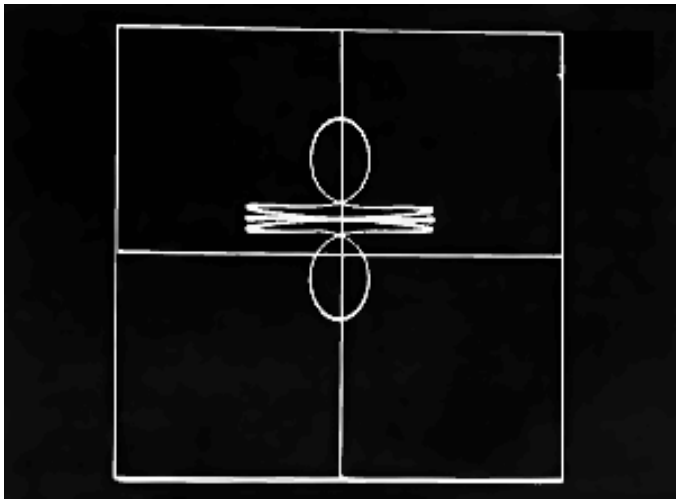
An Open Ecosystem

Linux is an open-source community that focuses on sharing power and responsibility among people instead of centralizing within a select group. The Linux kernel – which acts as the foundation for many Linux-based distributions – is built on an even older framework that matured alongside computers.

> The PDP-7 ran the first Unix code – used for creating the demo video game Space Travel.



An Accidental Movement

Unix was a general-purpose operating system that began to take shape in the mid-1960. This was a collaborative project between the Massachusetts Institute of Technology, Bell Labs, and General Electric. Academic researchers within the burgeoning computer science field experimented with the potential for time-sharing to innovate what was possible with these new digital machines.

UNIX®
An Open Group Standard

Unix itself was based on an even older exploration in computers – an operating system called Multics. Pronounced as "eunuchs", the name itself was intended as a pun on it's predecessor. Multics had yielded truly innovative ideas, but it's exploratory nature didn't yield immediate profit potential.

> What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication.
>
> — Dennis Richie, UNIX pioneer

The original AT&T Unix – created in 1969 – was a proprietary and closed-source operating system first investigated by Bell Labs. As the result of a result of a 1958 ruling by the US Department of Justice, AT&T was forbidden from entering into the computer business under consent decree. This was a part of the larger breakup of the Bell systems that continued through the 1980s.

This meant that AT&T was required to license it's non-telephone technology to anyone that asked. While Unix was intended for use within their labs, they began licensing it to colleges and corporations for a modest fee. This lenient licensing scheme played an important part in the widespread adoption of Unix and the eventual open-source movement.

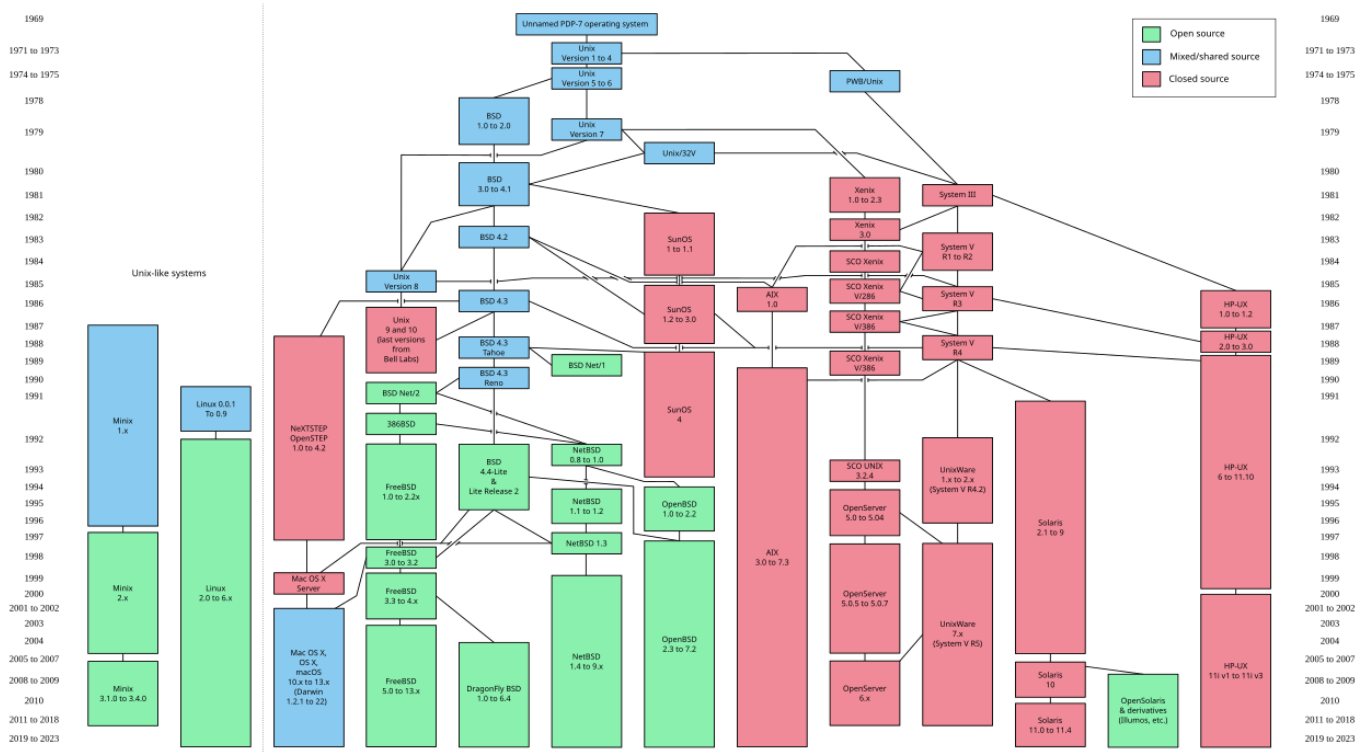
Cathedral and the Bazaar

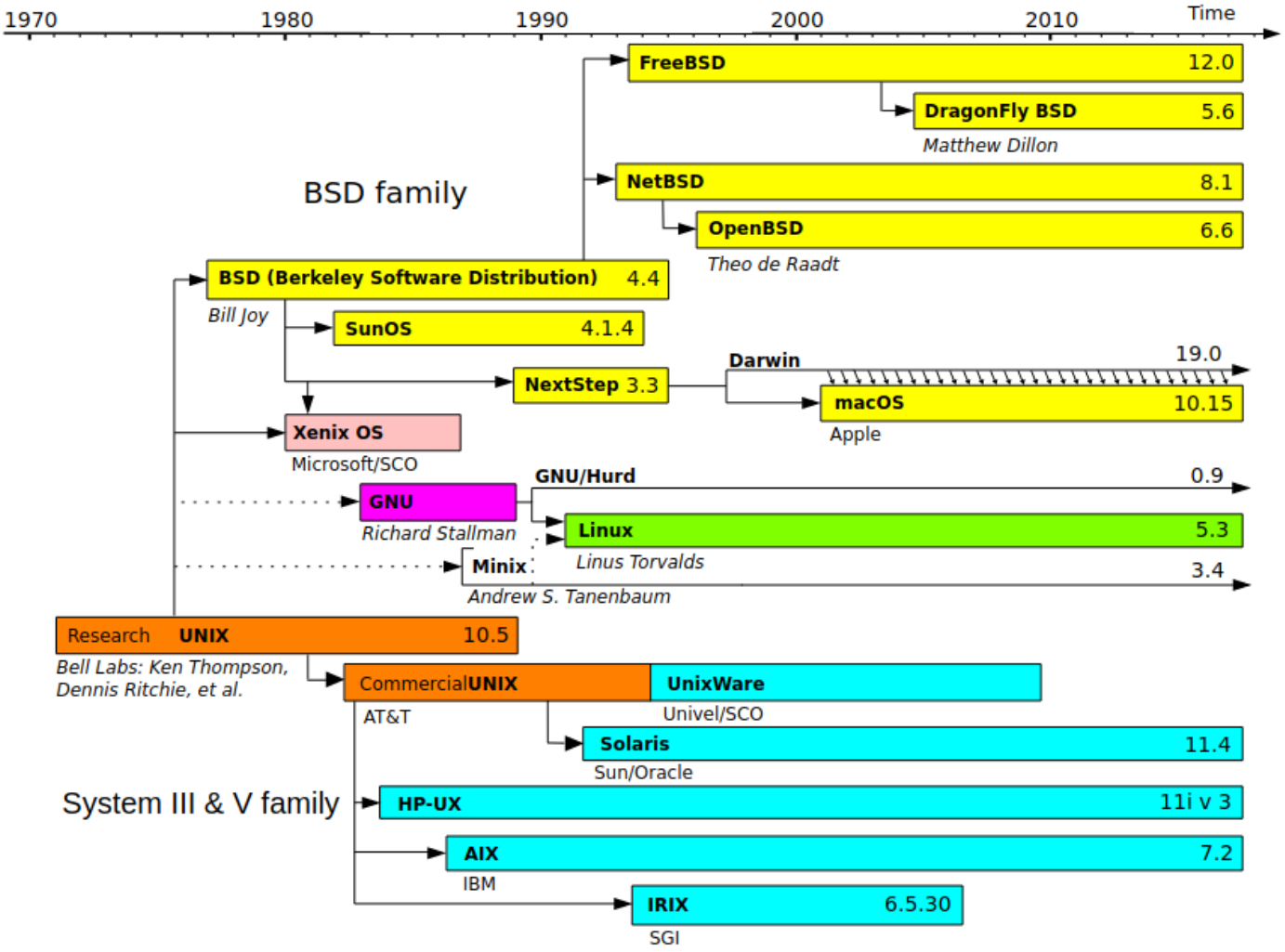
During the early days of computers, programmers and researchers were one and the same. While developing programming languages like C – the backbone of Unix – we were also exploring what computers could accomplish for the first time. To that end, it was common to share software and learn from each other while studying computing.

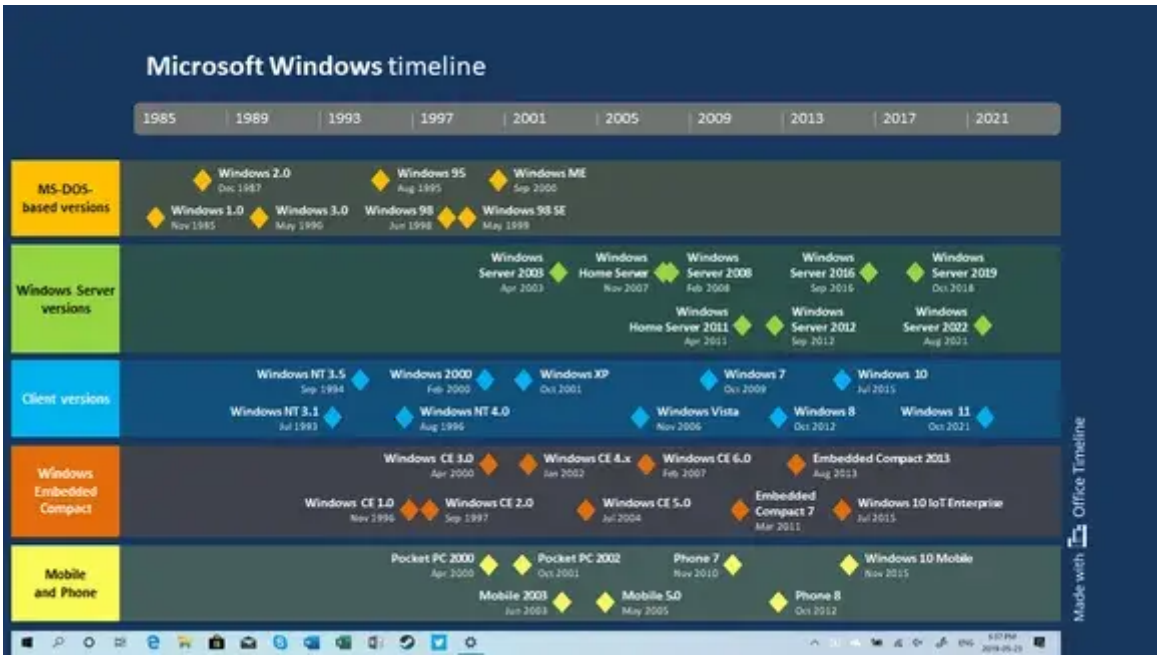
> Cathedral and the Bazaar was a foundational book by Eric S. Raymond about opposing software project management styles.

Unix was revolutionary not only as an operating system, but because it came bundled with a complete copy of the source code used to build it. This allowed researchers to modify the code to fulfill their needs while also enabling corporations to create their own custom Unix distributions – for use in-house or as a marketable product. This led to a proliferation of Unix operating systems, each with exciting new features.

Windows vs Mac vs Linux vs Unix timeline graphic


Open source
Mixed/shared source
Closed source
Unix-like systems
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
Unix/32V
PWB/Unix
BSD 1.0 to 2.0
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD Net/1
BSD Net/2
386BSD
BSD 4.4-Lite & Lite Release 2
Unix Version 8
Unix 9 and 10 (last versions from Bell Labs)
SunOS 1 to 1.1
SunOS 1.2 to 3.0
SunOS 4
AIX 1.0
AIX 3.0 to 7.3
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix V/286
SCO Xenix V/386
SCO UNIX 3.2.4
OpenServer 5.0.4
OpenServer 5.0.5 to 5.0.7
OpenServer 6.x
System III
System V R1 to R2
System V R3
System V R4
UnixWare 1.x to 2.x (System V R4.2)
UnixWare 7.x (System V R5)
Solaris 2.1 to 9
Solaris 10
Solaris 11.0 to 11.4
OpenSolaris & derivatives (Illumos, etc.)
HP-UX 1.0 to 1.2
HP-UX 2.0 to 3.0
HP-UX 6 to 11.10
HP-UX 11i v1 to 11i v3
Minix 1.x
Minix 2.x
Minix 3.1.0 to 3.4.0
Linux 0.0.1 To 0.9
Linux 2.0 to 6.x
NeXTSTEP OpenSTEP 1.0 to 4.2
Mac OS X Server
Mac OS X, OS X, macOS 10.x to 13.x (Darwin 1.2.1 to 22)
FreeBSD 1.0 to 2.2x
FreeBSD 3.0 to 3.2
FreeBSD 3.3 to 4.x
FreeBSD 5.0 to 13.x
DragonFly BSD 1.0 to 6.4
NetBSD 0.8 to 1.0
NetBSD 1.1 to 1.2
NetBSD 1.3
NetBSD 1.4 to 9.x
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 7.2
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2002
2003
2004
2005 to 2007
2008 to 2009
2010
2011 to 2018
2019 to 2023


1970
1980
1990
2000
2010
Time
BSD family
FreeBSD 12.0
DragonFly BSD 5.6
Matthew Dillon
NetBSD 8.1
OpenBSD 6.6
Theo de Raadt
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
NextStep 3.3
Darwin 19.0
macOS 10.15
Apple
Xenix OS
Microsoft/SCO
GNU
Richard Stallman
GNU/Hurd 0.9
Linux 5.3
Linus Torvalds
Minix 3.4
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson, Dennis Ritchie, et al.
CommercialUNIX
AT&T
UnixWare
Univel/SCO
Solaris 11.4
Sun/Oracle
System III & V family
HP-UX 11i v 3
AIX 7.2
IBM
IRIX 6.5.30
SGI



Software – like hardware – became increasingly commercialized throughout the 1970s. Corporations sought to mold hardware into compact personal devices while simultaneously fashioning software into the killer application that would draw consumers to their products. The Unix Wars throughout the 1980s exacerbated the friction between vendors as the operating system became fragmented between multiple competing standards.

As corporations navigated this space, many preferred to follow the proprietary development model. These release cycles are often measured in years – meaning that software was released as polished product with meticulous planning put into final 'gold' release. On the flip side, bug fixes and feature requests could take years to manifest in the publicly available product. Important software updates may never emerge – or may even be released as part of the product's successor.

This 'release late—release rarely' philosophy arises when the software developers regard their project as a consumer product. While the product is marketed towards consumers, their role in the creative process is rather limited. Their feedback is often collected reactively during formative beta testing – or even after the product is released to the public.

Proprietary software is often "closed-source", meaning that the code to create it is private and legally protected – or even a trade secret. The code is compiled into a binary file containing the raw binary data – ones and zeros – used to control a computer system. The information it is not human-readable and only applies to a specific platform – such as Windows, MacOS or Debian Linux.

This makes it relatively difficult to reverse engineer, but it also means that the code was created to run efficiently on your specific computer system. The software is compiled towards 'minimum system requirements' and more advanced hardware is rarely leveraged to your advantage.

Software Freedoms

During the 1970s, the original computer hacker culture – who enjoyed the creative challenge of overcoming hardware and software limitations – formed within academic institutions.

It was around this time that the Free Software Movement began to take shape. Researchers continued to develop software collaboratively by sharing their discoveries and the source code that powered them. This was foundational to the continued growth of the Unix experiment.

In 1984, Richard Stallman resigned from his position at MIT citing that proprietary software stifled collaboration by limiting his labs ability to share source code. He began work on the GNU Project – which stands for *GNU's Not Unix* – and represented an idealized "free" operating system. It behaved almost exactly like Unix to attract developers, but the source code would be available for anyone to modify.

> The word "free" in our name does not refer to price; it refers to freedom. First, the freedom to copy a program and redistribute it to your neighbors, so that they can use it as well as you. Second, the freedom to change a program, so that you can control it instead of it controlling you; for this, the source code must be made available to you.
>
> — GNU's Bulletin, Volume 1

The Free Software Foundation he sparked – through his call-to-action known as the GNU Manifesto – initially caused some confusion. He often had to explain that he meant "free" as in "freedom" not as in "beer". This led to the foundation of the movement: the four software freedoms.

Counter_1	**Freedom 1** The freedom to run the program as you wish, for any purpose.
Counter_2	**Freedom 2** The freedom to study how the program works, and change it so it does your computing as you wish.
Counter_3	**Freedom 3** The freedom to redistribute copies so you can help your neighbor.
Counter_4	**Freedom 4** The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes.

Fulfilling these freedoms required unrestricted access to the underlying source code. Through GNU, a new decentralized model of development emerged that enabled everyone to contribute bug fixes, code suggestions and feature requests. Communication took place primarily on internet newsgroups – one of the first examples of a digital bulletin board.

GNU developed in sharp contrast to proprietary software with many open-source projects following the 'release early—release often' development philosophy. These software programs are not generally viewed as a consumer product, but as a tool to reach an end.

While these projects may feel less polished, users have the power to add their voice throughout the entire development process. This means the potential for bugs to be fixed promptly and – depending on community feedback – features can be quickly integrated into the ongoing evolution.

The GNU Project is an umbrella for the hundreds of smaller projects that are required to build an operating system. While developed through collaboration, these constituent projects are often produced independently of the others.

Modular by Design

While lying the foundations for Unix, computer scientists were careful to consider it's design philosophy. They decided that Unix should provide a simple set of tools – each able to perform limited function with well-defined parameters. This enabled a modular and decentralized approach to developing the new operating system.

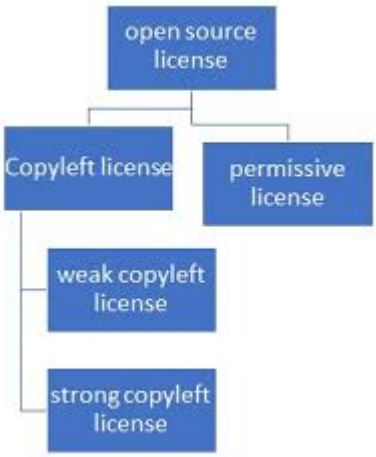
This philosophy disconnected the lifecycle of applications from each other – as well as from the operating system. This allowed all of the communities surrounding these projects to make their own decisions about how their software would be developed. There are a myriad combinations of both competing and complimentary projects.

This can leads to collective communities embracing a core ideology or philosophy. The modular nature of Unix enabled the operating system and software to run independently.

When there is an error or security vulnerability in a module, the damage is more localized rather than affecting the whole. This allows you to interact with your computer without a graphical interface through console command. The Windows graphical interface, by comparison, is heavily integrated into the Windows kernel. There is functionally no Windows without windows.

Licensing

GNU Software was released under the GNU General Public License and allowed full use by anyone – with specific restrictions. This was an early example of a copyleft license which mandated that all derivative works have a similar license ensuring reciprocity.



By 1989, University of California, Berkeley introduced BSD – or the Berkeley Software Distribution – and created the first publicly accessible Unix operating system. By rewriting proprietary AT&T Unix code from the ground up, they released BSD openly to facilitate open collaboration.

They created their own permissive software license that placed barely any restrictions on how you could use the software, while also providing no warranty. This even allowed proprietization – meaning it could be used within private, "closed-source" programs.

Share-alike

https://en.m.wikipedia.org/wiki/Share-alike

Weak copyleft licenses also obligate users to release their changes. However, this requirement applies to a narrower set of code. The Mozilla Public License 2.0 and the CDDL (Common Development and Distribution License) are examples of weak copyleft licenses that illustrate this principle. If a user keeps the licensed code in separate files, they can then combine it with additional and/or modified code to create an aggregate work. The newly added files may be released under a different license or kept proprietary (closed-source). This is sometimes referred to as file-based copyleft. Another example is the LGPL, which mainly applies to libraries. Any changes to the library must be released under the same license, but a work that simply uses the library is exempt.

Linux, on the other hand, provides the user with the option to use pre-compiled binaries or to compile from source. This is much more powerful and fits the needs of a far broader spectrum of users.

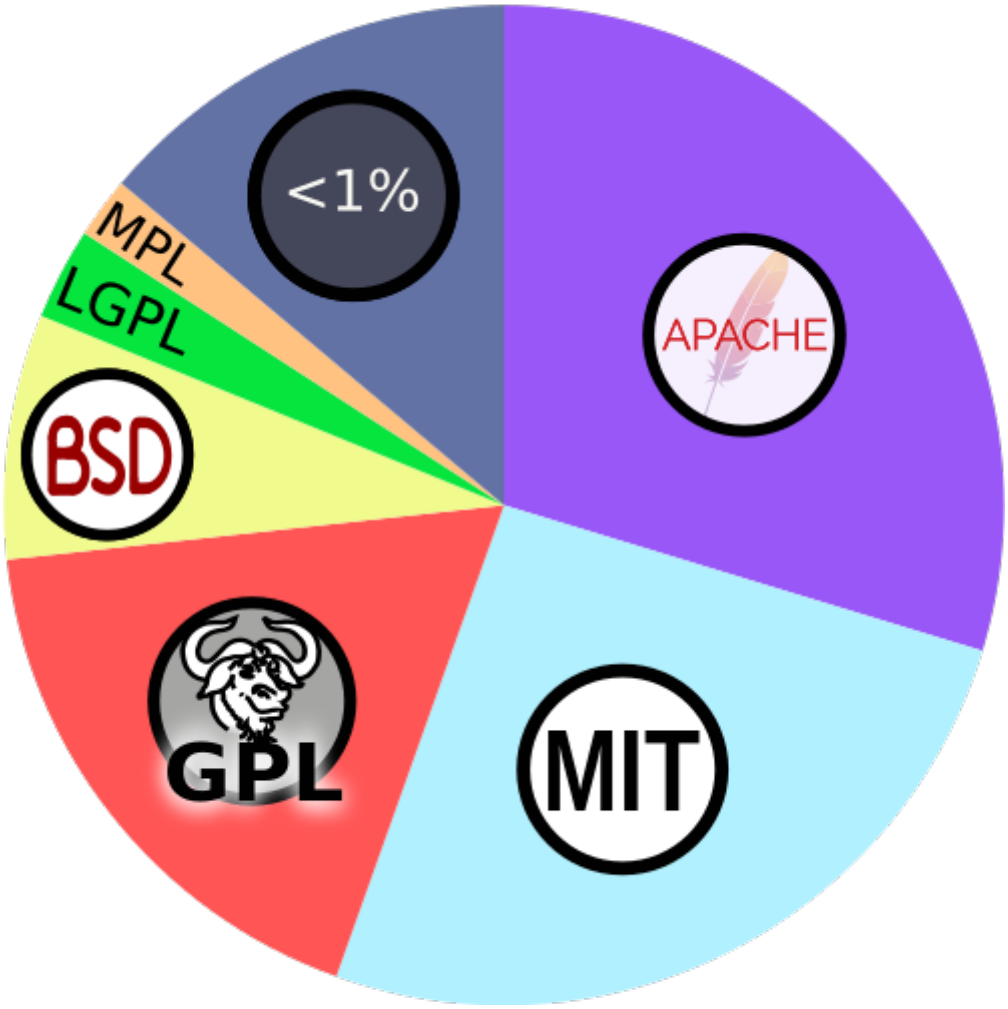
The fact that the software licenses explicitly permit redistribution, however, provides a basis for larger-scale projects that collect the software produced by stand-alone projects and make it available all at once in the form of a Linux distribution.

https://choosealicense.com/

Creative commons vs copy left/permissive

Creative Commons (CC) and open source licenses are both about sharing and usage rights, but they differ in their primary focus and application. Creative Commons licenses are designed for creative works like music, art, and writing, while open source licenses are primarily for software code. Open source licenses allow for free use, modification, and redistribution of the software, while CC licenses offer a broader range of options for sharing and using creative content.

https://www.mend.io/resources/blog/open-source-licenses-trends-and-predictions/



https://en.m.wikipedia.org/wiki/Creative_Commons

https://creativecommons.org/share-your-work/

	Copyleft Licenses
Counter_1	**GPL** A strict copy left license that comes with many conditions for usage within derivative software while providing express patent allowance.
Counter_2	**The Unlicense** Test
Counter_3	**Mozilla Public License 2.0** https://en.m.wikipedia.org/wiki/GNU_Lesser_General_Public_License

	Permissive Licenses
Counter_1	**Apache** A permissive license that allows that this software can be incorporated into larger projects that can be released under a different license.
Counter_2	**MIT** This straightforward license only requires that the licensing information is shown, otherwise the software can be used freely for commercial or personal usage
Counter_3	**BSD 3-Clause** https://en.m.wikipedia.org/wiki/BSD_licenses

Free – or Open?

During the early 1990s, the GNU Project proceeded until it neared completion – the only thing it was missing was a kernel. This integral system handles all interactions between software and hardware within a computer system. Without it, the operating system wouldn't even be able to operate. Their free kernel – known as GNU Hurd – was still incomplete.

Linus Torvalds, operating independently of the GNU Project, created the first version of the Linux kernel during his time as a computer science student. It was also released under the copy left General Public License. GNU adopted Linux as it's kernel – which was now rapidly growing into a community.

The resulting operating system is now generally referred to as Linux – even though there has been a movement to change this to GNU/Linux. Linux was quickly adopted as the flag-ship project of the newly forming Open-Source Movement.

A collective of developers concluded that the Free Software Movement's social activism was not appealing to companies. The group – later known as the Open Source Initiative – felt that more emphasis needed to be placed on the business potential for openly sharing and collaborating on source code.

https://en.m.wikipedia.org/wiki/The_Open_Source_Definition#Debian_Free_Software_Guidelines

Access to the source code:

2. Freedom of Modification:

3. Free Redistribution:

4. Permissive licence:

5. Community and Collaboration:

6. Transparency and security

The modular nature often allowed projects to work together regrdles of different philosophy. Debian began to provide criteria for compatible license.

Free Software Movement vs Open Source Movement

https://e.foundation/what-is-the-difference-between-free-software-and-open-source-software/

Philosophy and objectives:
– Free Software : Focuses on the ethical and moral freedoms of users. The Free Software Foundation (FSF) emphasises the user's freedom to control the software and co-operate with the community.
– Open Source : Emphasises the practical benefits such as quality, flexibility and innovation of sharing source code. The Open Source Initiative (OSI) focuses less on ethical aspects and more on the development model.

1993: Over 100 developers work on the Linux kernel. With their assistance the kernel is adapted to the GNU environment, which creates a large spectrum of application types for Linux. The oldest currently existing Linux distribution, Slackware, is released for the first time. Later in the same

year, the Debian project is established. Today it is the largest community distribution.

https://lists.debian.org/debian-announce/1997/msg00017.html

Debian social contract

Debian believes the makers of a free software operating system should provide guarantees when a user entrusts them with control of a computer. These guarantees include:

- Ensuring that the operating system remains open and free.
- Giving improvements back to the community that made the operating system possible.
- Not hiding problems with the software or organization.
- Staying focused on the users and the software that started the phenomenon.
- Making it possible for the software to be used with non-free software.

Dfsg

Debian Free Software Guidelines (DFSG)

Debian thought unity between free and open software alongside proprietary software was possible - even preferable

Community Security

In software development, **Linus's law** is the assertion that "given enough eyeballs, all bugs are shallow". The law was formulated by Eric S. Raymond in his essay and book *The Cathedral and the Bazaar* (1999), and was named in honor of Linus Torvalds.[1][2]

A more formal statement is: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone." Presenting the code to multiple developers with the purpose of reaching consensus about its acceptance is a simple form of software reviewing. Researchers and practitioners have repeatedly shown the effectiveness of reviewing processes in finding bugs and security issues.[3]

The persistence of the Heartbleed security bug in a critical piece of code for two years has been considered as a refutation of Raymond's dictum.[6][7][8][9] Larry Seltzer suspects that the availability of source code may cause some developers and researchers to perform less extensive tests than they would with closed source software, making it easier for bugs to remain.[9] In 2015, the Linux Foundation's executive director Jim Zemlin argued that the complexity of modern software has increased to such levels that specific resource allocation is desirable to improve its security. Regarding some of 2014's largest global open source software vulnerabilities, he says, "In these cases, the eyeballs weren't really looking".[8] Large scale experiments or peer-reviewed surveys to test how well the mantra holds in practice have not been performed.[10]

Empirical support of the validity of Linus's law[11] was obtained by comparing popular and unpopular projects of the same organization. Popular projects are projects with the top 5% of GitHub stars (7,481 stars or more). Bug identification was measured using the corrective commit probability, the ratio of commits determined to be related to fixing bugs. The analysis showed that popular projects had a higher ratio of bug fixes (e.g., Google's popular projects had a 27% higher bug fix rate than Google's less popular projects). Since it is unlikely that Google lowered its code quality standards in more popular projects, this is an indication of increased bug detection efficiency in popular projects.

GNU programs have been shown to be more reliable than their proprietary Unix counterparts.[32][33]

Revision #25
Created 11 May 2025 04:59:52 by metaphorraccoon
Updated 31 May 2025 23:47:01 by metaphorraccoon