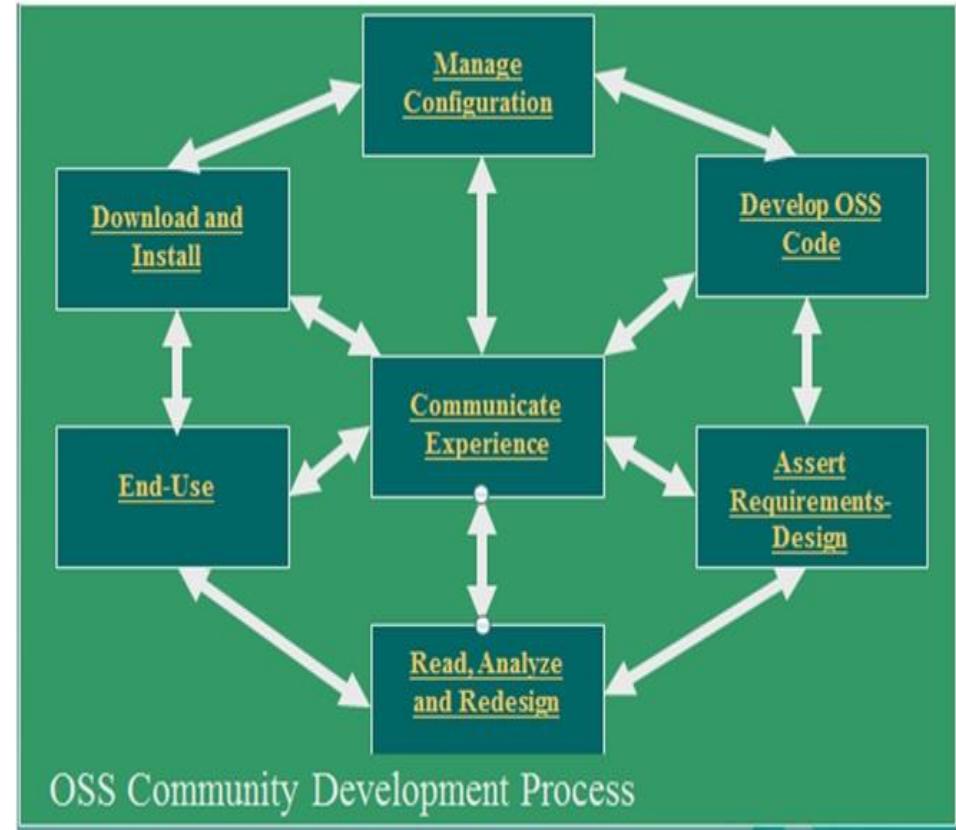
- A group of contributors or developers align their activities together and engage toward the development of open source software
- These group of people are often termed as “Community”
- A community can also be defined as a group of people
 - Which are diverse in nature
 - And engage in sharing ideas , work and experiences
 - Through a common platform for a common cause
- Community comprises of various jobs , roles and multiple sub teams : **Developer's group, Builders group , Testers group**
Release management group

Community Driven Development Model

- **Self driven** OSS development communities , often provide flexibility to learn members to shuffle between job roles.
- These communities are **open to customers or end users** , **inviting them to join the community** and contribute to the project
- **Agile mindset, self motivated, self organised**

Community Driven Development Process

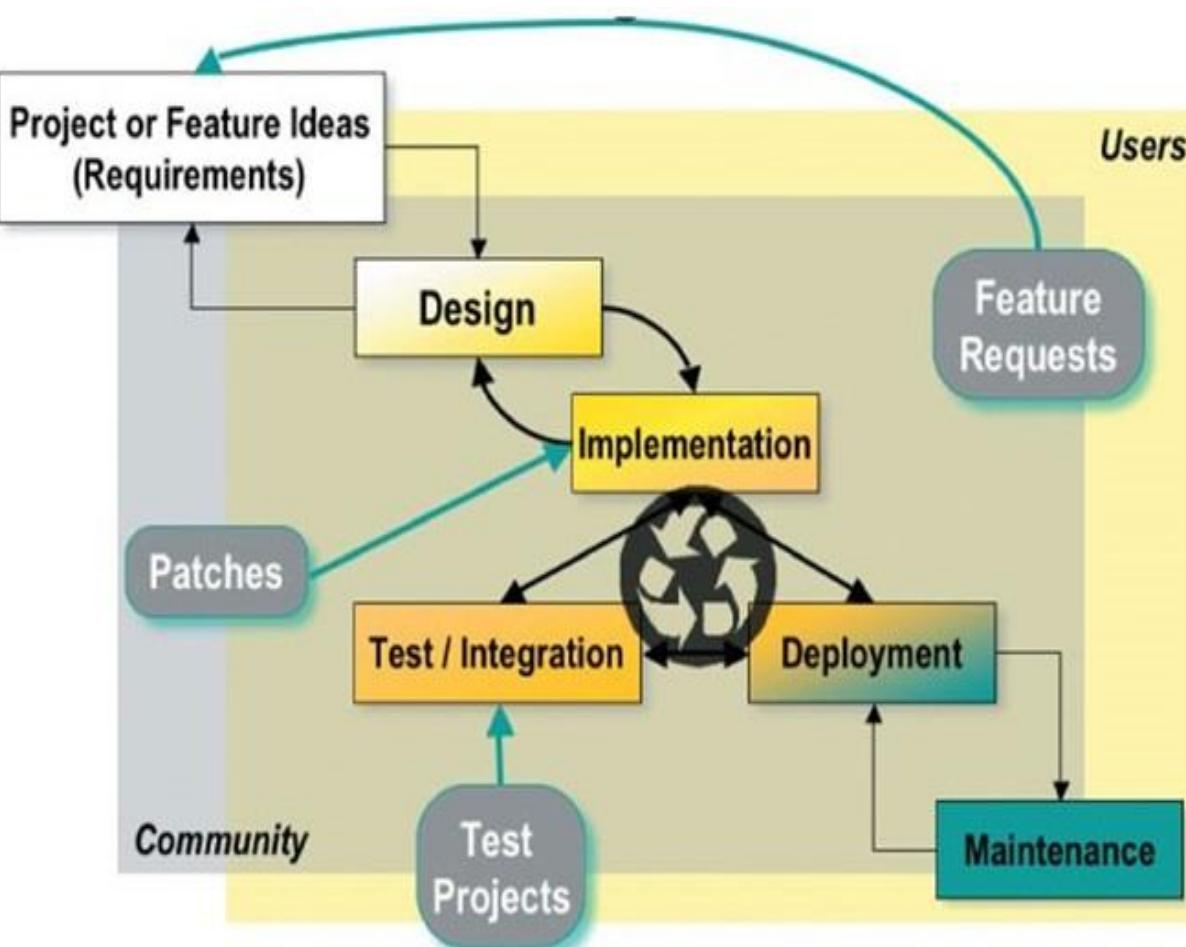
- Requirements & Design
- Develop Code
- Manage Configuration
- Download and Install
- End-Use
- Communicate Experience
- Read, Analyze and Redesign



Open Source Software Development Model

- OSSD Model is different from the traditional waterfall or cascading model of software development.
- Open Source Development Model Involves an interconnected OSS Community Development Process in which each stage or phase plays a vital role in building the ethics of the community, keeping contributions of each developer in mind, working with the latest technologies, keeping a track on the version control system and fixing the bugs in the software.
- The moment the software runs (partially , mostly), it is released as development release (even though it may contain known and unknown bugs) .
- In alignment of Philosophy : “**Release early, release often** “

Open Source Software Development Model



Release early, release often

- Idea for a new project or new feature or functionality
- Design for proposed solution
- Implementation by running (partially or fully)
- Development release (even bug)

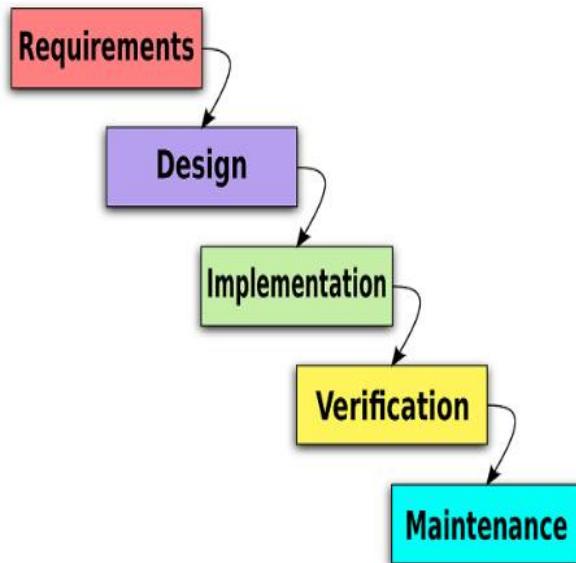


Comparing OSS development methodologies with traditional methodologies - waterfall / agile

OSSD Vs. Waterfall Model

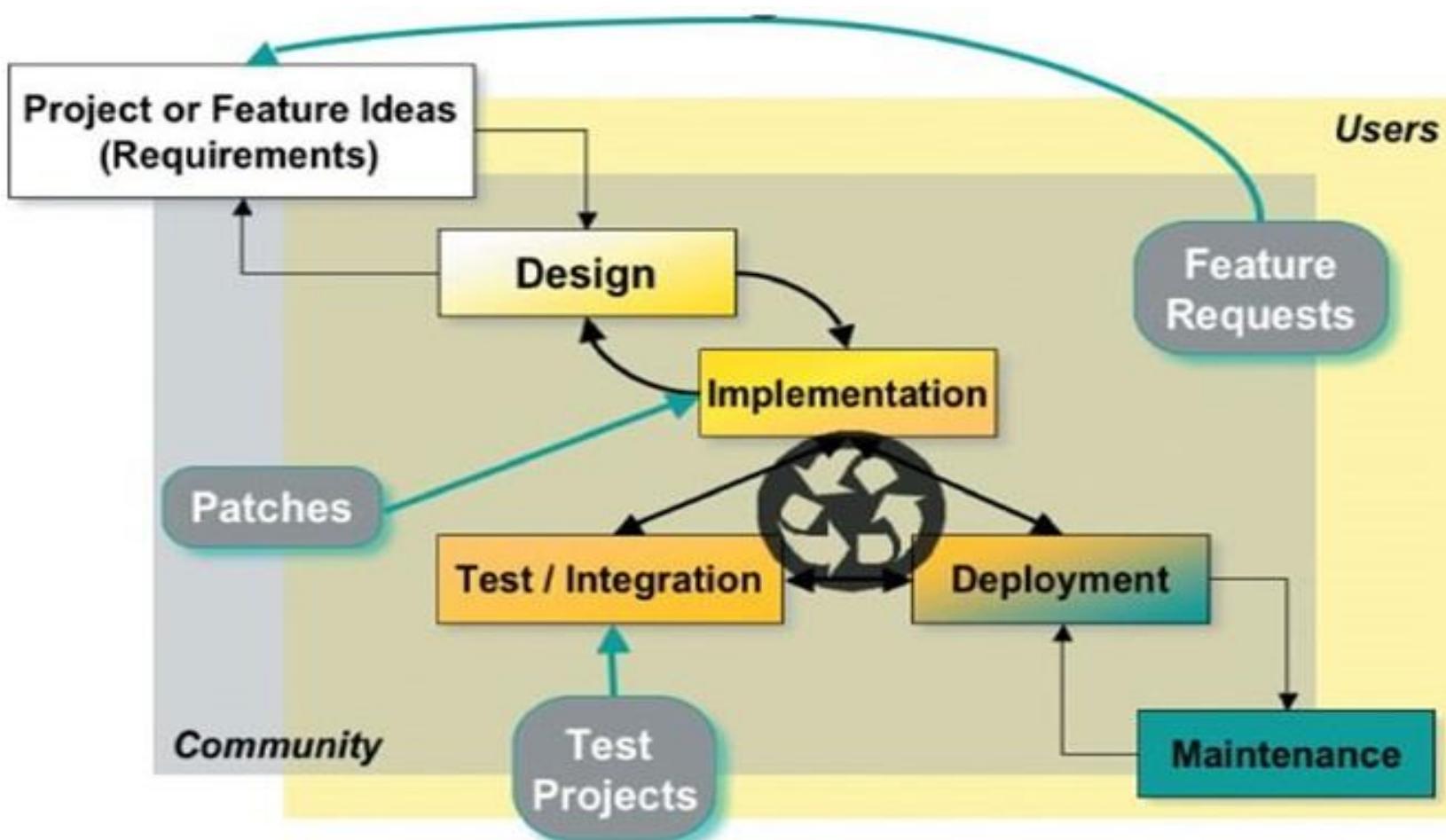
Traditional software development : Waterfall

- Activities arranged in a linear fashion ,
- works starts on a particular activity only after the previous activity is completed



OSSD model is different from the traditional waterfall or cascading model of software development .

Open source development Model

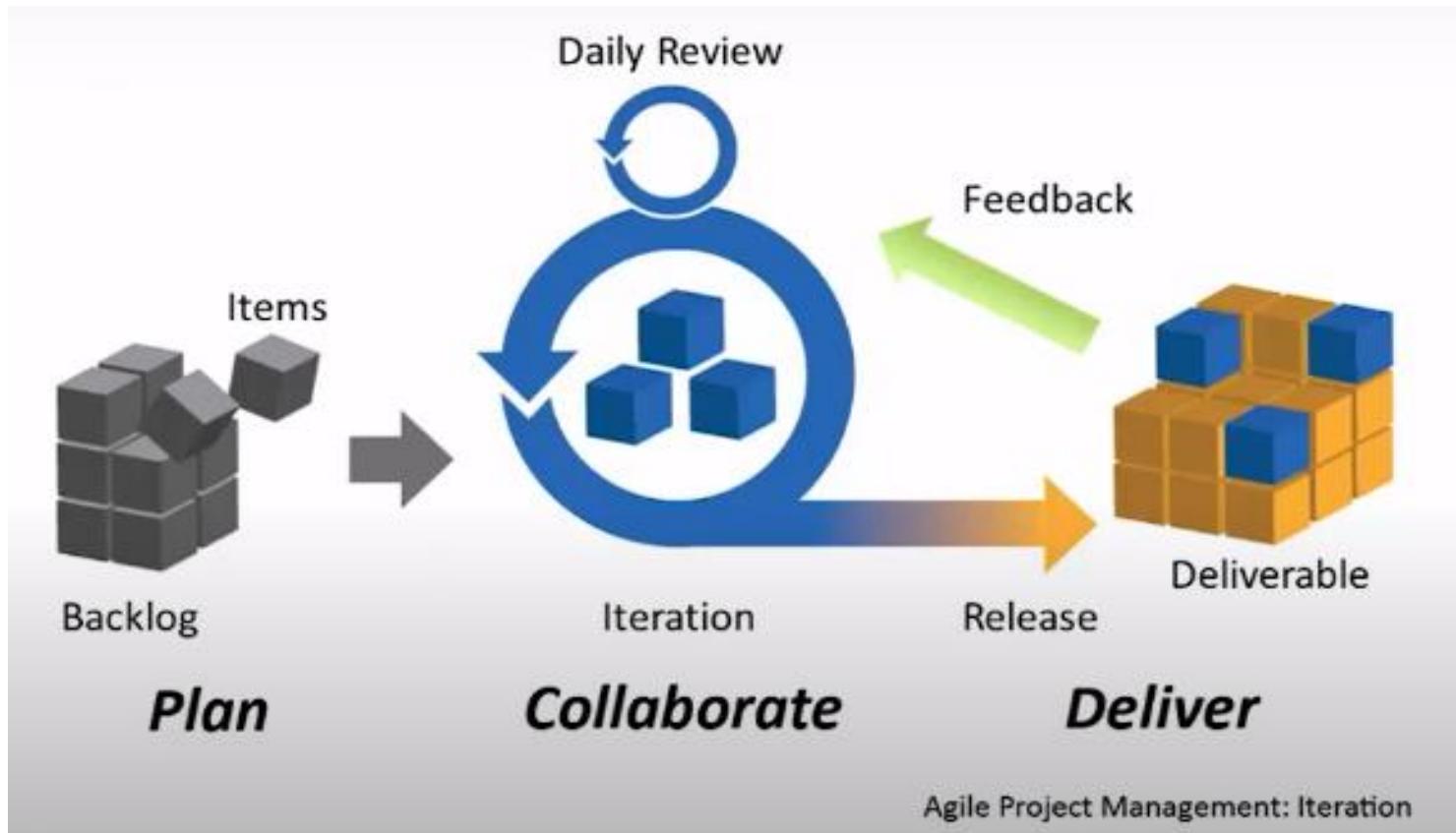


Agile Methodologies

Agile Methodology emphasizes :

- Small design steps
- Incremental development , and
- Frequent customer interactions and frequent release
- The **developers and customer together decide on the set of features and capabilities** - to be incorporated in the next increment of the software
- Each Iteration of development is of **few weeks** only
- The software is then **released to the Customers** ; who test and use the same and find bugs , and suggest fixes and updates
- Agile and OSSD models share many principle and value however ,there are also **contrasting differences in the philosophy** behind them

Agile Methodologies



Agile vs. OSSD Methodologies

As per Agile Manifesto :

- “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software “

As per OSSD Model :

- The **concept or analogy of customers** does not exist in open source software . There are **contributors and users of the projects / product**.
- The **user community , indirectly , serves as the customers**; frequent releases are made - in order to promote peer review, continuous and increased end user involvement.

Agile vs. OSSD Methodologies

As per Agile Manifesto :

“ Welcome changing requirements , even late in development . Agile processes harness change for the customer ‘s competitive advantage “

As per OSSD model :

- **Open source projects are not customer driven** ; and hence often **resist incorporation of major changes** in the requirements.
- However , there always exist the **possibility of incorporating major changes by forking the project in to as separate repository and making changes** - done in case a larger set of community feels the need for the same.

Agile vs. OSSD Methodologies

As per Agile Manifesto,

“Deliver working software frequently , from a couple of weeks to a couple of months , with a preferences to the shorter time scale”

As per OSSD model,

- Open source follows the philosophy of : **release early , release often**
- Delivering of code cycle in case of open – source is much shorter – usually every night.

Agile vs. OSSD Methodologies

As per Agile Manifesto

“Business people and developers must **work together daily throughout the project**”

As per OSSD model ,

There does not exist the concept of “ **Business people** “ in case of open – source projects ;

However , **end-users who participates in the project serve the same role.**

Agile vs. OSSD Methodologies

As per Agile Manifesto ,

- **Build projects around motivated individuals . Give them environment and support they need , and trust them to get the job done”.**

As per OSSD model :

- Participation in open source projects is completely **voluntary; self-motivated and self-driven people join open source community , out of interest and hence motivation is guaranteed**
- Open source projects use selected tools for project management , version control , bug and issue tracking and discussions forums.

Agile vs. OSSD Methodologies

As per Agile Manifesto

- The most efficient and effective method of conveying information to and within a development team is **face to face conversation.**“

As per OSSD model :

- **This is a major point of differences between Agile methodologies and open source**
- Open source projects lay complexly emphasis on written communication over face –to-face communication
- Additionally open source projects can be widely distributed , and do not require collocation

Agile Vs. OSSF Methodologies

As per Agile Manifesto :

- At regular intervals, the team reflects on how to become more effective , and then tunes and adjust its behaviour accordingly.

As per OSSD Model :

- **Self reflection is not** a major activity carried out in most of the open source projects.
- However , some of the mature open source projects tends to evolve and implement some **governance mechanism**

Agile vs. OSSD Methodologies



As per Agile Manifesto

- “Working software is the primary measure of progress”

As per OSSD Model :

This principle perfectly holds true for the open source software development model as well.

Agile vs. OSSD Methodologies



As per Agile Manifesto

- “Agile process promote sustainable development . The sponsors , developers and users should be able to maintain a constant pace indefinitely. ”
- As per OSSD Model :
 - The spirit behind this principle is in some manner embraced by open source;

Agile vs. OSSD Methodologies

As per Agile Manifesto :

- “ Continuous attention to technical excellence and good design enhances agility”

As per OSSD Model :

- Open source model emphasize heavily on good design , technical excellence and well written documentation

Agile vs. OSSD Methodologies

As per Agile Manifesto :

- The best architecture , requirement and design emerge from self –organising teams

As per OSSD Model,

- Although, open source projects might not state this explicitly , but the underlying philosophy behind the open source projects makes this being true for these projects.
- **Self organising and self motivating community does results in some of the best products , with best architecture and designs**

Agile vs. OSSD Methodologies

As per Agile Manifesto

- “**Simplicity – the art of maximizing the amount of work done is essential**”

As per OSSD Model,

- since open source projects mostly rely on **Voluntary commitments, and not contractual commitments** , hence the **amount of work done is never a point of major concerns**
- The amount of work to be done wholly **depends on the belief of the individual developers**.

Conclusion

To summarise :

- While both agile and open source methodologies embrace a number of principles and values , some of which are parallel , but some are intersecting as well.

However , both of them :

- Involve continuous interactions with users during all phases of the development;
- Value good design and effective documentation
- Engage self organising and self motivated individuals ; and
- Strive to develop software that is tailored especially for a class of users / customers



Unique characteristic of the OSS Development Process Model

Unique Characteristic of OSS Development model

Release early, release often : The code is made public and a development version of the software is released **as soon as the same is available** :

- Does not wait for a fully working version
- **Advantage :** allows for peer review , early bug fixes, comments , Improved code quality, incremental changes - easy to understand and test and correct

Peer Review : Exhaustive review by community

Small, incremental changes : For a larger period of the development cycle, the changes are in the form of small code patches which are easy to understand, test and resolve

- Less likely to introduce cascading or unintended consequences ,
- Beneficial since help in focusing more on testing phase,
- Small changes are less likely to introduce cascading or unintended consequences

Unique Characteristic of OSS Development model

Highly secure code : OS community considers security as very important aspect and any code that raises security concerns is flagged and not included in the project

Continuous quality improvement : Exhaustive peer review and early identifications and fixing of bugs leads to improved code quality

Test projects : Large open source projects also create separate test projects consisting of all large numbers of test suites and make use of automated testing tools.

East user involvement : The end user are continuously involved in all phases of the development model

Recommended Practices

Corporates can benefit from adopting certain practices :

- Increased team communication
- End user feedback
- Peer review
- Release early and often
- Transparency
- Good code designs

References



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
01 April, 2023





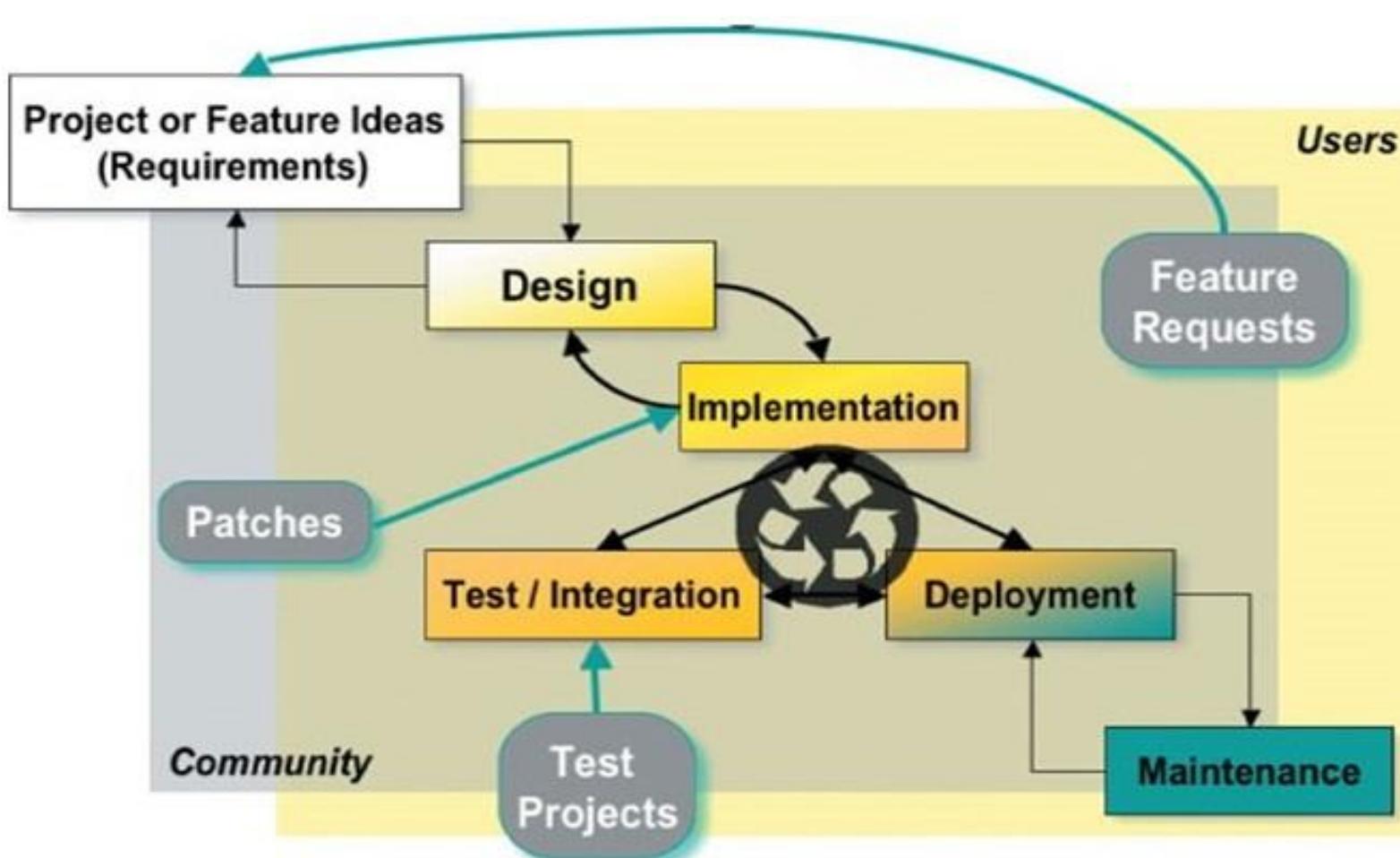
Session 10 : Contributing to Open Source Software Projects

Recap

Lifecycle and methodologies in Open Source Software

- **Open Source Software Development Model**
 - Comparing OSS development methodologies with traditional methodologies – Waterfall, and Agile
 - Unique characteristics of the Open Source Software Development Process Model

Open Source Software Development Model



Unique Characteristic of OSS Development model

- **Release early, release often**
 - **Peer Review**
 - **Small, incremental changes**
 - **Highly secure code**
 - **Continuous quality improvement**
 - **Test projects**
 - **End user involvement**
-

Agenda

- **Contributing to Open Source Software Projects**
 - Contribution models and roles in OSS
 - Familiarizing with the open source software ecosystem
 - Starting your own Open Source Project
 - Best practices in running/ managing OSS project

Contribution Roles

- There are Number of roles that exit in any open source project.
- Depending on your interest, you may choose to contribute as any one or many of them .
- **Project Leader / Main Developer :** A leader acts like the CEO of the company and is responsible for all major decisions. Smaller projects may have several developers in the lead role, similar to board of directors in an organisation
 - You may acquire the role of a project leader , when you start your own project or open source an existing projects seeking contributions.
- **Programmer :** Programmers are mainly responsible for implementation of features and fixing.

Contribution Roles

Designer : UI/UX or web designers or art work designers

Document writer : project documentation has gained importance since , for the not so skilled end users , installing and using open source software is difficult. Detailed documentation helps novice users as well as beginner contribution in understand the project scope and code.

Translator : A large number of open source projects welcome contributions who are willing to translate the project manuals in their native language.

Active user : They form the base of the pyramid I use the software ; provide feedbacks and customised the product.

How to contribute other than code to Open source software projects ?

- Create , edit or translate the documentation of existing Open source projects
 - Contribute Tutorials on how to sue the project
 - Create newsletter
- Plan events
 - Help in organizing workshops , meetings , conferences
- Refine or contribute designs
 - Refine project navigation/ menus
 - Create a style guide for visual layout
- Contribute in organising project
 - Restructure project layout
 - Track issues, Link duplicate issues , suggest new issue labels
- Contribution to the forums :
 - Answer question about the project
 - Help moderate the discussion boards or conversation channels



Familiarizing yourself with an Open source Project

Familiarization yourself with an open source project

Pre-requisite before starting to contribute :

- Learn a Programming Language
 - Get started with any language of your choice
 - Do not emphasize on learning a new language for contributing to open source
 - Go with what you already know
- Familiarize yourself with version control system (VCS)
 - There are many version control system such as Git, CVS and SVN
 - Git is the most popular and most widely used version control system in the industry

Search an Open source project (Active) in the chosen Language

- Navigate GitHub repository of the selected project
- Read the documentation
- Go through the license information
- Check if the project has a CLA that need to be signed
- Join the IRC (internet Relay chat) channel (discord/ slack)

Familiarizing yourself with an Open source

- Start with going through beginner-friendly issues search for first –timer issues as mentioned above
 - Try to work on as many issue as you can either across projects or for a single project
 - You can also create issues after running the application locally

To search for easy or beginner level issues :

- Navigate to the repo of the project
- Different VCS follow different conventions for making issues
- Search for issue which are marked as follows :
 - Good First issue is : issue is : open label : good first issue
 - Easy issue is : issue is : open label : easy
 - First Timer issue is : issue is : open label : first –timers-only

Familiarizing your self with an Open Source projects



Understand the anatomy of project

- Author, owner , Maintainers . Contributors , event Organizers
- Read through these documents :
 - **LICENSE** every open source project consists of a file that describe the license used by it , if the project does not have a license , it is not open source
 - **README** : It explains the usefulness of the project and how it can be used . Might also contains instructions about how a new comer can start contributing
 - **CONTRUBUTING** : explains the type of contributors that are needed and how the entire contributions process works . This document might NOT be present in may projects, yet its presence indicates that it is a welcoming projects, inviting contributors.

Familiarizing yourself with an open source project



- **CODE_OF_CONDUCT** : Explains the basic rules for participants behaviour and helps in facilitating the projects welcoming environment
- Other documentation : The project might contain some additional documentations , such as tutorials, walkthrough or governance policies etc.

Familiarize yourself with the communication tools used in the project:

- Issue tracker : used to discuss issues related to the project
- Pull request : used to discuss and review changes that are in progress
- Discussions forums or mailing List : channels used by some projects for conversational topics
- Synchronous chat channel : These are used for casual conversation, collaboration and quick exchange of messages.



Starting your own Open Source Project

Starting your own Open Source Project

Prerequisite for a stating a new OSS project

- Develop a new idea
- Evaluate the concept or idea
- Study and analyse if something is already available ?
 - If yes, improve your idea
 - If no, move ahead
- Evaluate if you posses the required skills and expertise ?
- If no , work with the community to gain the same and later, re-evaluate .
- If yes , start your own project

Starting your own Open Source Project

Here are some ways in which the work on an open source project can start :

- An individual conceptualise the idea for a project and announces the same in public
- An individual starts the implementation on the project code base (limited feature but working) , release it to the public as the first version of the open source project
- The source code of a mature project is released to the project
- A well established open source project is forked by an interested individual or outside organisation with intent to extend it

Starting your own Open Source Project

Process model for development of open source software projects

Feasibility study phase – Investigate what exactly exists - if a similar or same project exist - refine or improve on your idea

Stage 1 : Initiation phase : in case a new project is started

- Project identification
- Searching for development Team
- Solution Identification and development of a work plan
- **Stage 2 : Execution Phase** : In case an existing project is adopted
 - Code Development and Testing
 - Code Review and commit
 - Documentation
- **Stage 3 : Release Phase** : Release code to the public

Starting your own Open Source Project

Develop the eco system for the project

- Establish a goal for your project
- Create a plan , roadmap or strategy to accomplish your goal
 - Choose an exiting , popular license
 - Build an official website for your project
- Find community members
 - Raise awareness
 - Post your project details to relevant forums
 - Set up communications channels for your project
 - Create separate mailing lists , project specific discussions boards etc
 - Request contributors to further promote your project - efforts multiply

Starting your own Open Source Project

Develop the eco –system for the project

- Choose and set up other technical requirements of the project , which includes software development tools for :
 - Version control,
 - Issue tracking,
 - Automating build process,
 - Designing ,
 - Coding an testing.

Arrange for necessary funds , if required.



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
08 April, 2023



Session 11 : Git and GitHub

History of Git

- The famous open source project – Linux kernel – used a proprietary Distributed Version Control System (DVCS) called Bitkeeper.
 - In 2005, the relationship between the company that developed Bitkeeper and the Linux community broke down
 - The Bitkeeper company NO longer offered the tool free of charge
 - The Linux community decide to develop their own DVCS for supporting large open source projects
-

What is Git ?

- An **Open-source distributed version control system (DVCS)**
- Primarily used by developers - for the task of writing code.
- It has a command line interface – **GIT CLI**
- It helps to keep track of modifications to your files that are stored in a project repository (**“Repo”**)
- Allows a developer to work on this Repo either independently or with a team of developers working collaboratively on the same project
- It provides a useful environment for Team Work; everyone can work independently on the files and manage their changes later
- Additionally, it also maintains a permanent record of who made which changes

What is GitHub ?

- GitHub is a **website** that allows developers to upload their Git repositories online – on the Internet
- GitHub comes with a rich **visual interface** for navigating the project repositories
- It not only helps in providing a backup of the files, but also make **collaboration over repositories** much more easier and effective.
- Additionally, it enables other people also to navigate through your repos and hence helps in **easier collaboration**
- It is **not mandatory to use Git**, while you are using GitHub, however , it is quite common to use GitHub while using Git

Let's build from here

The complete developer platform to build, scale, and deliver secure software.

100+ million

Developers

4+ million

Organizations

330+ million

Repositories

90%

Fortune 100

- GitHub is a code hosting platform for collaboration and version control.
 - GitHub lets you (and others) work together on projects.
-

GitHub CLI

- **GitHub CLI is an open source tool for using GitHub from your computer's command line.** When you're working from the command line, you can use the GitHub CLI to save time and avoid switching context.

GitHub CLI includes GitHub features such as:

- View, create, clone, and fork repositories
 - Create, close, edit, and view issues and pull requests
 - Review, diff, and merge pull requests
 - Run, view, and list workflows
 - Create, list, view, and delete releases
 - Create, edit, list, view, and delete gits
-

GitHub Desktop



- GitHub Desktop is an open source tool that enables you to be more productive.
- GitHub Desktop encourages you and your team to collaborate using best practices with Git and GitHub.

Just a few of the many things you can do with GitHub Desktop are:

- Add changes to your commit interactively
- Quickly add co-authors to your commit
- Checkout branches with pull requests and view CI statuses
- Compare changed images

GitHub Enterprise

GitHub provides two types of Enterprise products:

- **GitHub Enterprise Cloud**
- **GitHub Enterprise Server**
- The main difference between the products is that GitHub Enterprise Cloud is hosted by GitHub, while GitHub Enterprise Server is self-hosted.

GitHub Mobile

- GitHub Mobile gives you a way to do high-impact work on **GitHub Enterprise Cloud quickly and from anywhere.**
- GitHub Mobile is a safe and secure way **to access your GitHub Enterprise Cloud data** through a trusted, first-party client application.

With GitHub Mobile you can:

- Manage, triage, and clear notifications
- Read, review, and collaborate on issues and pull requests
- Edit files in pull requests
- Search for, browse, and interact with users, repositories, and organizations
- Receive a push notification when someone mentions your username
- Secure your GitHub.com account with two-factor authentication
- Verify your sign in attempts on unrecognized devices

In summary

- Git is responsible for everything GitHub-related that happens locally on your computer.
- You can also install **GitHub CLI** to use **GitHub** from the command line.
- If you want to work with **Git locally**, but **do not want to use the command line**, you can download and install the **GitHub Desktop client**.
- If you do not need to work with files locally, **GitHub Enterprise Cloud** lets you complete many Git-related actions directly in the browser

GitHub essentials

- Repository : Creating , forking , cloning
- Branches
- Commits
- Pull Requests

GitHub essentials : Repository

- A GitHub **repository** can be used to store a development **project**.
 - It can contain **folders** and any type of **files** (HTML, CSS, JavaScript, Documents, Data, Images).
 - A GitHub repository should also include a **license** file and a **README** file about the project.
 - A GitHub repository can also be used to store ideas, or any resources that you want to share.
 - A project repository helps in managing project work ; it also helps in executing collaborative engagements and discussions over the project.
 - Individual as well as shared ownership (with other people in an organisations) of the repository is also possible.
 - Repository's visibility used to restrict access to the repository – PUBLIC/ PRIVATE
-

Creating a Repository

- One can create a new repository on a personal account or an organisation's account (with sufficient permission)
- For this, navigate to the upper –right corner of the home page of your GitHub account page and use the drop down menu , and select **New repository**
- Type a name for your repository , and an optional description
- Choose a repository visibility – set it to public
- Choose a license
- Click Create Repository

Forking a Repo

- In order to contribute to a repo created by someone else , the first step will be to fork that repo in to account
- Typically , most of the repositories are configured so that others (non owners) are not allowed to directly “Push” changes to them.
- Instead, one need to request the administrator of the repository to “pull” the proposed changes from the online repository in to the main repository .
- In order to do this , it is necessary for you to have a version of the repo on GitHub from where the admin can pull the changes proposed by you.
- Hence , forking a repo is generally the first step that is executed in order to make a copy of the current state of the repo and associate that copy with your account.

How to fork a repo ?

- Login into your account on GitHub.
- Navigate to the repo you wish to contribute to.
- Use the fork option to fork a repo.
- Once completed , a writable copy of this repo will be created in your GitHub account.
- This process will NOT bring about any changes in your local computer.

Cloning a Repo

- One can clone a GitHub repository (hosted online) and create a local copy of the same on to the local system
 - Cloning a repository brings down a complete copy of the repository data that GitHub has at that point in time , including all version of every files and folder for the project
 - One can now make changes to this local repo, commit them and later push them back to the forked repo on GitHub
-

Commit changes to Repo

- This helps to save the changes that you would make to your local repository
- Commit **will not cause any changes** to be made to your remote repository on GitHub

GitHub essentials : Branch



- A GitHub branch is used to work with different **versions** of a repository at the same time.
- By default a repository has a **master** branch (a production branch).
- Any other branch is a **copy** of the master branch (as it was at a point in time).
- New Branches are for bug fixes and feature work separate from the master branch.
- When changes are ready, they can be merged into the master branch. If you make changes to the master branch while working on a new branch, these updates can be pulled in.

GitHub essentials : Commits



Commits : At GitHub, changes are called commits. Each commit (change) has a description explaining why a change was made.

Pull Requests : Pull Requests are the heart of GitHub **collaboration**.

- With a pull request you are **proposing** that your changes should be **merged** (pulled in) with the master.
- Pull requests show content **differences**, changes, additions, and subtractions in **colors** (green and red).
- As soon as you have a commit, you can open a pull request and start a discussion, even before the code is finished.

What is a Pull request ?

- Pull requests are used to inform others about the changes that you have made to your repository on GitHub
- After opening a Pull request , one can carry out discussions over the proposed changes with the owner of the open-source project and other collaboration as well.
- All can review the proposed and add follow –up commits before merging the changes in to the base branch

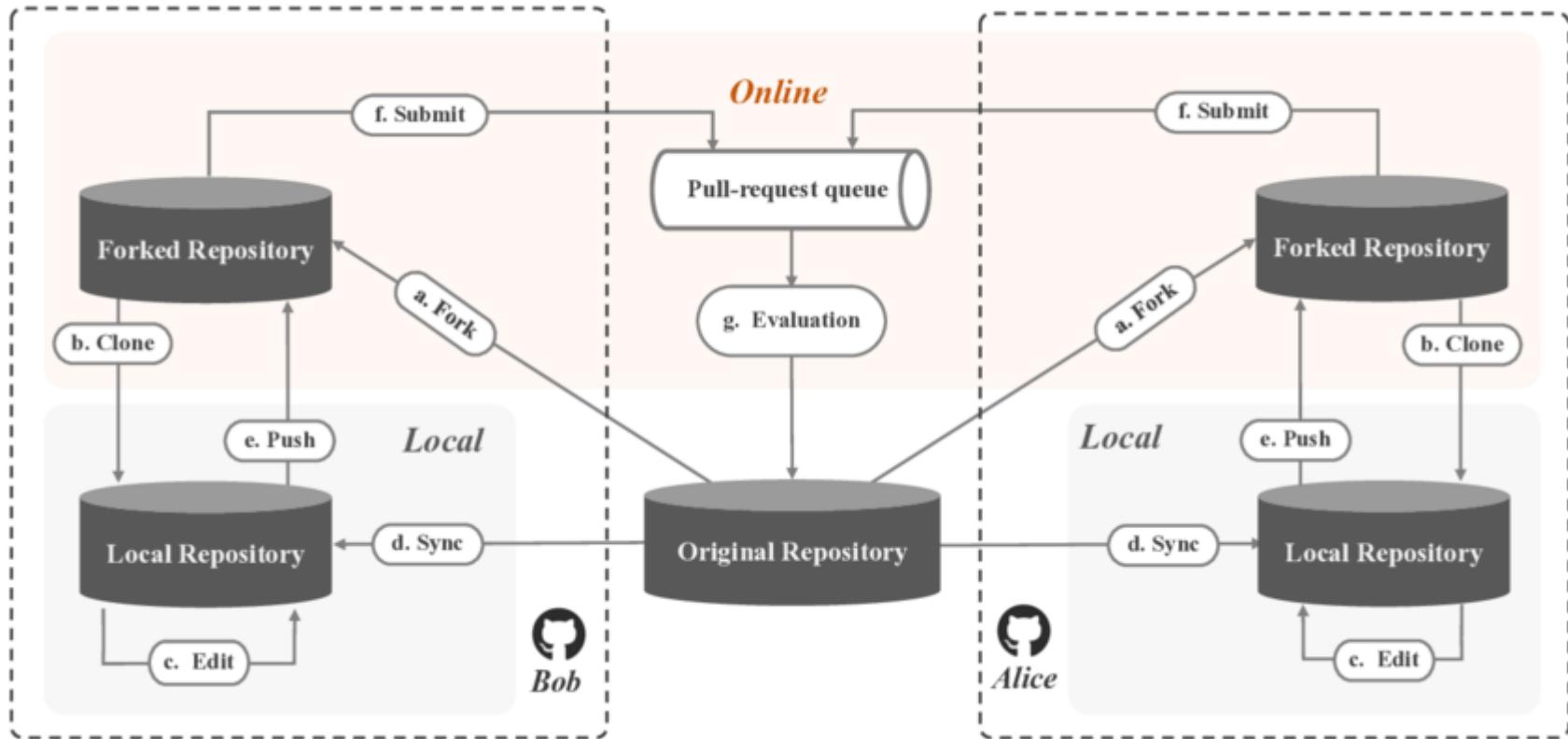
Merging or closing a Pull Request

- By default , any pull request can be merged at any time (if there does not exist any conflict).
- In case of conflict, it is the responsibility of the project owner to manually resolve the conflict and then merge the changes.

Push changes to GitHub

- One must push the changes that have been committed in the **Local Repo** on to the **Remote Repo** on GitHub

Workflow of Pull Development model





GitHub Products and Accounts

GitHub's Products

- GitHub offers **Free and Paid products** for storing and collaborating on code. Some products apply only to personal accounts, while other plans apply only to organization and enterprise accounts.
- **Git Hub Accounts** allow you to organize and control access to that code. There are three types of accounts on GitHub.
 - **Personal accounts** : Every person who uses GitHub signs into a personal account.
 - **Organization accounts** : An organization account enhances collaboration between multiple personal accounts.
 - **Enterprise accounts**: An enterprise account allows central management of multiple organizations.

Personal Account

- Each Personal account uses either **GitHub Free or GitHub Pro**.
- All personal accounts can own an unlimited number of public and private repositories, with an unlimited number of collaborators on those repositories.
- If you use GitHub Free, private repositories owned by your personal account have a limited feature set.
- You can upgrade to GitHub Pro to get a full feature set for private repositories.
- Most people will use one personal account for all their work on GitHub.com, including both open source projects and paid employment.
- If you're currently using more than one personal account that you created for yourself, recommended to combin the accounts.

Organization accounts

- All organizations can own an unlimited number of public and private repositories.
- Organizations are **shared accounts** where an unlimited number of people can collaborate across many projects at once.
- You can use organizations for free, with **GitHub Free**, which includes limited features on private repositories.
- To get the full feature set on private repositories and additional features at the organization level, including SAML single sign-on and improved support coverage, you can upgrade to **GitHub Team** or **GitHub Enterprise Cloud**.
- Like personal accounts, organizations can own Resources such as repositories, packages, and projects. However, you cannot sign into an organization.
- Instead, **each person signs into their own personal account, and any actions the person takes on organization resources are attributed to their personal account.**

Organisation account

- Each personal account can be a member of multiple organizations.
- The personal accounts within an organization can be given **different roles in the organization, which grant different levels of access** to the organization and its data.
- All members can collaborate with each other in repositories and projects, but **only organization owners and security managers can manage the settings for the organization and control access** to the organization's data with sophisticated security and administrative features.
- You can also create **nested sub-groups of organization members called teams, to reflect your group's structure and simplify access management.**

Enterprise accounts

- **GitHub Enterprise Cloud and GitHub Enterprise Server** include enterprise accounts, which allow administrators to centrally manage policy and billing for multiple organizations and enable inner sourcing between the organizations.
- Your enterprise account on GitHub.com allows you to manage multiple organizations.
- Your enterprise account must have a handle, like an organization or user account on GitHub.
 - Organizations are shared accounts where enterprise members can collaborate across many projects at once.
 - Organization owners can manage access to the organization's data and projects with sophisticated security and administrative features.

Enterprise accounts

Administrators for the enterprise account

- Billing and usage (services on GitHub.com, GitHub Advanced Security, user licenses)
- Security (single sign-on, IP allow lists, SSH certificate authorities, two-factor authentication)
- Enterprise policies for organizations owned by the enterprise account

Best practices for enterprises



- Use human-readable usernames
- Avoid extensive collaboration in user-owned repositories
- Minimize the number of organizations
- Use policies
- Identify the best authentication method for your enterprise
- Assign multiple owners



Open Source Software Engineering

SE ZG587

BITS Pilani
Pilani Campus

Kumar Manish
15 April, 2023



Session 13 : Git and GitHub

Git, GitHub and GitHub Enterprise Cloud

Let's build from here

The complete developer platform to build, scale, and deliver secure software.

100+ million
Developers

4+ million
Organizations

330+ million
Repositories

90%
Fortune 100

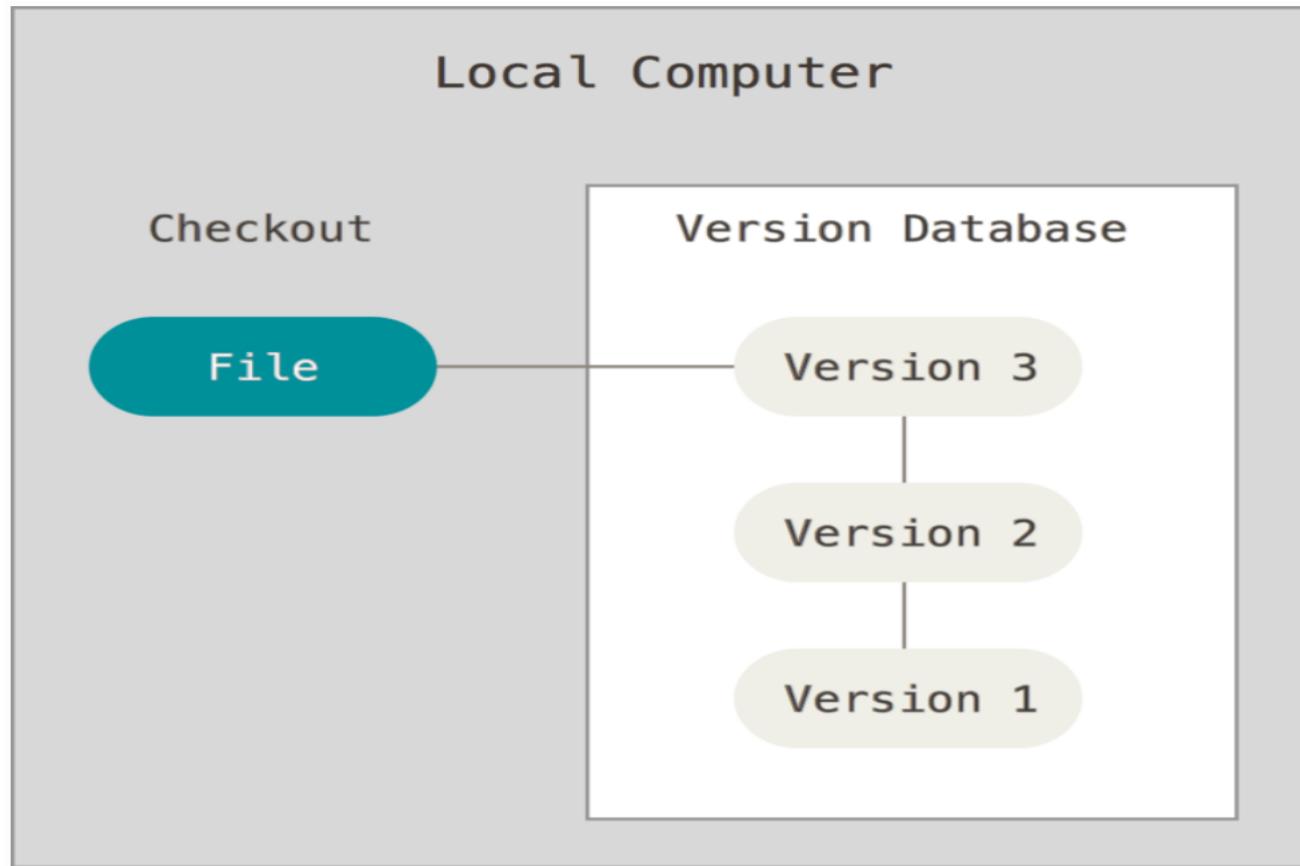
- GitHub is a code hosting platform for collaboration and version control.
- GitHub lets you (and others) work together on projects.

Development goals of Git

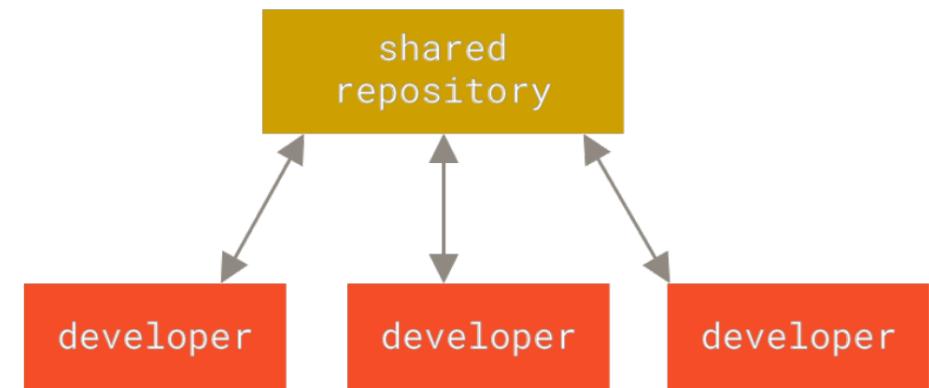
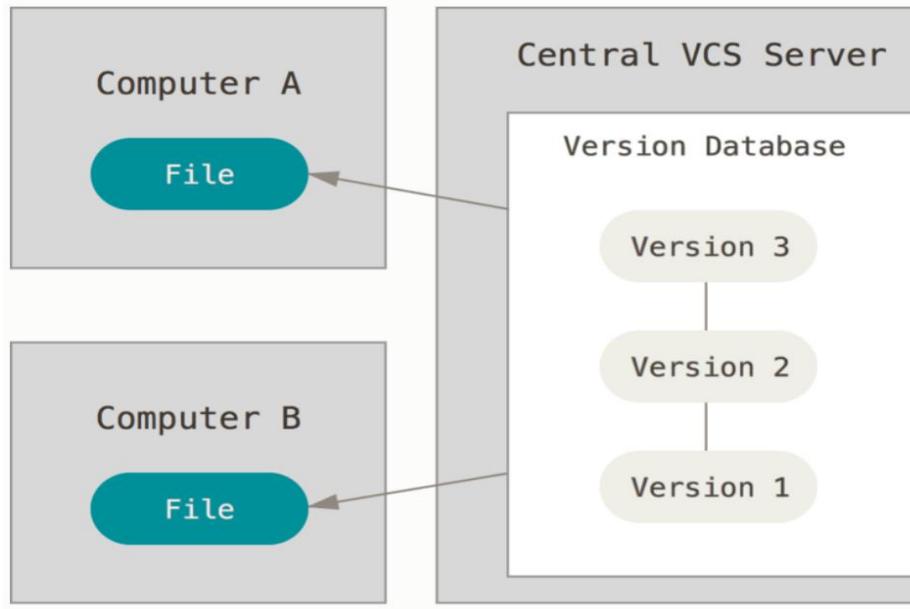
- Speed,
- Simplified design
- Completely distributed
- Strong support for branching (non linear development), involving thousands of parallel branches
- Able to handle large projects like the Linux kernel efficiently; in terms of speed and data size.

Local version control system

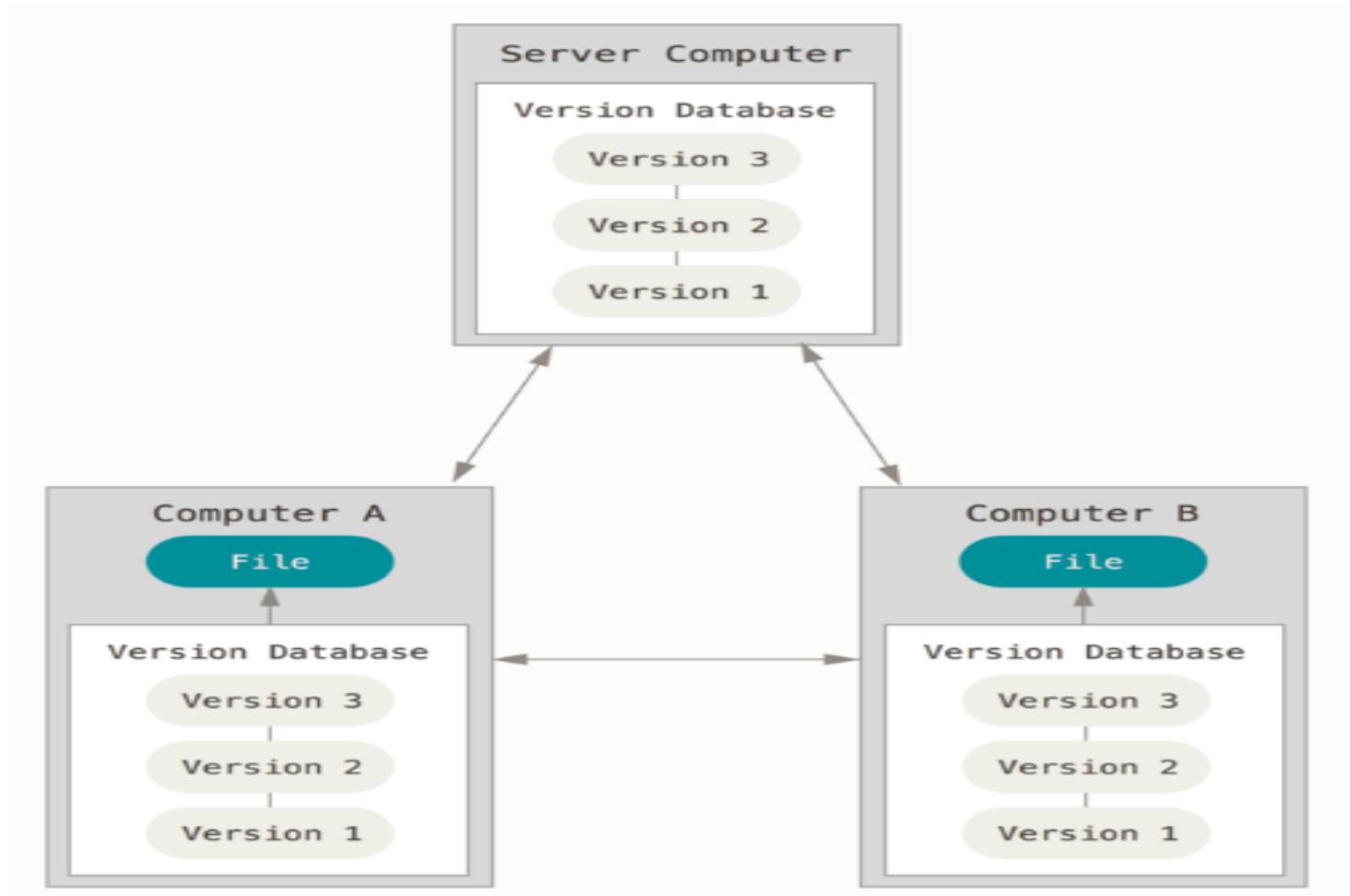
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.



Centralised version control system

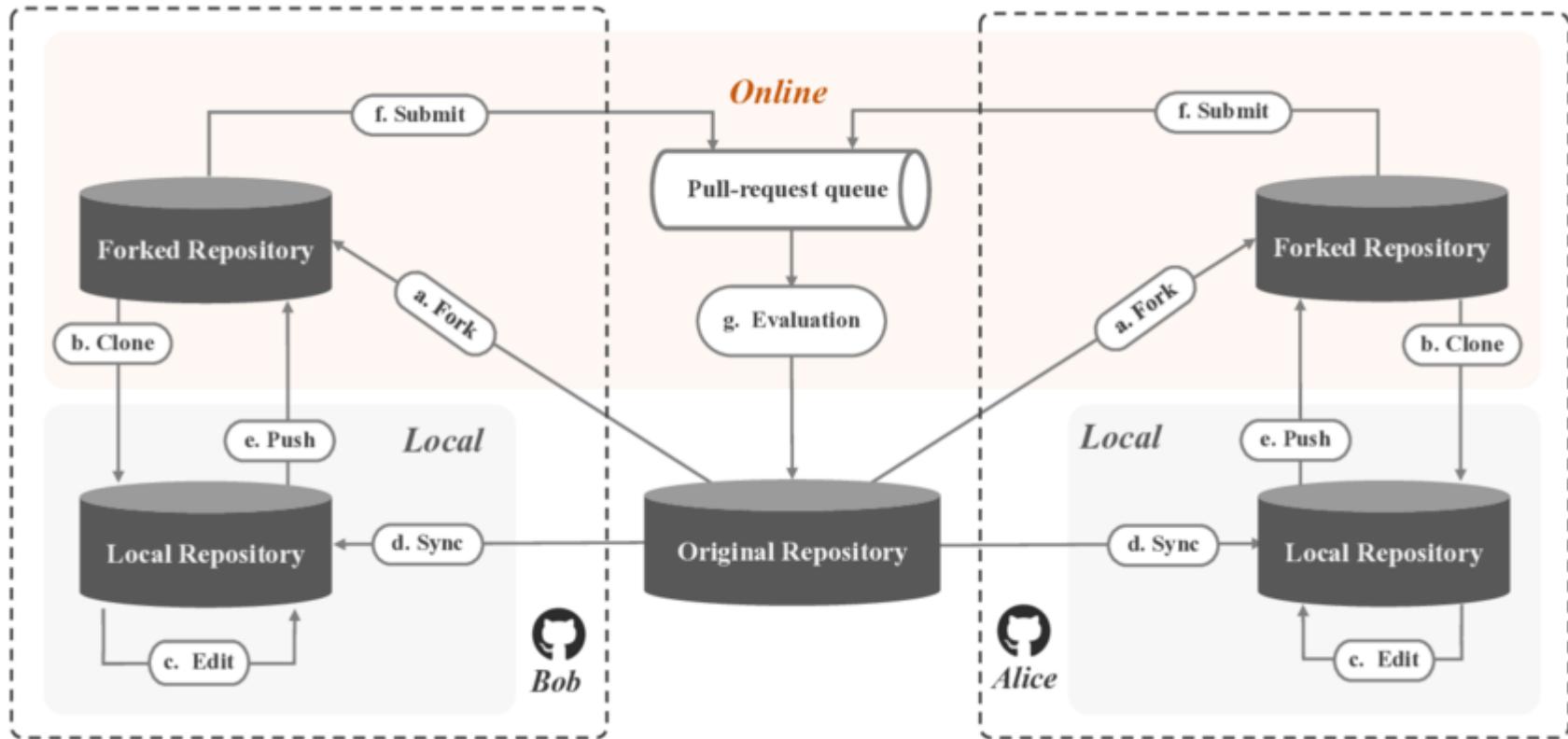


Distributed version control system



Working with git & GitHub

Git Workflow - Pull based model



• **Git Hub Accounts :**

- **Personal accounts :** Every person who uses GitHub signs into a personal account.
- **Organization accounts :** An organization account enhances collaboration between multiple personal accounts.
- **Enterprise accounts:** An enterprise account allows central management of multiple organizations

Personal Account

- Each Personal account uses either **GitHub Free** or **GitHub Pro**
- Own an **unlimited number of public and private repositories**, with an **unlimited number of collaborators on those repositories**.
- **GitHub Free** : private repositories owned by your personal account have a limited feature set.
- **GitHub Pro** : To get a full feature set for private repositories.

Organization Accounts

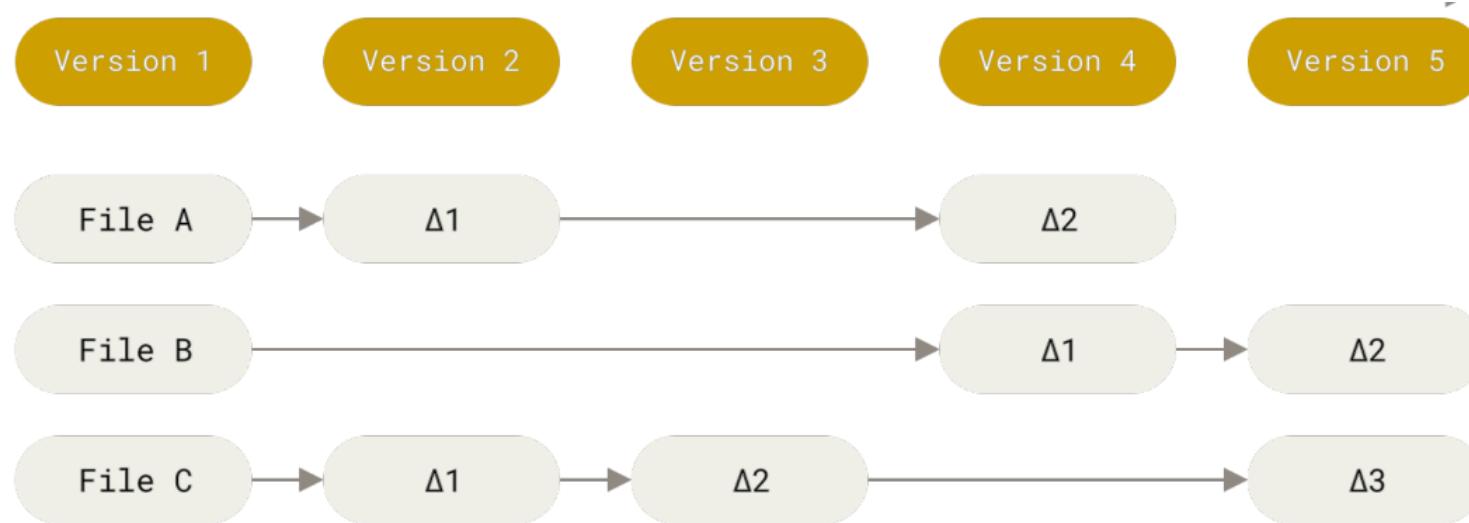
- Organizations are **shared accounts** where an **unlimited number of people** can collaborate across many projects at once.
- **GitHub Free : For Free use in Organization** but limited features on private repositories.
- **GitHub Team or GitHub Enterprise Cloud** : To get the full feature set on private repositories, including SAML single sign-on and improved support coverage
- Nested sub-groups of organization members called **Teams**, to reflect your group's structure and simplify access management.

Enterprise accounts

-
- **GitHub Enterprise Cloud and GitHub Enterprise Server**
 - Include enterprise accounts, which allow administrators to centrally manage policy and billing for multiple organizations and enable inner sourcing between the organizations.
 - Enterprise account on GitHub.com allows you to manage multiple organizations.
 - Organizations are shared accounts where enterprise members can collaborate across many projects at once.
 - Organization owners can manage access to the organization's data and projects with sophisticated security and administrative features.

Characteristics of other version control system

- Older VCS like Subversions, Perforce, etc store information as a list of file based changes knowns as delta –based version control – store only the changes made to each file over time



Unique Characteristics of Git

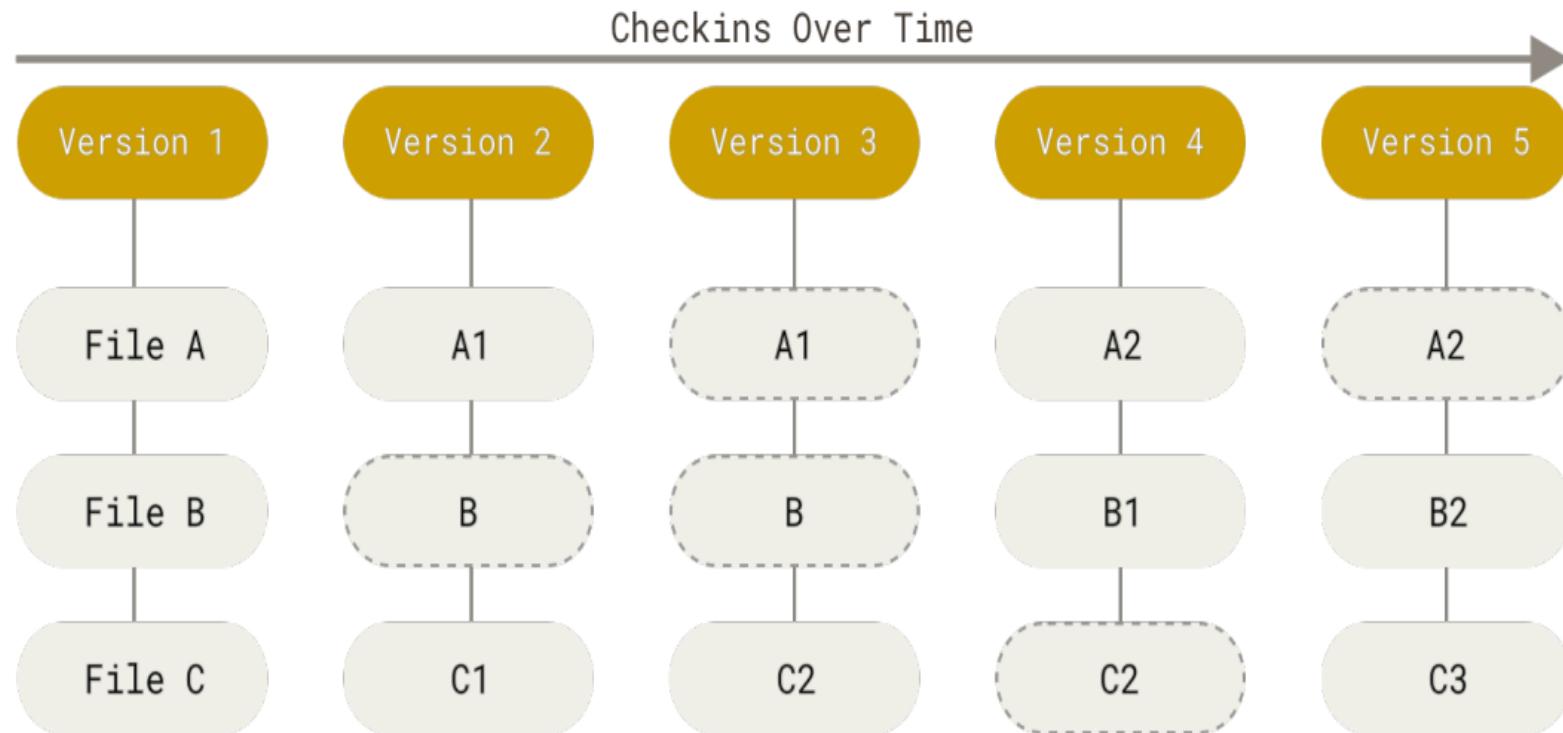


Stores Snapshots , Not differences

- Git does not use the delta approach; rather it uses an approach in which it stores data like a series of snapshots of a miniature file system - **Stream of snapshots**
- With every commit, Git takes a snapshot of all the files at that moment and stores a reference to the snapshot.
- If some files have not changed, Git does not store the file again, but only a link to the previous version of the same file, which is already stored with it.
- This makes git more powerful, efficient and fast as compared to its counterparts.
- Also, helps in implementing branching efficiently

Unique Characteristics of Git

Storing data as snapshots of the projects over time



Unique characteristics of git

Nearly Every operation is Local :

- Nearly every **operation in git requires only local files** and resources to execute ; usually no information is required from other computers on the network.
- This is because the **entire history of the project is stored in the local computer of every individual** , making various operation faster.
- In order to browse the entire history of the project , Git **does not need to get the same from the server** – it simply takes it directly from the users' local database and the same immediately available.

Unique characteristic of Git

The **three states in Git** : In Git, a file resides in one of the three states : **modified** , **staged** and **committed**.

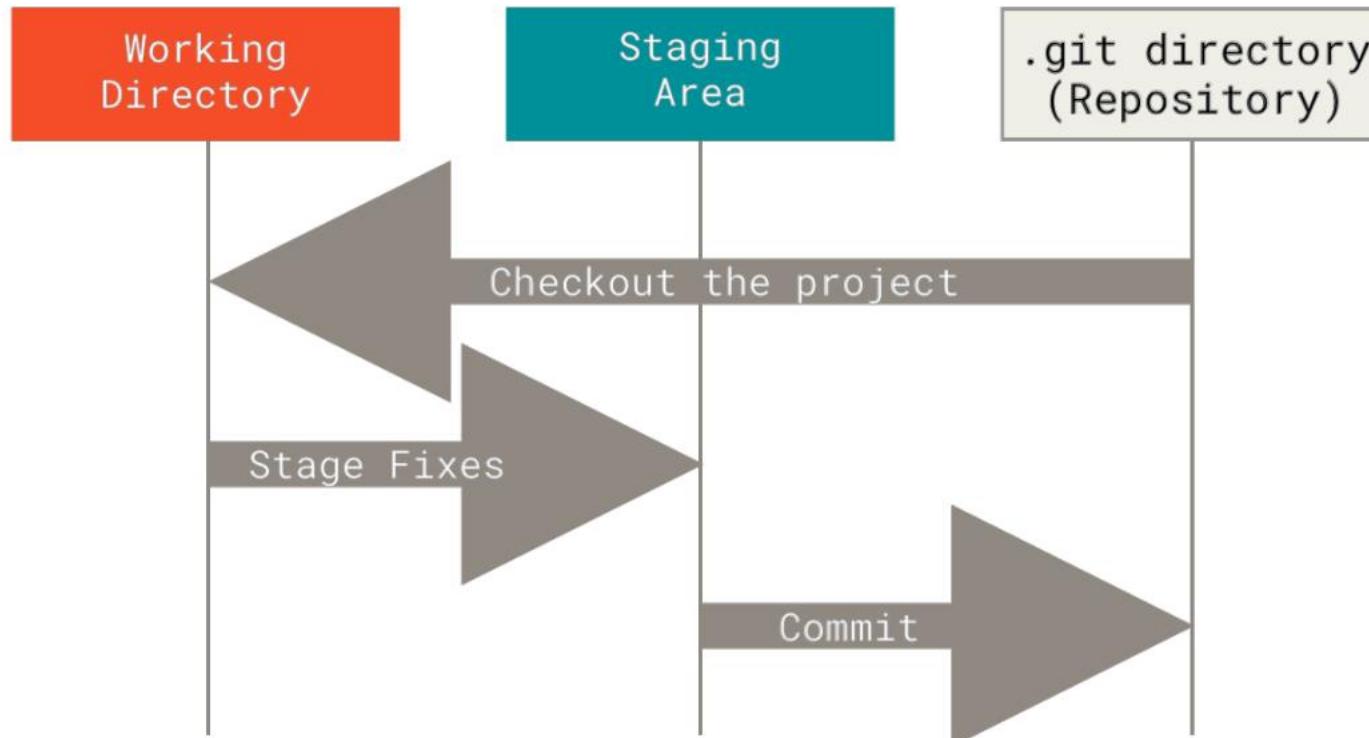
- **Modified state** – The file has changed but the changes have not been committed to the database yet
- **Staged state** - the file has been marked as modified file in its current version and is marked to go in to the next commit snapshot.
- **Committed state** – a file resides in this state is safely stored in the local database

Hence, Git project has three main section :

- The working tree
- The staging area
- And the Git directory

Unique characteristic of Git

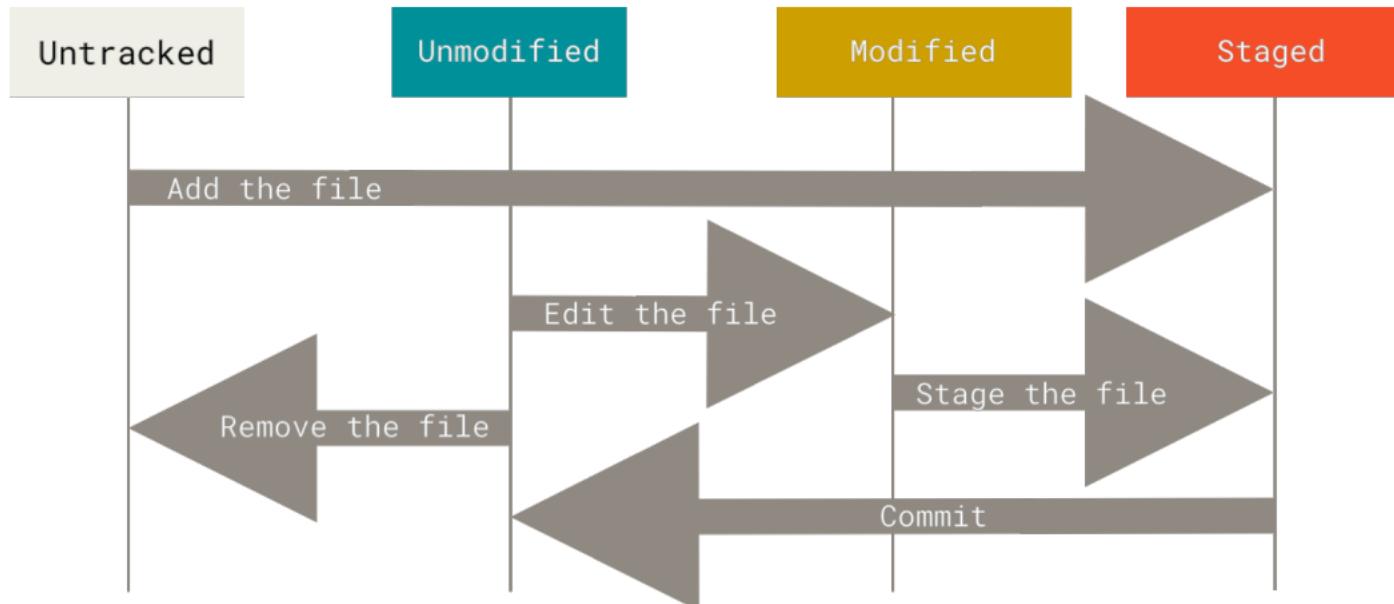
The Three stages in Git



The three states in Git

- The **working directory** is the single checkout of the latest version of the projects. The project files are pulled out of the compressed database in the Git directory and placed on disk for use.
 - The **Staging area** is a file , usually contained in the Git directory, and stores the information about the files that are staged and are hence marked what go into the next commit.
 - The **Git directory** is used by Git to store the metadata and object database for the project. This is the most important part of Git, and is copied to the local system when a repository is cloned from a remote repository
-

Life cycle of the status of Files



Basic Git workflow

The basic git workflow goes like this :

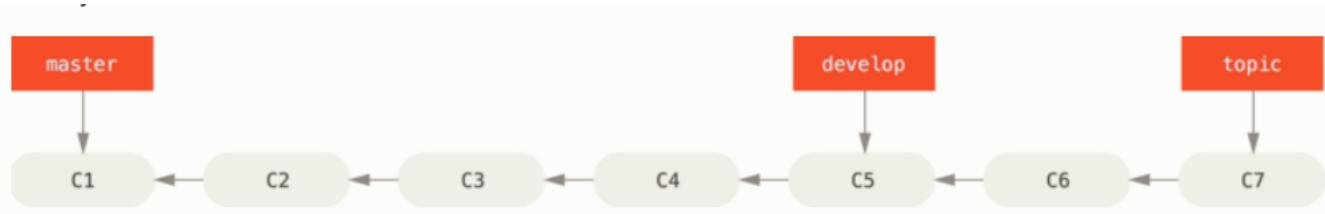
- The files are modified in the working directory
- The files are selectively staged and only those changes that are required to be part of the next commit are moved to the staging area
- Commit is executed, which takes the files as they are in the staging area and stores that snapshot permanently to the Git directory

Git Branching

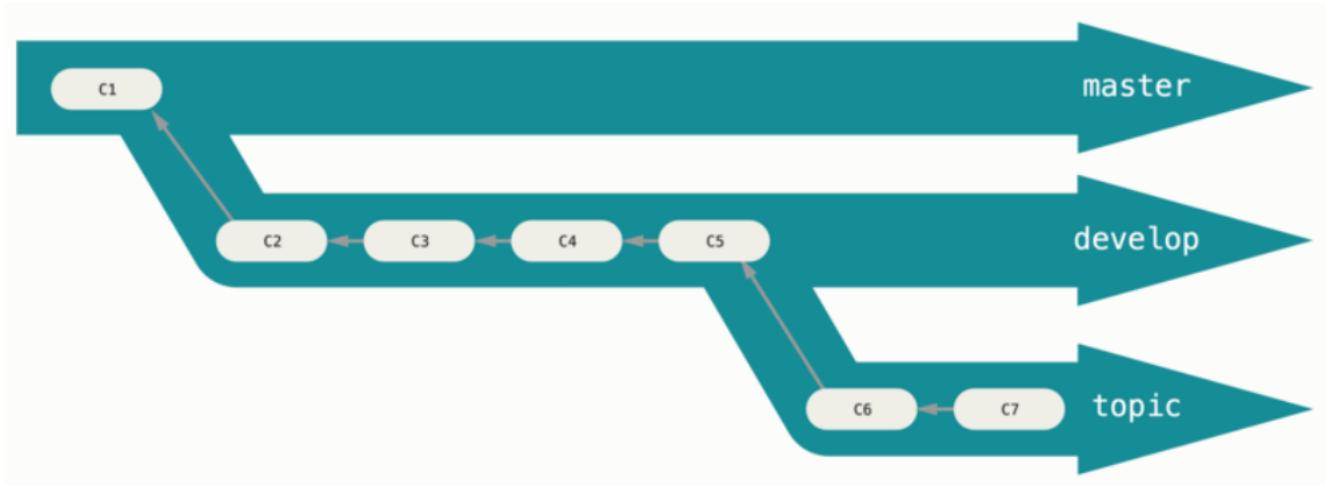
- Need of Branching :
- Helps to work in isolation
- It can orchestrate parallel development allowing developers to work on tasks simultaneously as part of team
- And parallel build and testing ensure developers get the feedbacks they need quickly
- Parallel tasks possible by creating multiple branches :
While you wait for changes from one branch to be pulled by owner , you may start your work on another branch

Branching strategy

Linear View of branching



A “silos” view of progressive stability branching



Branching in Git

- Since Git stores data as a series of snapshots and not as a series of changes sets - it is able to provide support for branching more efficiently as compared to other VCS
- When commit is done,
 - A pointer to the snapshot of the staged content is stored
 - This pointer object also contains author's name and email address , commit message and pointer to the commit (s) immediately before the current commit
 - Zero parents for the initial commit
 - One parent for a normal commit
 - Multiple parents for a commit resulting from a merge of tow or more branches

References : for further readings

- Types of version control systems

https://subscription.packtpub.com/book/application_development/9781849517522/1/ch01lvl1sec12/types-of-version-control-systems

- Git Book

<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

- GitHub Pages

<https://pages.github.com/>

Assignments :

Understanding Open-source project , submit on anyone from following lists :

- **Moodle :**
 - <https://moodle.org>
 - Introduction and history
 - Managing a Moodle site
 - Managing a Moodle course
 - Managing content
- **Linux Project :**
 - <https://www.linux.org/>
 - <https://www.linux.com/>
 - <https://www.linuxfoundation.org/projects/hosting>
 - Introduction and History of movement
 - Linux Shell, and file structures
 - Security : Encryption, Integrity checks and signatures
 - Linus Desktop : GNOME , KDE
- **Kubernetes Project :**
 - <https://kubernetes.io/>
 - Introduction and history of movement
 - Kubernetes Components
 - Kubernetes API, Kubernetes Objects, Pods
- **Eclipse Project**
 - <https://www.eclipse.org/>
 - Introduction and History of movement
 - Installation and explore windows, explore menus views,
 - Perspective and workspaces

Next sessions :

-
- GitHub commands
 - Collaboration and communication tools on GitHub
 - Working with GitHub pages
 - Understanding Markdown



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
22 April, 2023

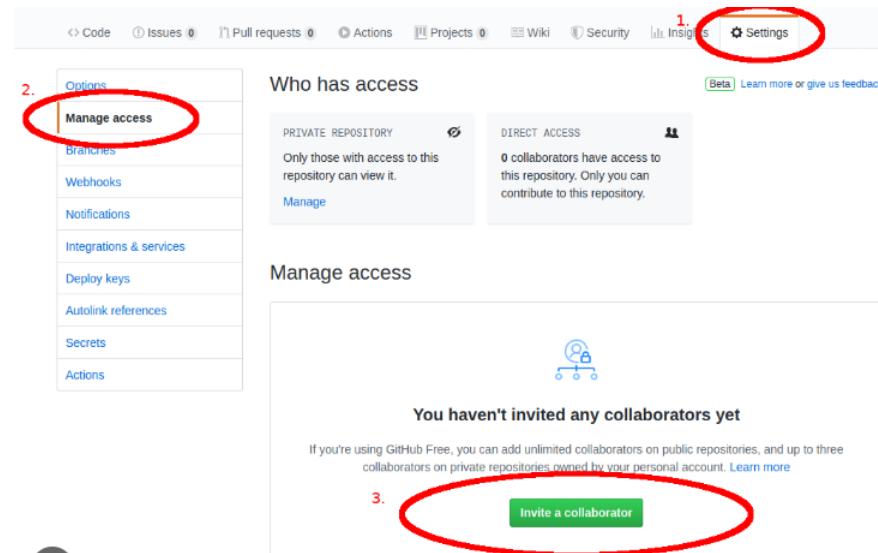




Session 13 : OSSE – Collaboration and communication tools on GitHub

Adding Collaborators to a Github repo

- Navigate to the repo to which you want to add collaborators
- Click the setting button → and then Manage Button on left side panel
- Invite collaborators using the “invite” button



Email Notifications

An email notification mechanism is put in place for communication

The owner of the repo gets a notification through email when a pull request is placed

This email notification consists of the following :

- Provides a list and diff of the files that been proposed changed in the Pull request
- It also provides a link to navigate to the Pull Request on GitHub

Collaborating on Pull request

- GitHub provides collaboration mechanism on Pull request through the conversation page or the discussion page, with the contributors who created the pull request
- The Project owner can comment on the entire Pull request or the entire commit or choose specific lines of code and comment ion the same
- Other collaborators / contributors may also provide comments on the Pull Request
- In this case, the owner of the project will continue to get email notifications about the discussion on the pull request.

Managing Issues

- Issues help to keep track of tasks, enhancements and bugs for a Project,
- An issue in GitHub consists of the following :
 - A title and description which helps to describe additional details about an issue
 - Colour coded labels help to categories and filter issue
 - A millstone acts like a container for issues. This helps in association issues with specific features or project phases
 - Issues can also be mapped to assignees – who would then be responsible for working on the issue at any given time.

Milestone , Label and Assignees

- Milestone, Labels and assignees help in filtering and categorizing issues
- Use the “**gears**“ button on the sidebar on the right in order to add or change labels , assignees and milestones
- Labels help to categorize issues in to different categorise
- An issue can have multiple labels; can filter issues on these labels - one by one or many labels at a time



Labels Milestones New milestone

2 Open 1 Closed

Sort ▾

beta release

No due date Last updated 3 days ago

50% complete 1 open 1 closed

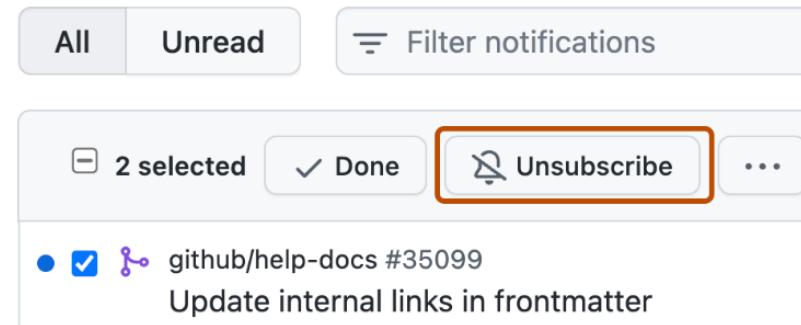
Edit Close Delete

Mentions and Notifications

- GitHub also provides another notifications system called Mentions - this can be used when you would want to pull in People into discussions and need feedback from them
- For this, in a comment , you need to start typing with the “@” character and it will begin to autocomplete with the names and usernames of people who are collaborators or contributors in that project - this is called mentions
- On posting a comment with use mention , that user will notified
- This notification mechanism proves to be an effective way of involving people into discussion.

Mentions and Notifications

- Whenever a user gets mentioned on a issue or pull request , he / she gets be “subscribed” to it and will start to get all related notifications.
- Moreover, by default , a contributors is be subscribed to something that he / she opened , or is watching a repository or comments on something
- In order to stop receiving notification, use the “unsubscribe “ button

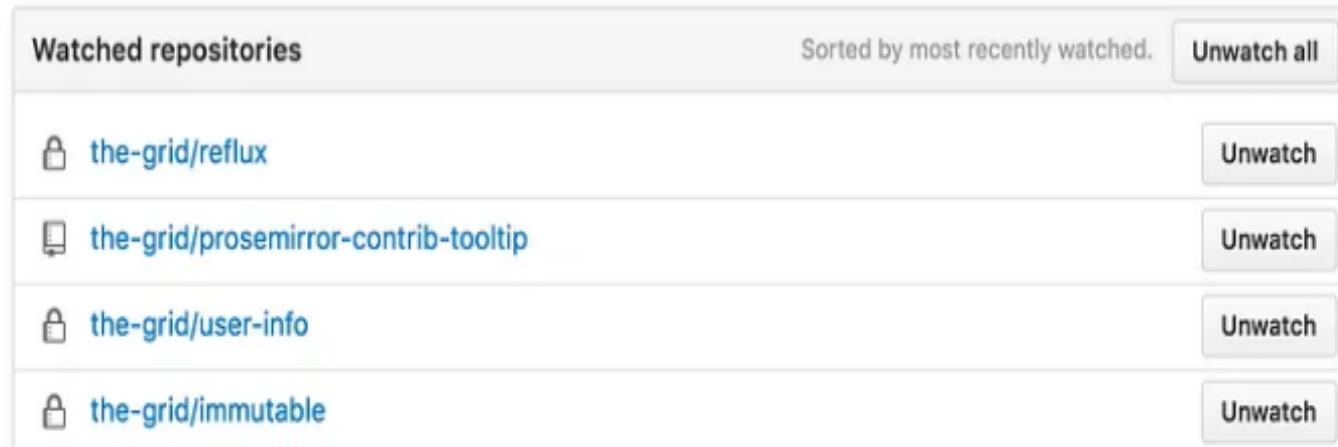


Watching a repo

- You can start watching a repo by simply navigating to that repo from your login and **click on the watch icon** on the extreme right corner, close to fork button
- If you are watching a repository , you will receive notification for all conversations – project issue, pull requests comments on commit and any other comments.
- In case you have **push access to a repository** -GitHub automatically adds it to your watch list – this is an **auto watch features**

Un-watching a repo

- To unwatch a repo, navigate to notification settings → List of all watch repo and selectively unwatch them
- By default all are in watch list



A screenshot of a web interface showing a list of watched repositories. The header reads 'Watched repositories' and 'Sorted by most recently watched.' A 'Unwatch all' button is located in the top right corner. Below the header, there are four repository entries, each with a lock icon, the repository name, and a 'Unwatch' button to its right. The repository names are: 'the-grid/reflux', 'the-grid/prosemirror-contrib-tooltip', 'the-grid/user-info', and 'the-grid/immutable'.

Watched repositories		Sorted by most recently watched.	Unwatch all
	the-grid/reflux		Unwatch
	the-grid/prosemirror-contrib-tooltip		Unwatch
	the-grid/user-info		Unwatch
	the-grid/immutable		Unwatch

Star / un-star a repo

- You can star a repo by simply navigating to that repo from your login and click on the star icon on the extreme right corner , next to fork button
- When you start a project you can keep track of it, but you won't be notified of every change
- In order to star repo , use the star button next to the watch button on the home page of every repository

Project Administration : Changing the default branch



If you are willing to use a branch other than the “master” as your default branch, and want that people should open Pull request on it or see it as default,

Then, navigate to the repository’s **setting page** under the Branches “ tab and change the branch

The screenshot shows the GitHub repository settings page. At the top, there is a navigation bar with links for Requests (2), Actions, Projects (0), Wiki, Security (0), Insights, and Settings. The Settings link is highlighted. Below the navigation bar, the title 'Default branch' is displayed. A sub-header explains that the default branch is the "base" branch against which all pull requests and commits are automatically made. There are two buttons: 'main' with a dropdown arrow and 'Update'. A modal window titled 'Switch default branch' is open. It contains a 'Filter branches' input field, a list of branches with 'main' checked and 'master' uncheckable, and a note about pushing and protection rules. A 'Learn more' link is also present.

Changing visibility of a repo

- Navigate to “Settings” --> Option → Danger zone → change visibility

Making a repo Private :

- GitHub will detach public forks of the public repository
- Changing a repository’s visibility from internal to private will remove forks that belongs to any user without access to the newly private repository
- Any published GitHub Pages site will be automatically unpublished

Making a repo public :

- GitHub will detach all private forks and turn them in to a standalone private repository.

Transfer a project

- In order to transfer a project to another user or an organisation in GitHub, use the Transfer ownership “ option at the bottom of the “Options“ tab of your repository settings.
 - This is helpful in case a project is to be abandoned ad some one else wants to take over ,
 - Or if the project is getting and you want to move it into an organisation,
 - This will move the repository along with all its watches and star to another place
 - It will also sets up a redirect from your URL to the new place
-

Transfer a project

Danger Zone

Make this repository private
Hide this repository from the public. [Make private](#)

Transfer ownership
Transfer this repository to another user or to an organization where you have admin rights. [Transfer](#)

Delete this repository
Once you delete a repository, there is no going back. Please be certain. [Delete this repository](#)



Archive a repo

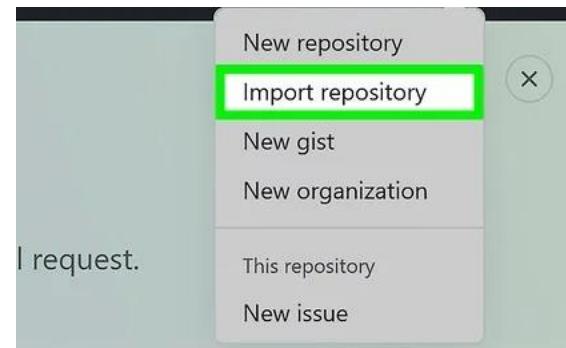
- You can archive a repository to make it read – only for all users and indicate that it's no longer actively maintained.
- You can also unarchive repositories that have been archived.
- It is recommended that you close all issues and pull requests, as well as update the README file and description, before you archive a repository.
- Navigate to settings --? Option → Danger zone → Archive this repo

Delete a repo

- Only members with owner privileges for an organisation or admin privileges for a repository can delete an organisation repository
- Deleting a public repository will not delete any forks of the repository.
- Navigate to settings → Option → Danger zone :

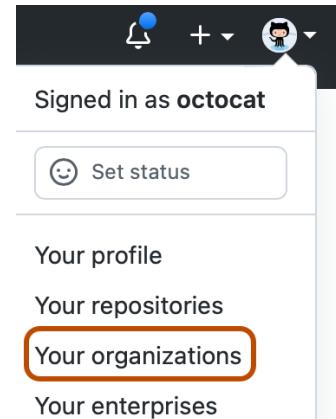
Import a repo in GitHub

- If you have a public project hosted on another version control system, you can automatically import it to GitHub using GitHub Import tool.
- You will be prompted to enter the URL of the old repo
- This is followed by the steps which are same when you create a new repo in GitHub



Managing an Organization

- GitHub also provides the option to create organisation , with shared ownership of project repositories
- For creating a new Organisation, click on the “+” icon on the top right corner of your GitHub page and select “New organisation”
- Give an appropriate name to your Organisation and an email id.
- Now invite co-owners to the account



Creating Teams

You may also divide people belonging to your organisation in to teams.

For e.g. – you may have teams like - front end developers team, back end team etc

Create repository with in a team and assign members to it

Understanding Markdown language



Markdown is a way to style text on the web

In GitHub , markdown is primarily used in:

- Writing comments in Pull request and issues
- Readme file or contributors files
- Files with .md or .markdown extension
- It helps to :
 - Modify the display of the document
 - Formatting text – as bold or italic
 - Adding images, hyperlinks
 - creating bulleted lists , numbered lists etc

Understanding Markdown : Text formatting

- It is just regular text with a few non –alphabetic characters , like # or “
- Use “word” or word to make a word bold
- Use “word” or word to make a word italics
- Link to website :
- [link to google](<http://google.com>)
- Header
 - Use # to indicate the heading level
 - For example :
 - # This is heading1
 - ## This is heading2
 - ### This is heading3
 - ##### This is heading 4

Understanding Markdown : Listing

- Use : for creating a bulleted list
- For example :
This can be used to create bullet points :
 - Point1
 - Point2
- Use simple numbering for creating an order list.
- For example : This can be used to create ordered list :
 - 1. Point 1
 - 2. Point 2
 - 3. Point 3

Markdown editors

[StackEdit – In-browser Markdown editor](#)

<https://dillinger.io/>

<https://pandao.github.io/editor.md/en.html>

What are GitHub Pages

GitHub Pages is a static site hosting service

They are commonly used to host a website about your project, or your organisation or even about yourself – directly from a repository on GitHub.com

It makes use of

- HTML
 - CSS and
 - JavaScript
-

Working with GitHub pages

- Navigate to GitHub.com
 - Create an account – it is linked to your email id
 - Example - username is kmbits
 - Create a repository in your login with the same name as your username - kmbits.github.io
 - You may NOT choose any license for the same
 - Create a file index.html
 - Use HTML to write content – create profile page
 - View your profile : <https://kmbits.github.io/>
-



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
29 April, 2023





Session 14 : working with GitHub

Getting started with Git

- Install Git for windows
- Navigate to the following link to install git
 - <https://github.com/git-guides/install-git>
- Installing on Linux
- Installing on macOS

Configure the Git

- The first step is to configure the username and email address for the current Git installations
- This will be used by git during every commit
- Launch Git bash and type the following commands :
 - `$ git config –global user.name “kmbits”`
 - `$ git config – global user.email mail@gmail.com`

You may also set your default text editor that you want to be used with Git

- `$ git config –global core.editor notepad`

Git configuration : check your settings

- Use the command given below to check the configuration settings you just made :

```
$ git config - - list
```

- Alternatively, you can also view the selectively the required parameters :

```
$ git config user.name
```

Getting Help

Use any of the following commands to get help on any Git Command

`$ git help <verb>`

`$ git <verb> --help`

`$ man git-<verb>`

For example , to get help on config command :

`$ git help config`

Cloning an existing repository

- In order to clone an existing Git repository , use the git clone command
- This will make a full copy of nearly all the data on the server. It also brings along the complete history of the project (with every version of every file).

- Syntax **git clone <url>**

```
$ git clone https://github.com/kmbits/demofolder.git
```

- This will create a directory named **demofolder** in the current working directory

Cloning existing directory

- If you wish to clone the repository with a directory name of your choice , provide the new name as an additional argument , as shown in the command below :

`$ git clone https://github.com/kmbits.demofolder.git MyClass`

- This helps in getting a working copy or checkout copy of the entire Git repository in your local system.

Checking file status

The git status command is used to check the status of the files in the current directory

`$ git status`

On branch master :

Your branch is up-to-date with 'origin/master',

Nothing to commit, working directory clean

- This implies that :
 - You have a clean working directory
 - No untracked files exist
 - No tracked file are in modified stage
 - No tracked file are staged
 - The current branch is origin/master
-

Tracked and untracked files

- Every file in your working directory can be in **tracked** or **untracked** state.
 - Tracked files are those which Git knows about;
 - They were a part of the last snapshot of Git;
 - They can be in any stage –unmodified , modified or staged
 - On cloning a repository , all of its files will be tracked and in unmodified state because Git has just checked them out and no modifications have been made on them so far.
 - Everything else in the directory are untracked files – these files were not in the last snapshot and are not in the staging area as well.
-

Adding untracked files

- Let us now add new file to the project
- For the navigate to the folder using explorer and add a `readUs.md` file
- [you may simply copy-paste the existing `Redame.md` file and rename it]
- Now execute the `git status` command and you will see something like this :

`$git status`

On branch master

Your branch is up-to-date with 'original/master'.

Untracked files :

(use "git add <file> " to include in what will be committed)

`ReadUs`

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

Adding untracked files

- Untracked primarily means that Git considers these files as unknown since they were not a part of the previous snapshot or commit.
- Git will not include these files in the commit snapshots until explicitly told to do so
- This features is added so that generated binary files or other irrelevant files are not automatically tracked

Adding untracked files

Execute the git status command again to check the status of the working directory

`$ git status`

On branch master

Your branch is up-to-date with “origin/ master’ ,

Changes to be committed:

New file : ReadUs

- The ReadUs file is now converted into a tracked file and also staged to be committed
- Additionally , if changes are committed at this point of the time, the version of the file at the time of running git add will be used.

Adding untracked file

- In order to begin tracking the new file, use the command **git add**
- The **git add** command takes the complete path name for either a file or a directory .
- If a directory name is mentioned , the command works to add all the files in that directory recursively.
- To begin tracking the ReadUs file, run the following command :

\$ git add ReadUs

Staging modified files

- Let us change the existing Readme.md file - which is already tracked
- The output after running **git status** command will be as follows :

\$git status

On Branch master

Your branch is up-to-date with 'origin/master/,'

Changes to be committed ;

New file : ReadUs

Changes not stages for commit :

Modified : ReadMe.md

Staging modified files

- The Readme.md file appears marked as “Changes not staged for commit” – this implies that
 - this file which is being tracked
 - Is a modified file and currently located in the working directory
 - but not yet staged
- In order to stage this modified file, run **git add** command
- **git add** is a multipurpose command . It is used to
 - begin tracking new files
 - To stage files,
 - Adding content for commit.

Staging modified files

Let's now move ReadMe.md file to the staging area by using **git add** command and run **git status** again :

```
$ git add ReadMe.md
```

```
$ git status
```

On branch master

Your branch is up-to date- with 'origin/master'.

Changes to be committed :

New file : ReadUs

Modified : Readme.md

Commit changes

Use the **git commit** command to commit the changes

```
$git commit -m "Myfirstcommit"
```

Visualizing Branching in Git

Let's assume you have a directory containing three files, that you stage and then commit, using the commands given below :

```
$ git add readus.txt license.txt  
$ git commit -m 'initial commit'
```

The git repository will now contain the following four objects :

- Two blobs – each one representing the contents of one of the three files
- One tree object that lists the contents of the directory and specifies which file names are stored as which blobs
- One commit with the pointer to the root tree
- And all commit meta data

Push changes to remote

Use the `git push` command to push the changes from your `master/main` branch to the `origin` server

`$ git push origin master`

- This command will work only if you have write access on the repository to which you are trying to push the changes to and if nobody has pushed in the meantime.
- If someone else also cloned the repository at the same time and he/she pushed the changes upstream and you try to push your changes upstream, your push will be rejected
- You will be required to fetch the changes made by them first, and incorporate it into yours and then you will be able to push your changes.



Moodle

Moodle : Freedom to learn.

- Moodle is open source under the [GPL license](#). Everything we produce is available for you to download and use for free.
 - Moodle is a free, online Learning Management system enabling educators to create their own private website filled with dynamic courses that extend learning, any time, anywhere.
 - <https://docs.moodle.org/401/en/Features>
 - [Moodle 1.0](#) was released in August 2002.
 - Founder : Martin Dougiamas
-

Linux Project



<https://www.linux.org/>

<https://www.linuxfoundation.org/projects/hosting>

<https://www.linux.com/>



BITS Pilani
Pilani Campus

Open Source Software Engineering

SE ZG587

Kumar Manish
6 MAY, 2023



Session 15 : Recap

Lifecycle and methodologies in Open Source Software

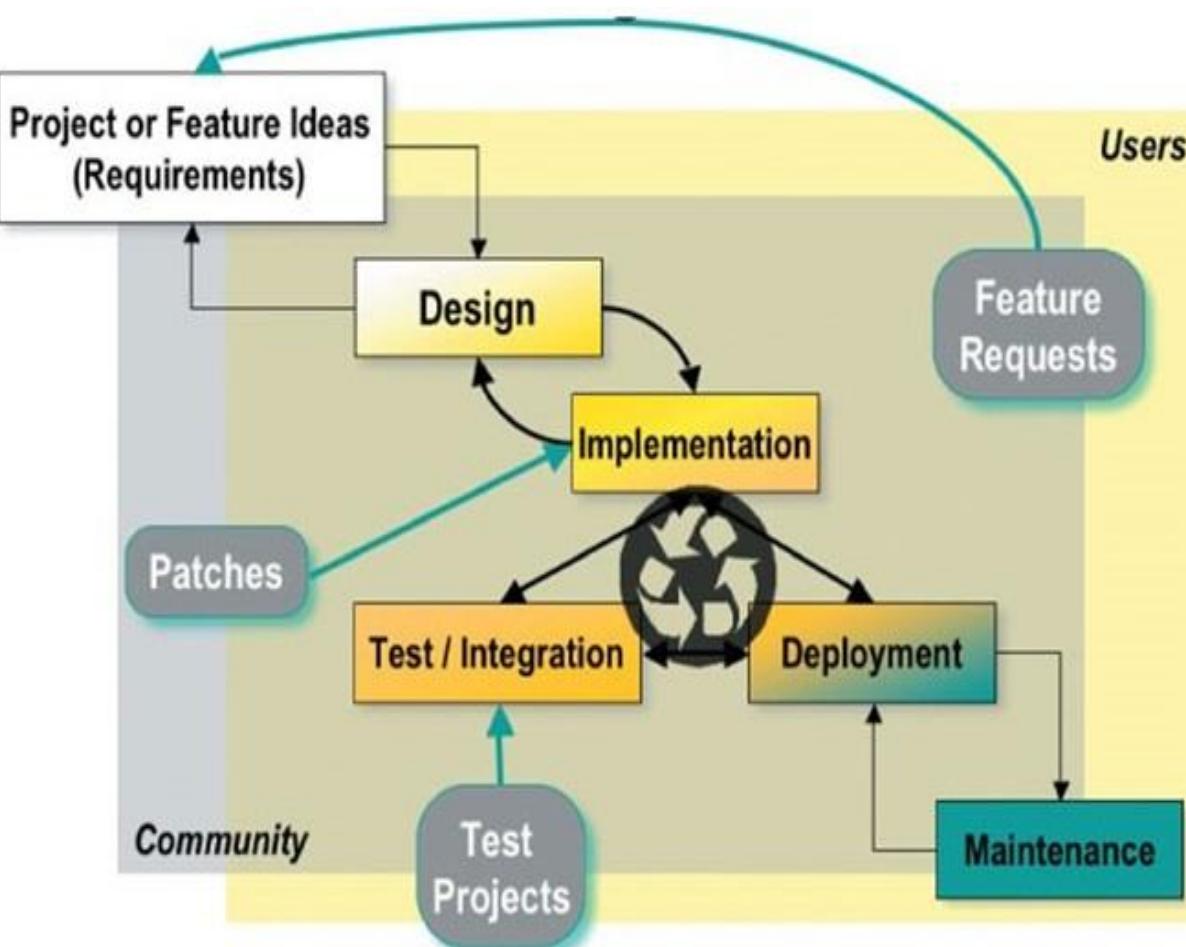


- Open Collaboration Model
 - Community Driven Development Model
 - Open Source Software Development Process Model
-

Open Source Software Development Model

- OSSD Model is different from the traditional waterfall or cascading model of software development.
- The moment the software runs (partially , mostly), it is released as development release (even though it may contain known and unknown bugs) .
- In alignment of Philosophy : “**Release early, release often** “

Open Source Software Development Model



Release early, release often

- Idea for a new project or new feature or functionality
- Design for proposed solution
- Implementation by running (partially or fully)
- Development release (even bug)

Unique Characteristic of OSS Development model

- **Release early, release often**
 - **Peer Review**
 - **Small, incremental changes**
 - **Highly secure code**
 - **Continuous quality improvement**
 - **Test projects**
 - **End user involvement**
-

Agile vs. OSSD Methodologies

As per Agile Manifesto :

- “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software “

As per OSSD Model :

- The **concept or analogy of customers** does not exist in open source software . There are **contributors and users of the projects / product**.
- The **user community , indirectly , serves as the customers**; frequent releases are made - in order to promote peer review, continuous and increased end user involvement.

Agile vs. OSSD Methodologies

As per Agile Manifesto :

“ Welcome changing requirements , even late in development . Agile processes harness change for the customer ‘s competitive advantage “

As per OSSD model :

- **Open source projects are not customer driven** ; and hence often **resist incorporation of major changes** in the requirements.
- However , there always exist the **possibility of incorporating major changes by forking the project in to as separate repository and making changes** - done in case a larger set of community feels the need for the same.

Agile vs. OSSD Methodologies

As per Agile Manifesto,

“Deliver working software frequently , from a couple of weeks to a couple of months , with a preferences to the shorter time scale”

As per OSSD model,

- Open source follows the philosophy of : **release early , release often**
- Delivering of code cycle in case of open – source is much shorter – usually every night.

Agile vs. OSSD Methodologies

As per Agile Manifesto

“Business people and developers must **work together daily throughout the project**”

As per OSSD model ,

There does not exist the concept of “ **Business people** “ in case of open – source projects ;

However , **end-users who participates in the project serve the same role.**

Agile vs. OSSD Methodologies

As per Agile Manifesto ,

- **Build projects around motivated individuals . Give them environment and support they need , and trust them to get the job done”.**

As per OSSD model :

- Participation in open source projects is completely **voluntary; self-motivated and self-driven people join open source community , out of interest and hence motivation is guaranteed**
- Open source projects use selected tools for project management , version control , bug and issue tracking and discussions forums.

Agile vs. OSSD Methodologies

As per Agile Manifesto

- The most efficient and effective method of conveying information to and within a development team is **face to face conversation.**“

As per OSSD model :

- **This is a major point of differences between Agile methodologies and open source**
- Open source projects lay complexly emphasis on written communication over face –to-face communication
- Additionally open source projects can be widely distributed , and do not require collocation

Agile Vs. OSSD Methodologies

As per Agile Manifesto :

- At regular intervals, the team reflects on how to become more effective , and then tunes and adjust its behaviour accordingly.

As per OSSD Model :

- **Self reflection is not** a major activity carried out in most of the open source projects.
- However , some of the mature open source projects tends to evolve and implement some **governance mechanism**

Conclusion

To summarise :

- While both agile and open source methodologies embrace a number of principles and values , some of which are parallel , but some are intersecting as well.

However , both of them :

- Involve continuous interactions with users during all phases of the development;
- Value good design and effective documentation
- Engage self organising and self motivated individuals ; and
- Strive to develop software that is tailored especially for a class of users / customers

Let's build from here

The complete developer platform to build, scale, and deliver secure software.

100+ million
Developers

4+ million
Organizations

330+ million
Repositories

90%
Fortune 100

- GitHub is a code hosting platform for collaboration and version control.
 - GitHub lets you (and others) work together on projects.
-

Development goals of Git

- Speed,
- Simplified design
- Completely distributed
- Strong support for branching (non linear development), involving thousands of parallel branches
- Able to handle large projects like the Linux kernel efficiently; in terms of speed and data size.

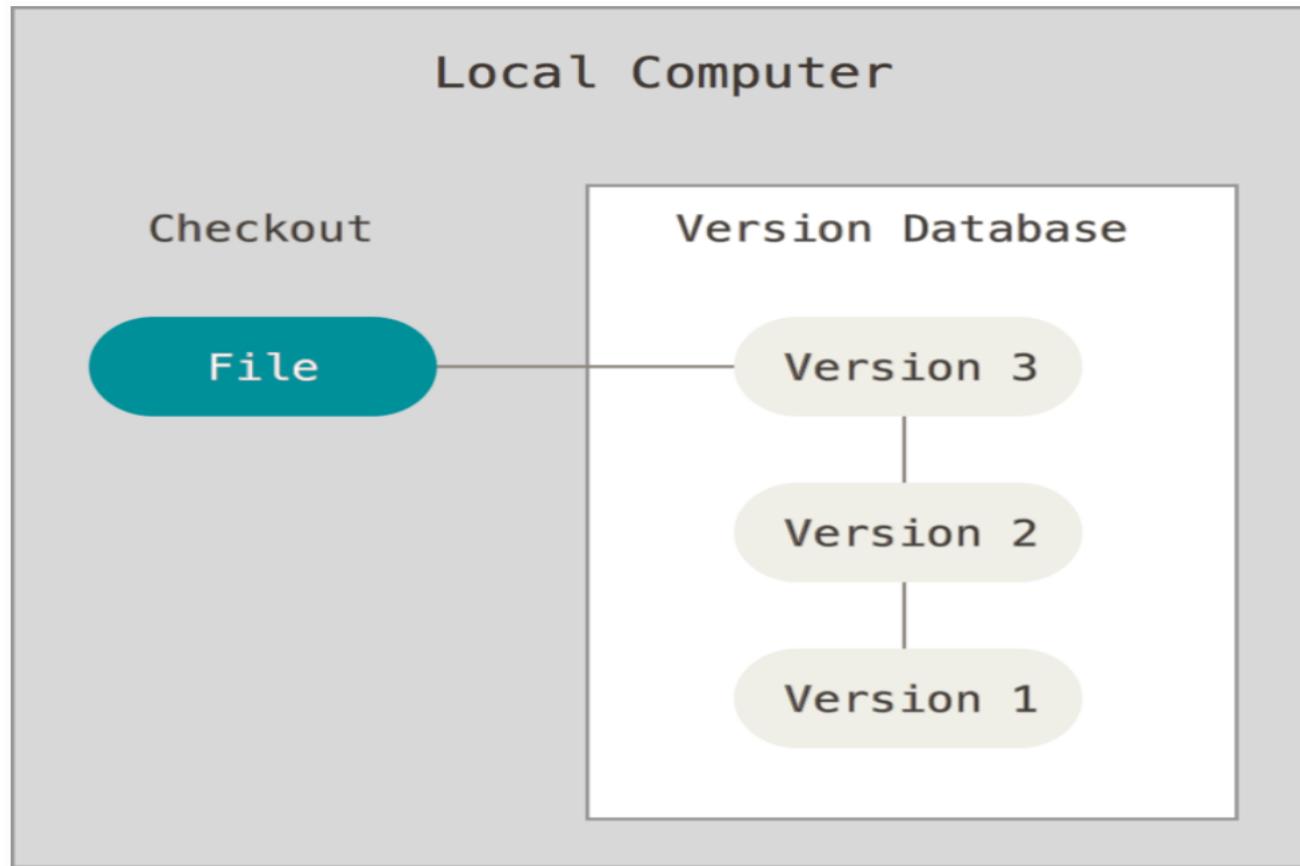
- Git : An **Open-source distributed version control system (DVCS)**
- GitHub CLI : **GitHub CLI is an open source tool for using GitHub from your computer's command line.**
- GitHub Desktop
- GitHub is a **website** that allows developers to upload their Git repositories online – on the Internet

GitHub Enterprise :

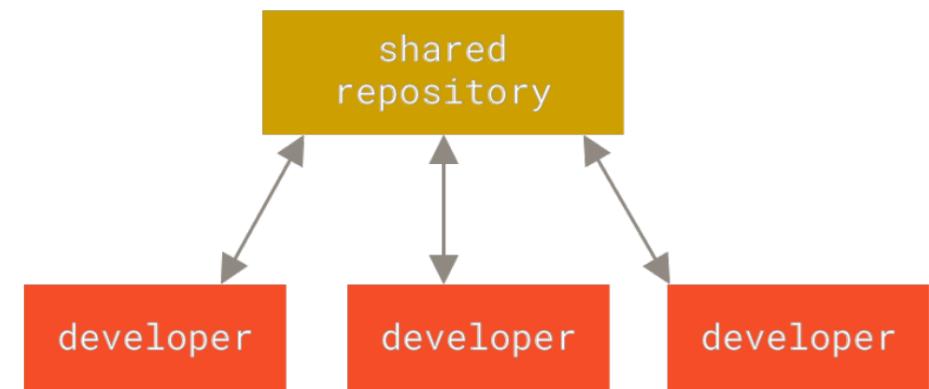
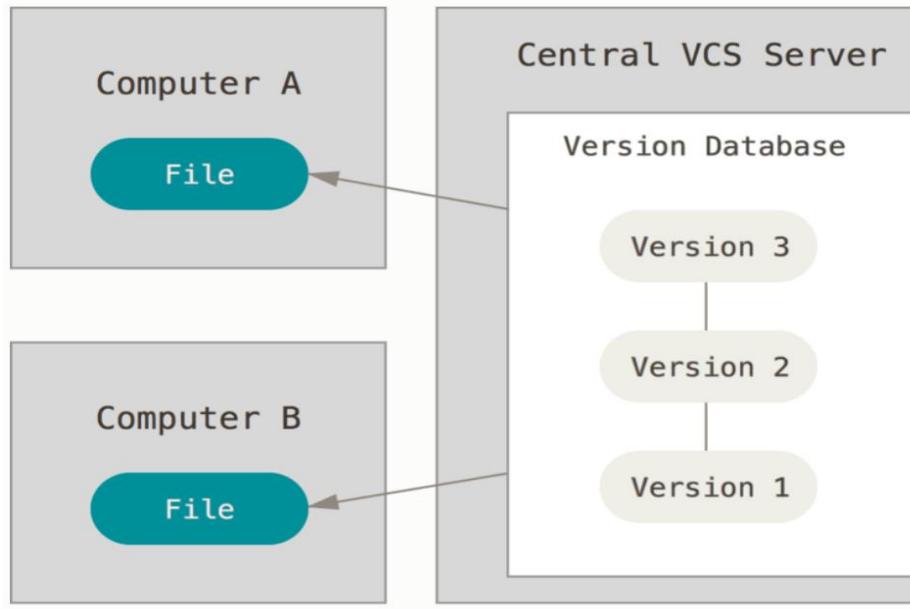
- **GitHub Enterprise Cloud**
- **GitHub Enterprise Server**
- The main difference between the products is that GitHub Enterprise Cloud is hosted by GitHub, while GitHub Enterprise Server is self-hosted.

Local version control system

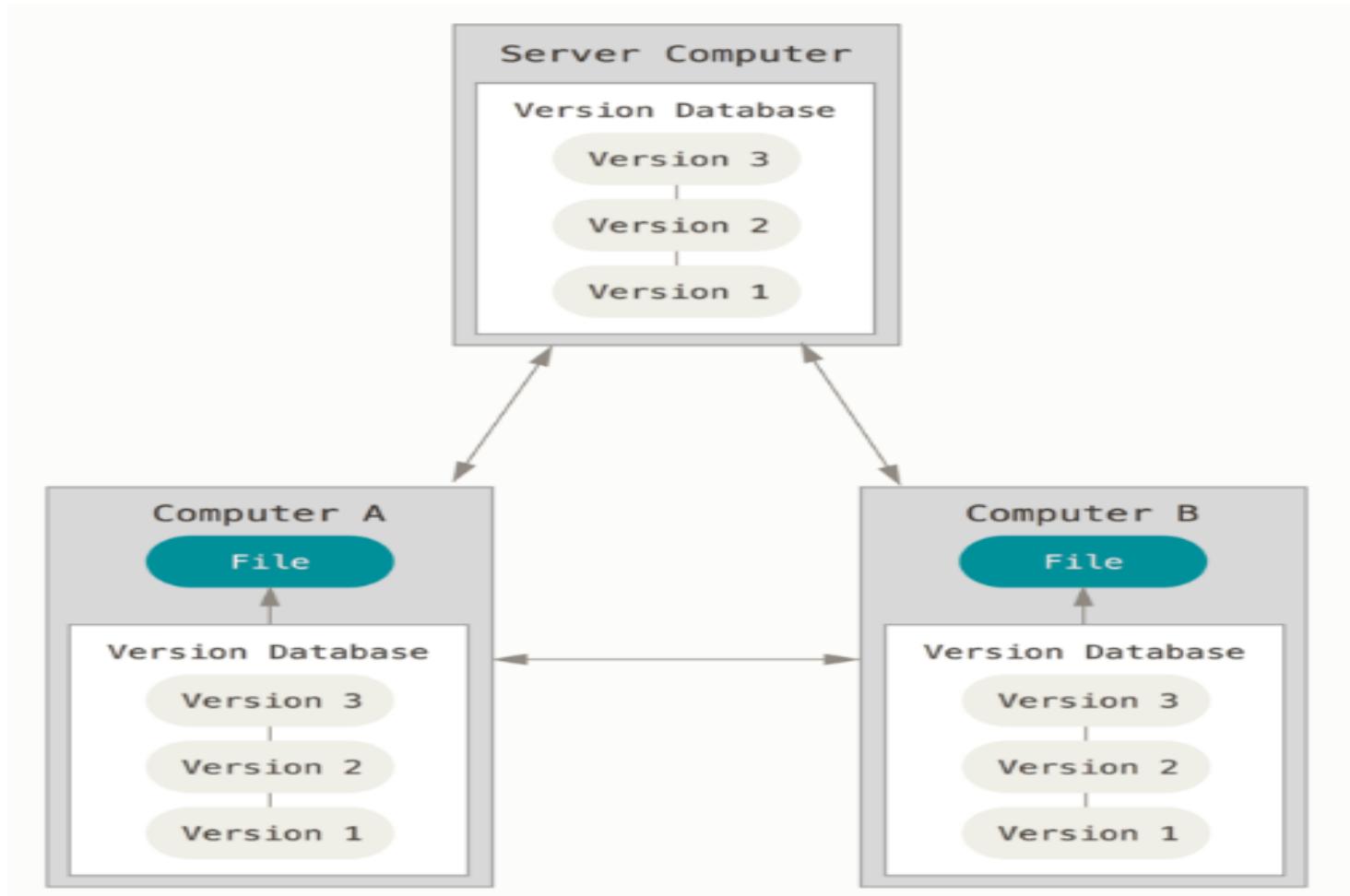
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.



Centralised version control system

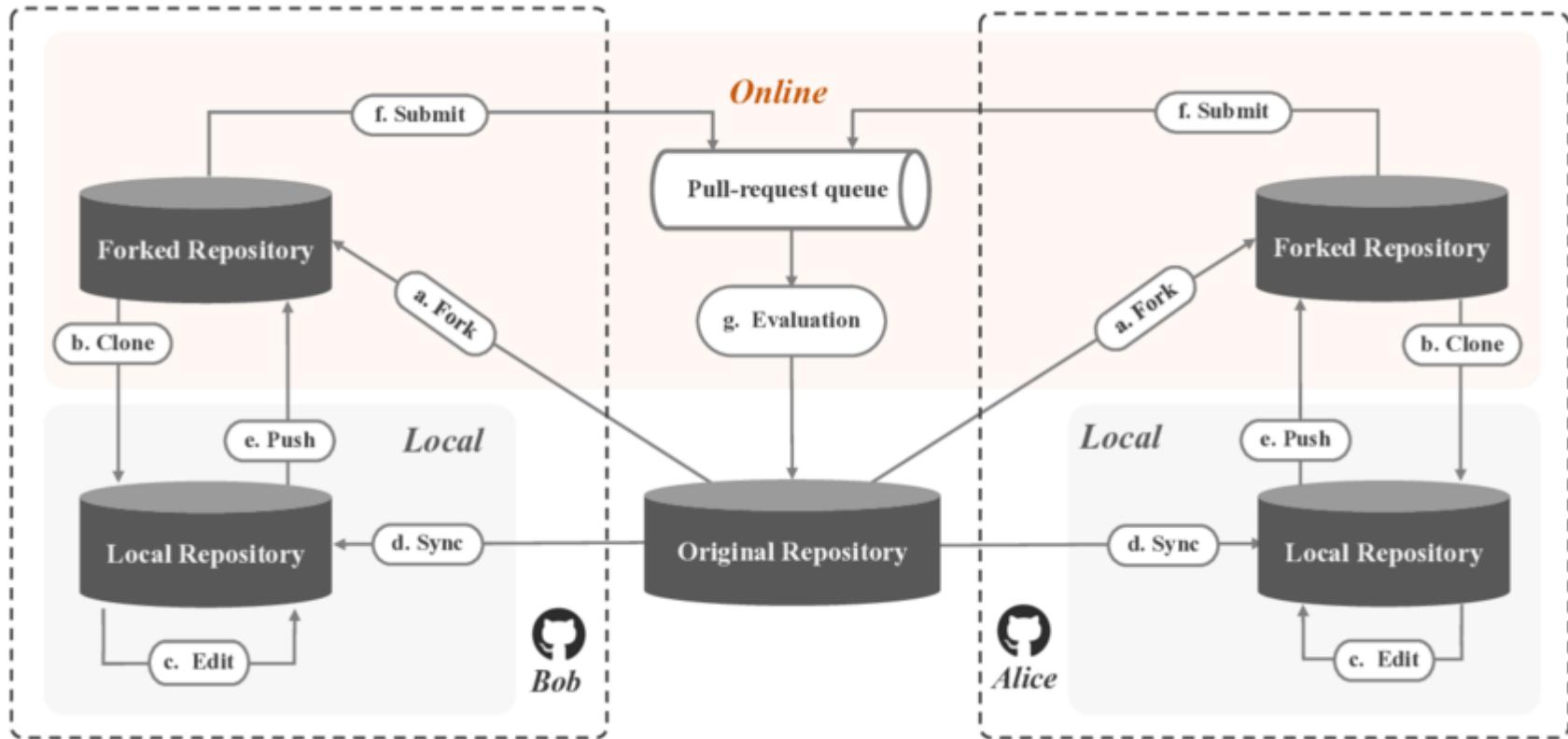


Distributed version control system



Working with git & GitHub

Git Workflow - Pull based model

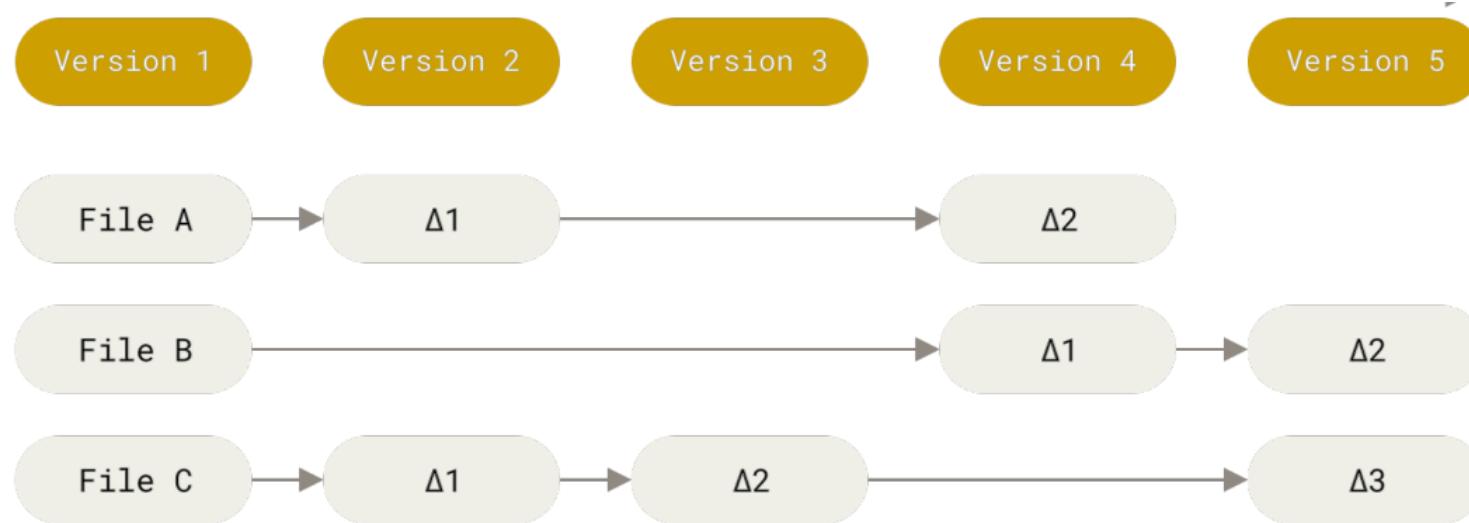


• **Git Hub Accounts :**

- **Personal accounts :** Every person who uses GitHub signs into a personal account.
- **Organization accounts :** An organization account enhances collaboration between multiple personal accounts.
- **Enterprise accounts:** An enterprise account allows central management of multiple organizations

Characteristics of other version control system

- Older VCS like Subversions, Perforce, etc store information **as a list of file based changes knowns as delta –based version control – store only the changes** made to each file over time



Unique Characteristics of Git

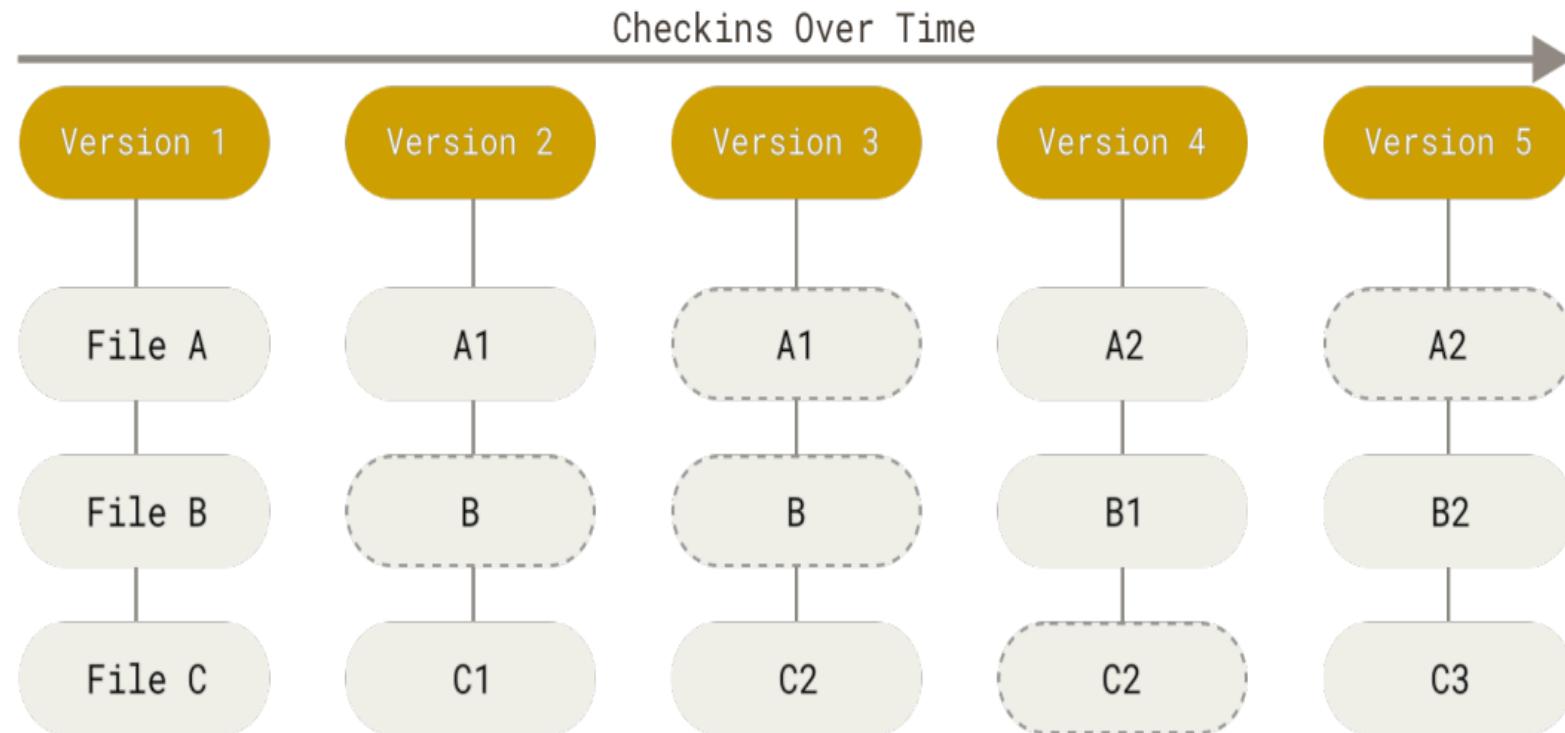


Stores Snapshots , Not differences

- Git does not use the delta approach; rather it uses an approach in which it stores data like a series of snapshots of a miniature file system - **Stream of snapshots**
- With every commit, Git takes a snapshot of all the files at that moment and stores a reference to the snapshot.
- If some files have not changed, Git does not store the file again, but only a link to the previous version of the same file, which is already stored with it.
- This makes git more powerful, efficient and fast as compared to its counterparts
- Also, helps in implementing branching efficiently

Unique Characteristics of Git

Storing data as snapshots of the projects over time



Unique characteristics of git

Nearly Every operation is Local :

- Nearly every **operation in git requires only local files** and resources to execute ; usually no information is required from other computers on the network.
- This is because the **entire history of the project is stored in the local computer of every individual** , making various operation faster.
- In order to browse the entire history of the project , Git **does not need to get the same from the server** – it simply takes it directly from the users' local database and the same immediately available.

Unique characteristic of Git

The **three states in Git** : In Git, a file resides in one of the three states : **modified** , **staged** and **committed**.

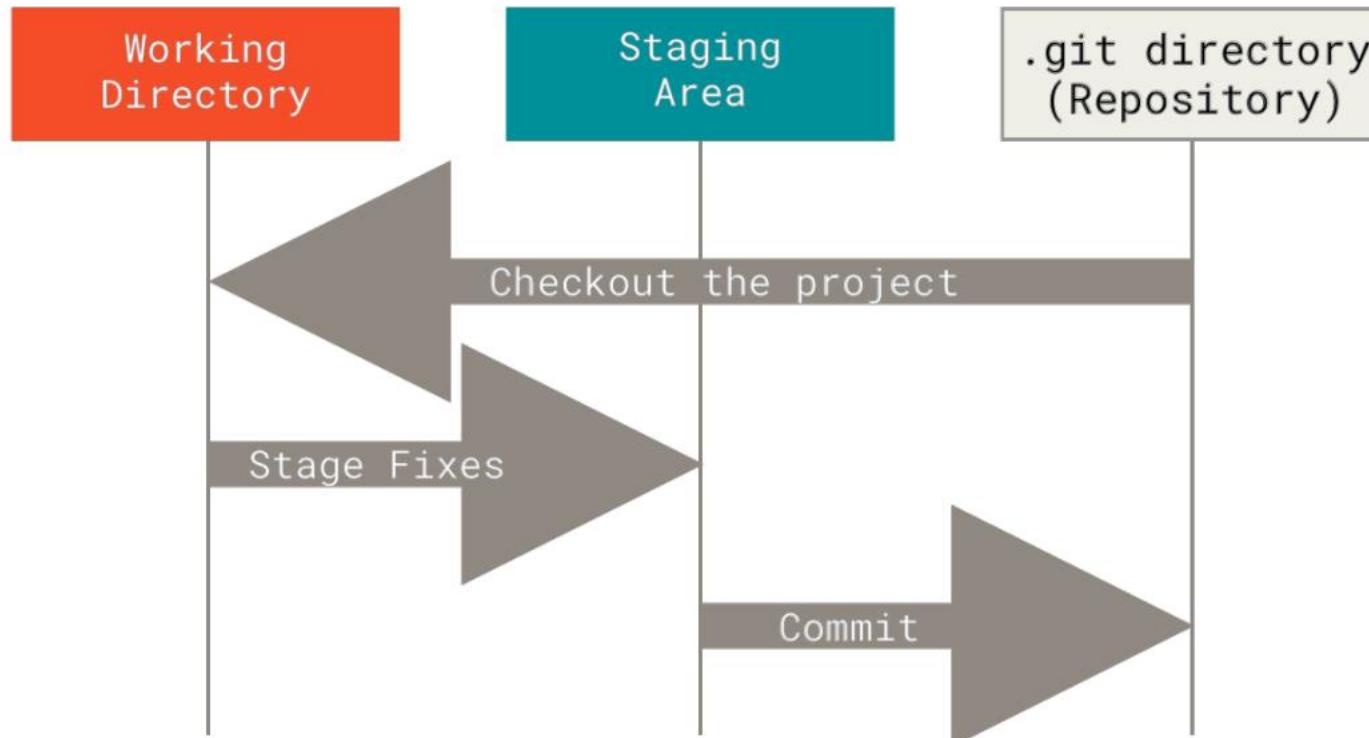
- **Modified state** – The file has changed but the changes have not been committed to the database yet
- **Staged state** - the file has been marked as modified file in its current version and is marked to go in to the next commit snapshot.
- **Committed state** – a file resides in this state is safely stored in the local database

Hence, Git project has three main section :

- The working tree
- The staging area
- And the Git directory

Unique characteristic of Git

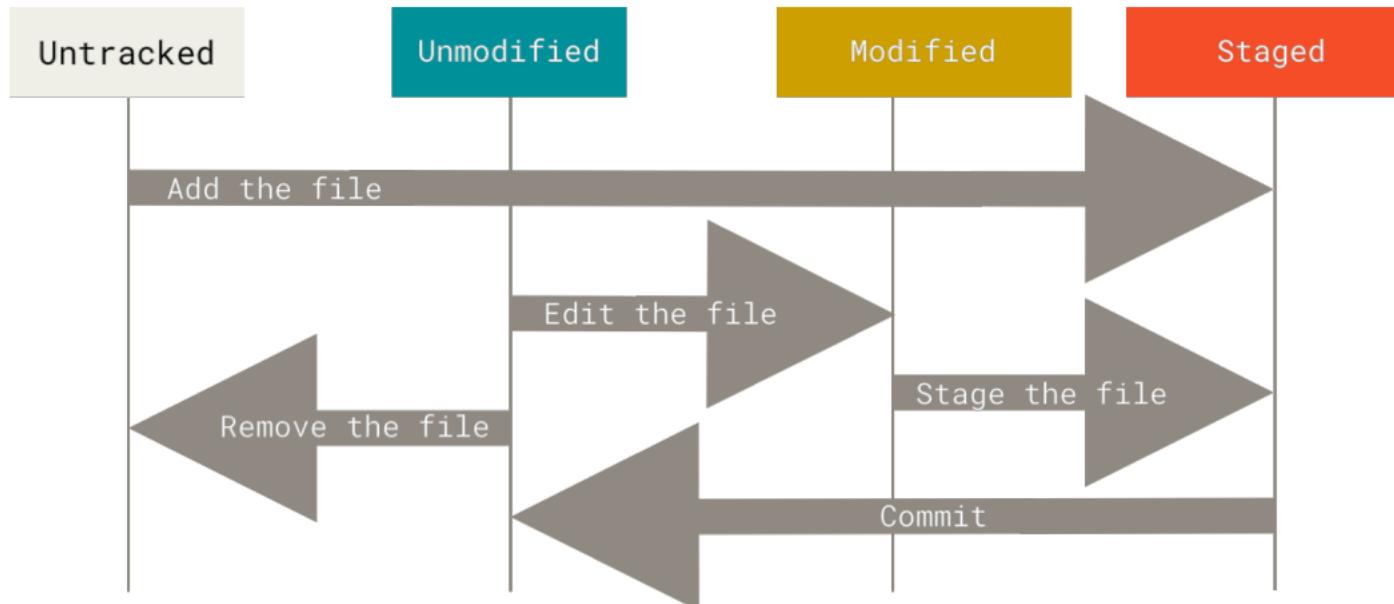
The Three stages in Git



The three states in Git

- The **working directory** is the single checkout of the latest version of the projects. The project files are pulled out of the compressed database in the Git directory and placed on disk for use.
 - The **Staging area** is a file , usually contained in the Git directory, and stores the information about the files that are staged and are hence marked what go into the next commit.
 - The **Git directory** is used by Git to store the metadata and object database for the project. This is the most important part of Git, and is copied to the local system when a repository is cloned from a remote repository
-

Life cycle of the status of Files



Basic Git workflow

The basic git workflow goes like this :

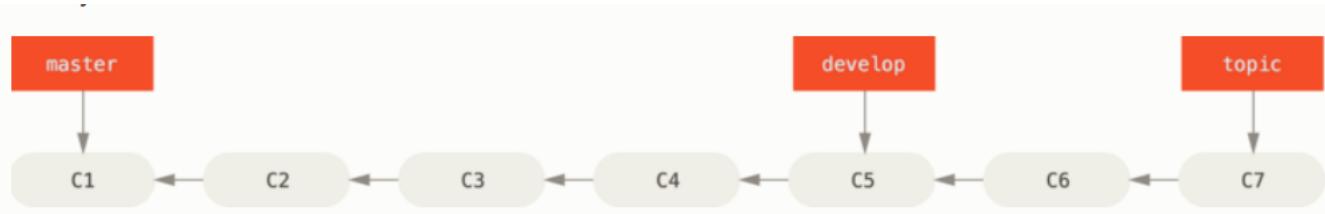
- The files are modified in the working directory
- The files are selectively staged and only those changes that are required to be part of the next commit are moved to the staging area
- Commit is executed, which takes the files as they are in the staging area and stores that snapshot permanently to the Git directory

Git Branching

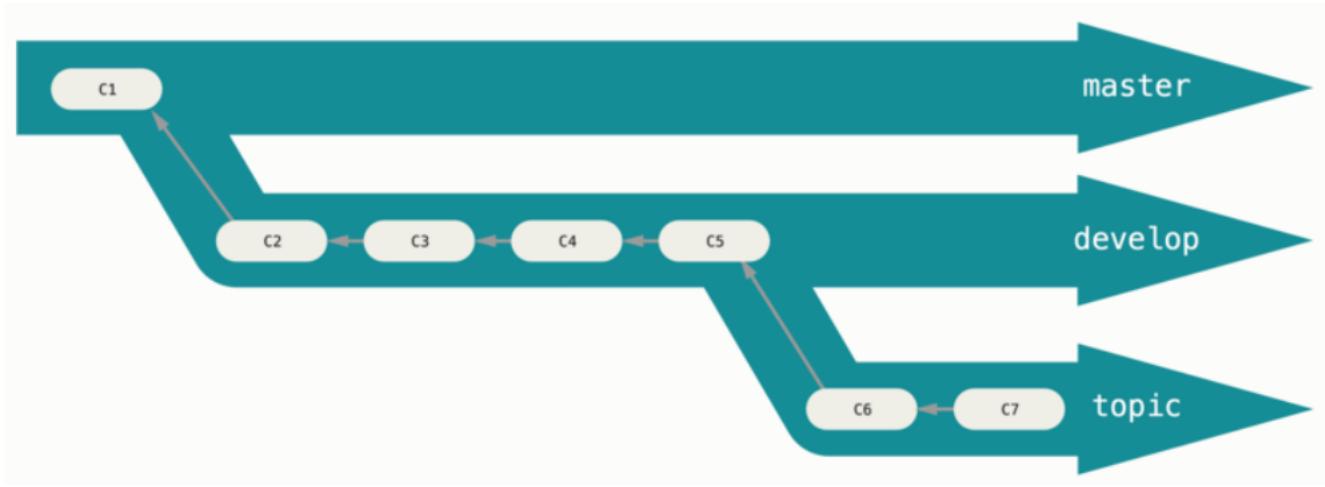
- Need of Branching :
- Helps to work in isolation
- It can orchestrate parallel development allowing developers to work on tasks simultaneously as part of team
- And parallel build and testing ensure developers get the feedbacks they need quickly
- Parallel tasks possible by creating multiple branches :
While you wait for changes from one branch to be pulled by owner , you may start your work on another branch

Branching strategy

Linear View of branching



A “silos” view of progressive stability branching



Branching in Git

- Since Git stores data as a series of snapshots and not as a series of changes sets - it is able to provide support for branching more efficiently as compared to other VCS
- When commit is done,
 - A pointer to the snapshot of the staged content is stored
 - This pointer object also contains author's name and email address , commit message and pointer to the commit (s) immediately before the current commit
 - Zero parents for the initial commit
 - One parent for a normal commit
 - Multiple parents for a commit resulting from a merge of tow or more branches

GitHub essentials

- Repository : Creating , forking , cloning
- Branches
- Commits
- Pull Requests