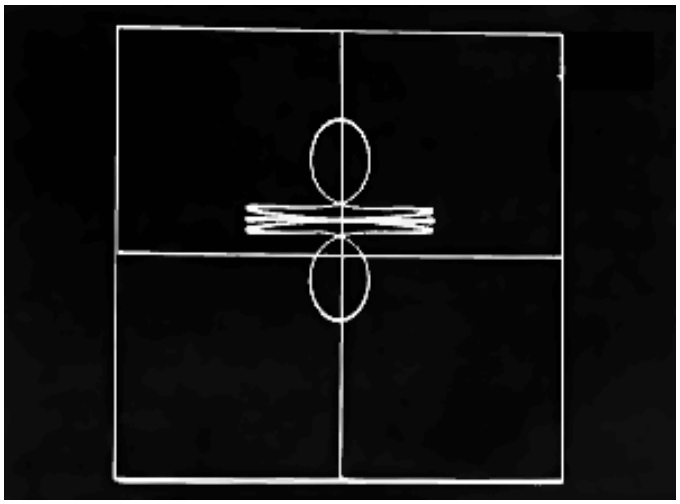
An Open Ecosystem

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-1960s. 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 wasn't immediately profitable and the project was eventually

shuttered.



> 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.

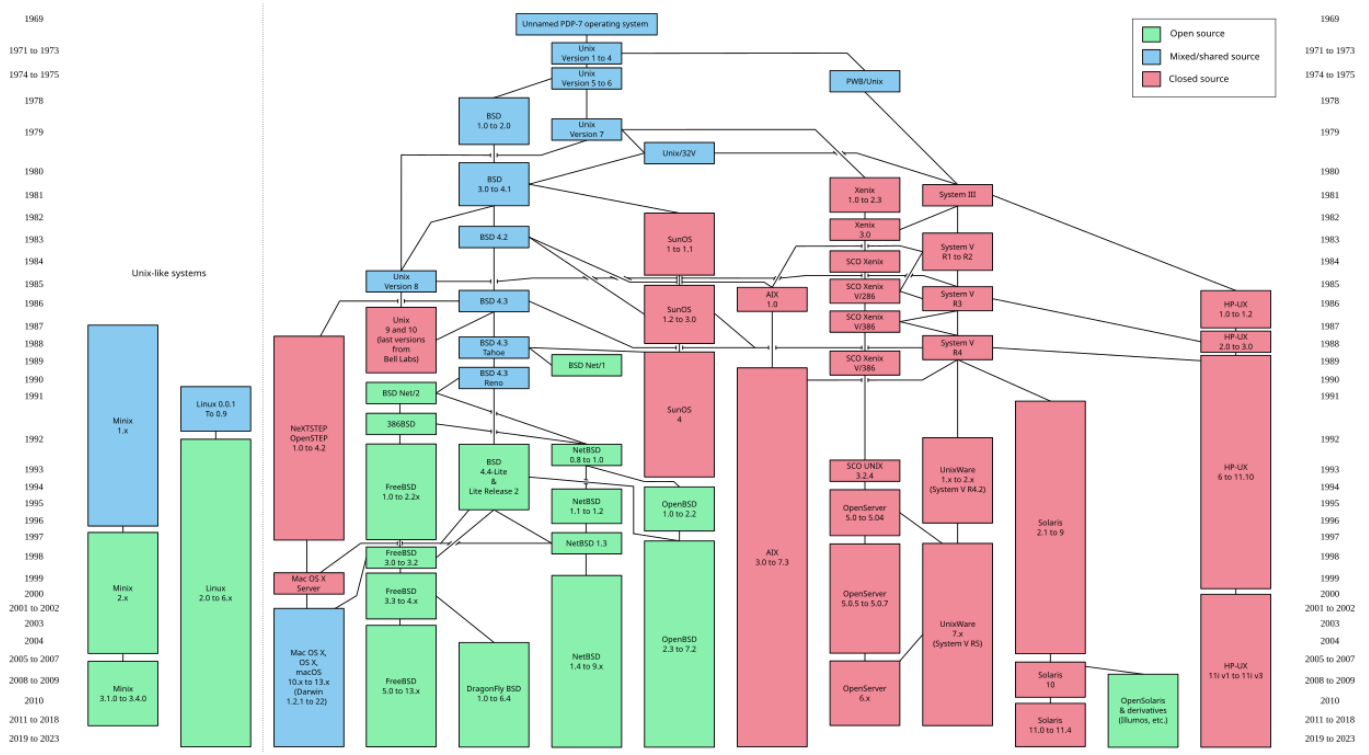
Cathedral and the Bazaar

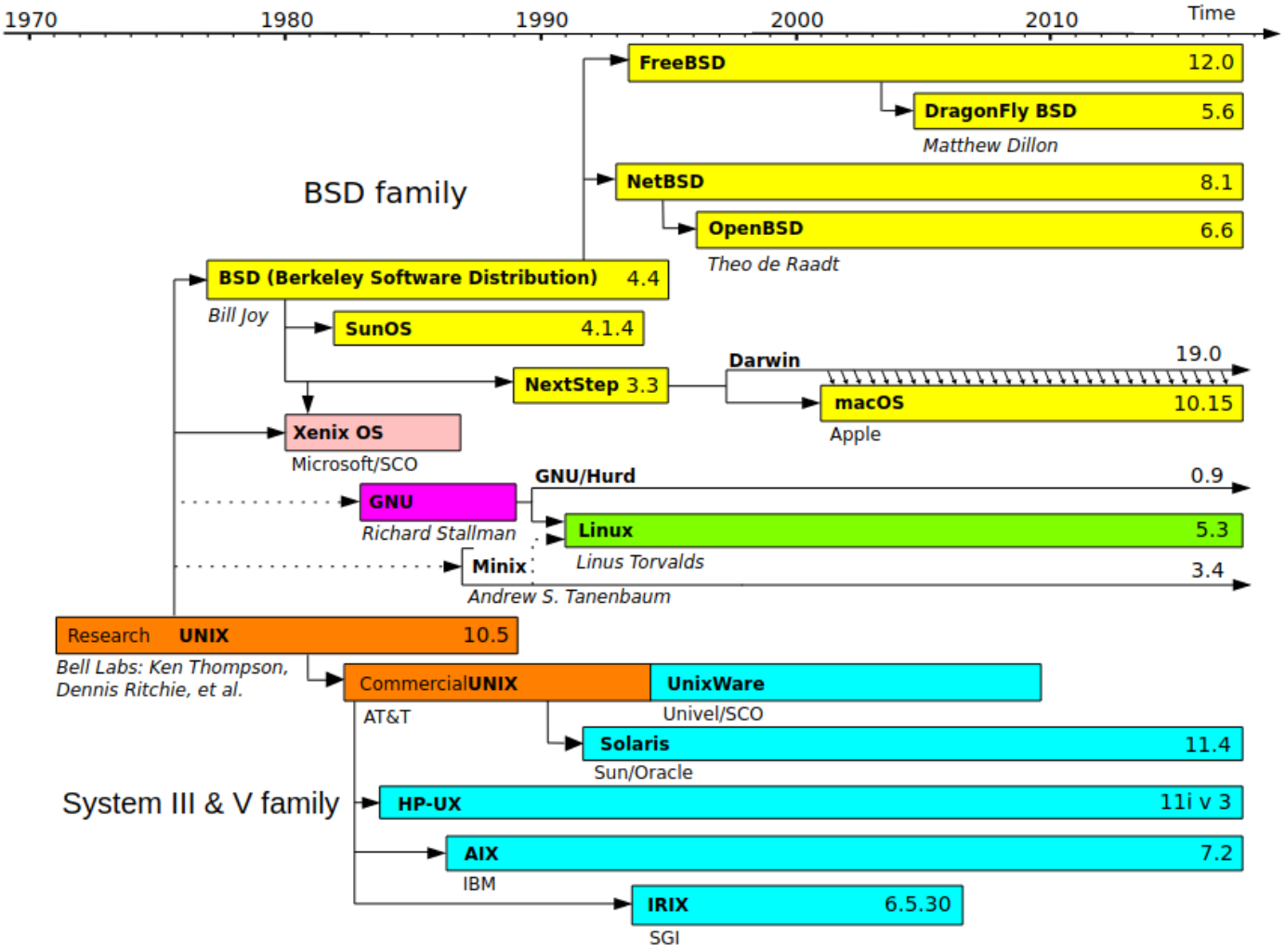
During the early days of computers, programmers and researchers were one and the same. While developing programming languages like C – the backbone of Unix – we were also exploring what computers could accomplish for the first time. To that end, it was common to share software and learn from each other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

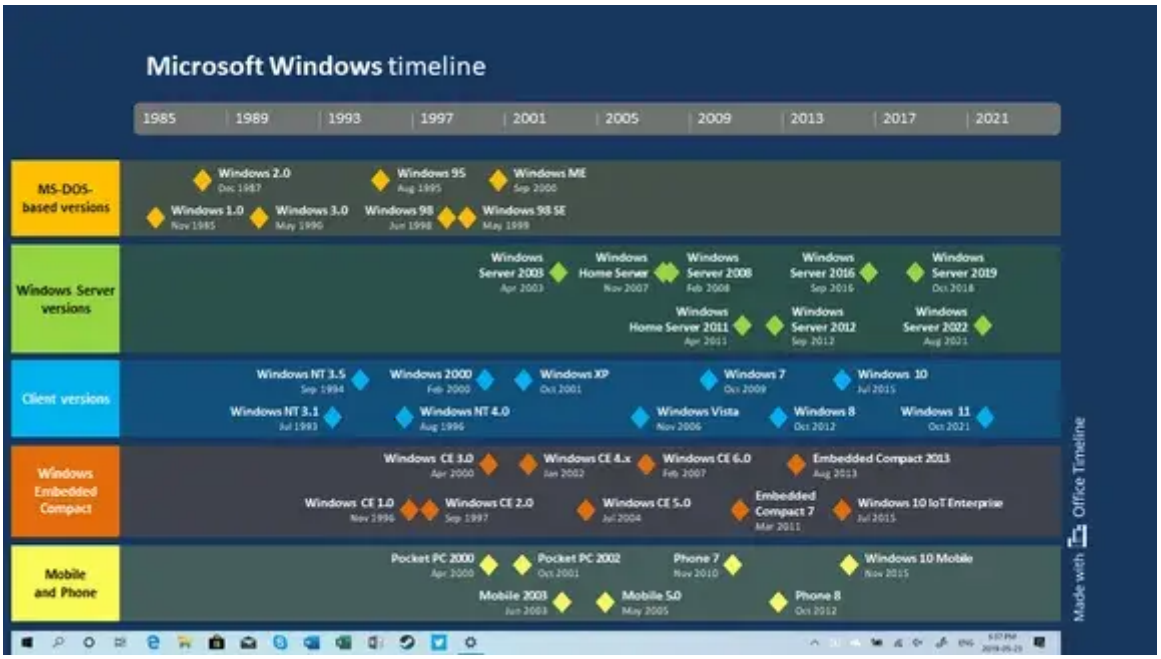
> Cathedral and the Bazaar was a foundational book by Eric S. Raymond about opposing software project management styles.

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. This data it is not human-readable and only works on 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 wasn't compiled to run efficiently on your specific computer system. Instead, it is compiled to meet '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

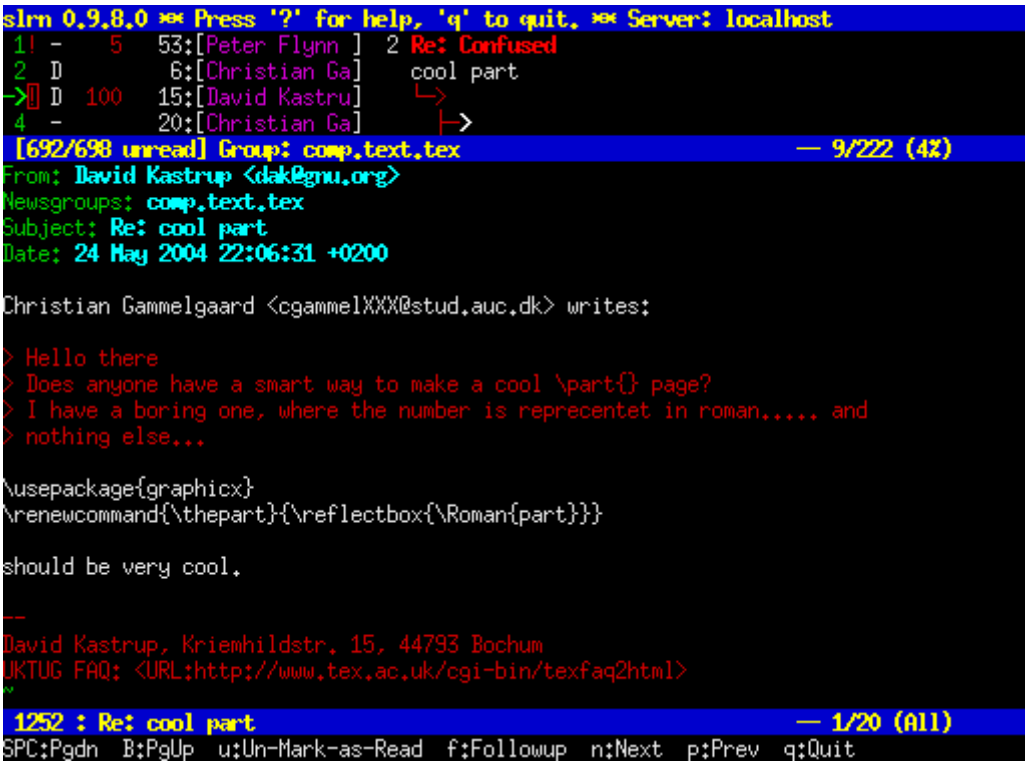
The freedom to redistribute copies so you can help your neighbor.

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 like Usenet – one of the first examples of a public-facing 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 necessary to build an operating system from the ground up. While developed through collaboration, these constituent projects are 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 a limited function with well-defined parameters. This facilitated 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 led to a rolling release model where individual software packages were updated frequently and released quickly. These modular programs could be maintained

independently and development teams only needed to coordinate how their modules would communicate – such as a mutually-defined API or the system call interface.

Software was specifically crafted to fill an empty technological niche and fulfill a desired goal. Communities formed to compare and contrast methods for reaching the same end. Developers held the autonomy to make their own decisions about how the project would progress.

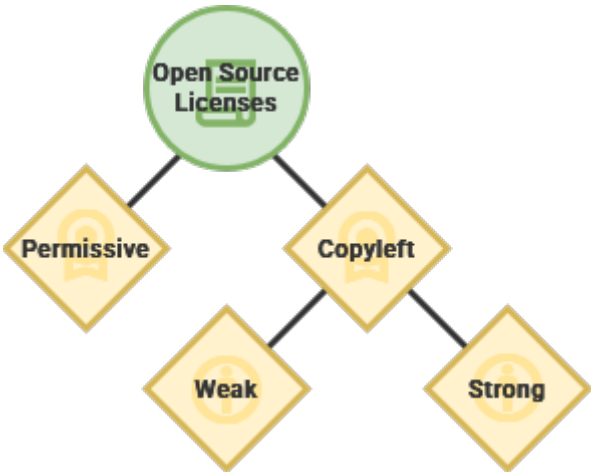
After using a high-level programming language to code a software program, it needs to be translated into machine code. This creates the executable program to can communicate with your underlying hardware through your operating system. This process happens in multiple stages – some well in advance, while others happen just-in-time.

((Diagram showing compilation happening in stages, including object code, machine code and just in time))

The Windows operating system revolves around compiled programs that are easy-to-use and operate. Similarly, Linux provides access to compiled software out of the box, but also provides the source code for anyone to compile software from the ground up for their specific hardware. In practice, this can bring new life into even the oldest computer systems.

Licensing

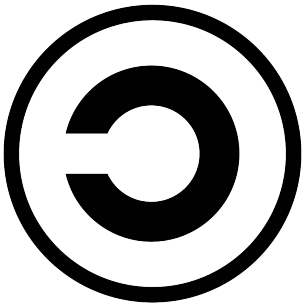
Copyright grants exclusive legal rights to the originator of a creative and intellectual work – like a book, movie or software code. Relatedly, copyleft is a philosophy that leverages the legal protections of copyright to prioritizing sharing and collaboration.



Starting in 1985, GNU Software was released under the GNU General Public License and enabled full use by anyone – with specific restrictions. This was the first copyleft license mandating that all derivative works maintain reciprocity.

Copyleft Licenses

There are a spectrum of requirements for fulfilling the developer's intended spirit of reciprocity – from maintaining a similar software license to simply attributing through a file.



While modern "weak" copyleft licenses allow embedding within proprietary projects, the original "strong" GPL license required that the derivative project retain the same license.

	Copyleft Licenses
License	**GPL** *1989* A strong copyleft license that comes with many conditions for usage within derivative software while providing express consent to use related patents.
License	**The Unlicense** *2010* This license foregoes intellectual copyright and attributes all work to the public domain. While not technically copyleft, this anti-copyright license is compatible with the GNU GPL.
License	**Mozilla Public License 2.0** *2012* This license balances the concerns of free software proponents and proprietary software developers.

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 to facilitate open collaboration.

They created their own permissive software license that placed barely any restrictions on how you could use the software. This also provided no warranty about the continues maintenance of the project and removed the developers from all liability. This even explicitly allowed proprietization – meaning it could be used within private, "closed-source" programs.

Permissive Licenses

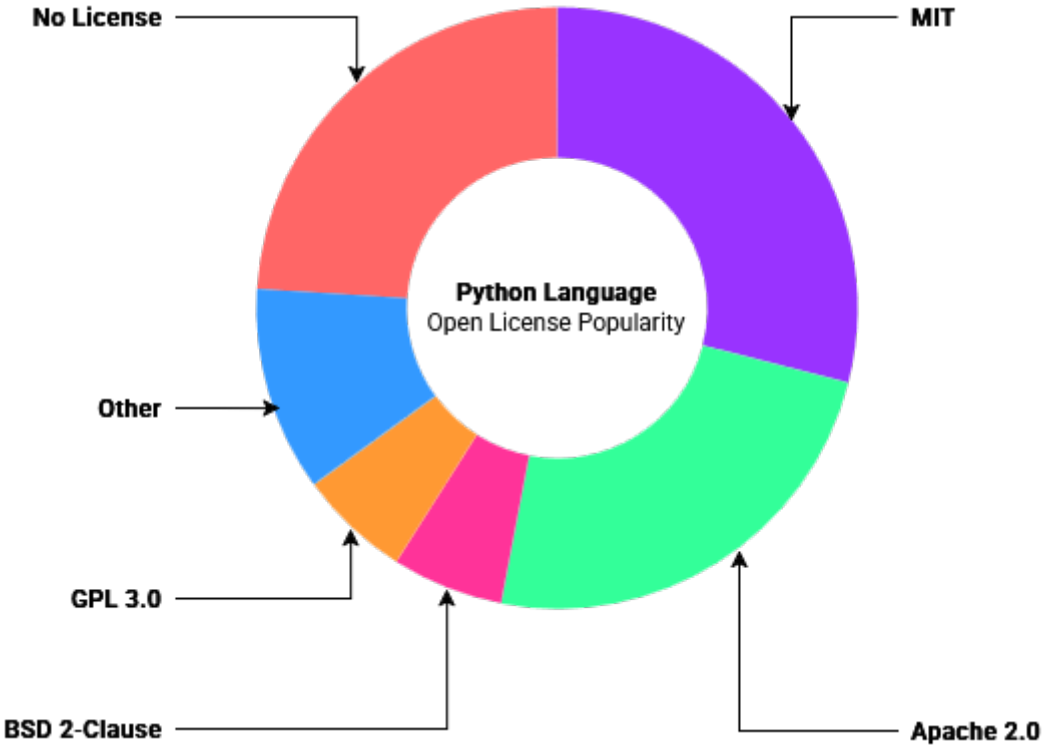
One of the primary differences between a copyleft license and a permissive license is the concept of "share-alike" – which relates to the Creative Commons.



Copyleft licenses require that information about the original work is available, often creating a web of interconnected projects. Permissive licenses, on the other hand, have few stipulations on how they can be used.

	Permissive Licenses
License	**Apache** *2004* A permissive license that allows that this software can be incorporated into larger projects that themselves are released under a different license.
License	**MIT** *1987* This straightforward license only requires that the licensing information is shown, otherwise the software can be used freely for any reason.

	Permissive Licenses
License	**BSD 4-Clause** *1990* The first in a family of permissive licenses with the original requiring acknowledgement in the advertisement of all derivative works.



Source

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. He released the software under the General Public License. This meant the software could be shared freely and modified by anyone.

GNU adopted the new Linux kernel as its own – which was now rapidly evolving into a global community. The resulting operating system is now known most commonly as Linux – even though

there is a movement to change this to GNU/Linux. Many Linux Distributions use the Linux Kernel by default.

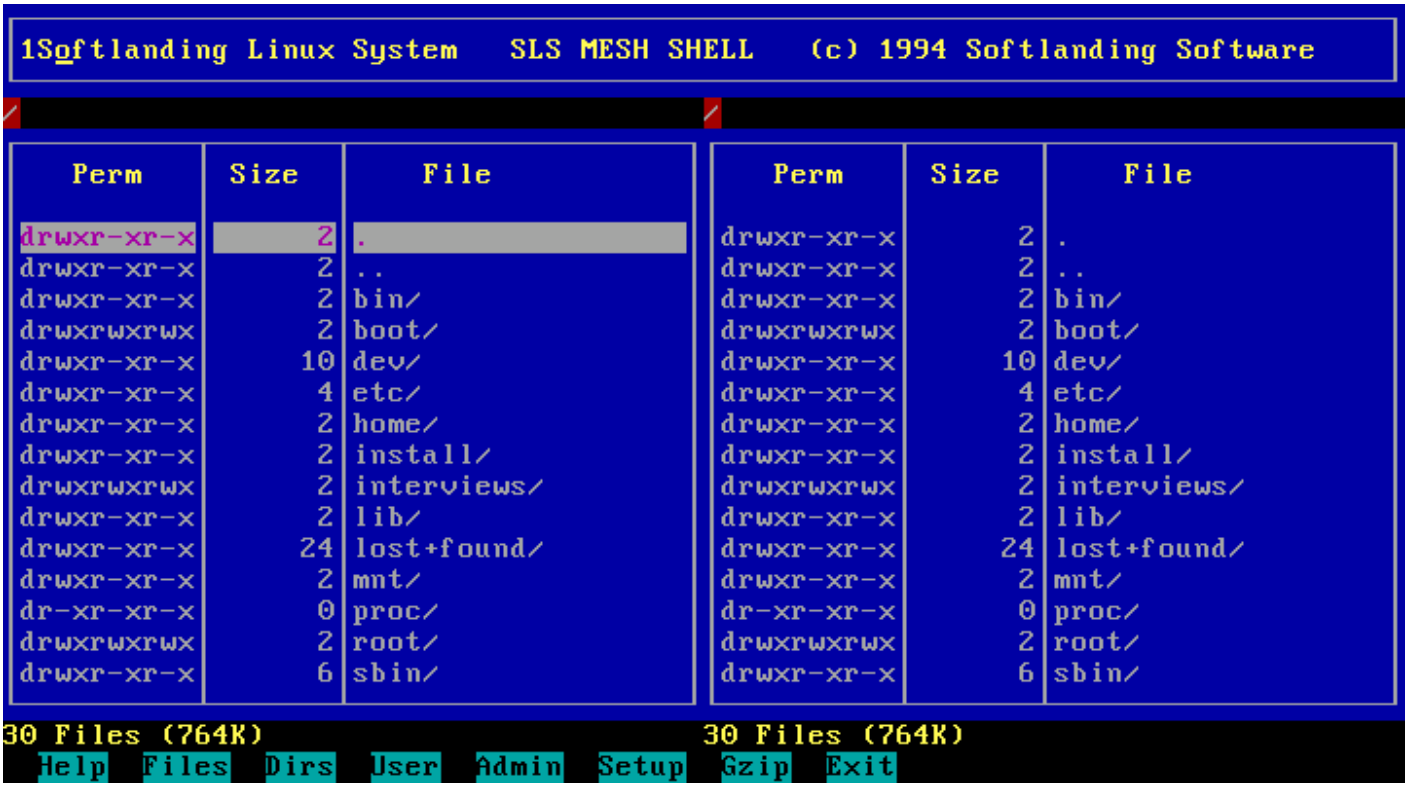
At a conference, a collective of software developers concluded that the Free Software Movement's social activism was not appealing to companies. The group – later known as the Open Source Initiative – felt that more emphasis was needed the business potential of open-source software. Linux was quickly adopted as the flag-ship project of the newly forming Open-Source Movement.

Compared to the earlier Free Software Movement, this new initiative took less of a focus on social, moral and ethical issues – like software freedom and social responsibility. Instead, the open-source movement chose to highlight the quality, flexibility and innovation that could be accomplished by sharing access to source code. In practice, this meant focusing on creating "something more amenable to commercial business use".

Social Contract

By creating mechanisms to decentralize software development and focus on discrete modules, Linux began to produce viable operating systems. The process was not always easy or direct because difficulty often arose while navigating the divide – both geographical and interpersonal. From the start, the computing experiments have produced notably strong personalities.

By 1992, over a million computers were connecting to the World Wide Web. The Linux project quickly ballooned to encompass many developers over the Internet. Combined with software from the GNU Project and other developers, complete Linux operating systems started taking shape.



By 1994, the World Wide Web had attracted [over 25 million subscribers](#) through services like [America Online](#). [SLS](#) – or Softlanding Linux System – was one of the first virally adopted distributions. Roughly patched together with a rudimentary interface, it was riddled with bugs and was notoriously difficult to use, but it gave a glimmer of the future.

Defining Simplicity

Slackware was created as a personal project to address the perceived problems with SLS, developed by [Patrick Volkerding](#). This new release was intended to streamline the experience while keeping every part true to the original developer's design.



He had no original intention to release Slackware, but it has become the oldest maintained distribution – used as the base for other operating systems. The continued development of Slackware rest fully on the project's reluctant [benevolent-dictator-for-life](#).

```
220-------------------------------------------------------------------------
220-                      R S Y N C . O S U O S L . O R G
220-                          Oregon State University
220-                              Open Source Lab
220-
220-       Unauthorized use is prohibited - violators will be prosecuted
220-------------------------------------------------------------------------
220-
220-                For more information about the OSL visit:
220-                    http://osuosl.org/services/hosting
220-
220-          This host is the home to the primary archives of several
220-           projects.  We would prefer that only primary/secondary
220-                    mirrors use this service.  Thanks!
220-
220-------------------------------------------------------------------------
220-
220
Name (ftp.slackware.com:klaatu): anonymous
Password (ftp.slackware.com:anonymous):
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> _
```

Slackware aimed to create the most "Unix-like" distribution and created a versatile "vanilla" Linux experience that could be completely customized from scratch. Volkerding sought to create the simplest experience – in referrence to how the system operated and not how it was used.

Slackware has a steep learning curve and the distro is by no means intended for people new to the Linux ecosystem. By default, minimal software is installed and only the command line interface is available. The operating system also makes no attempt to install the specific software required by programs, instead relying on oversight by an administrator.

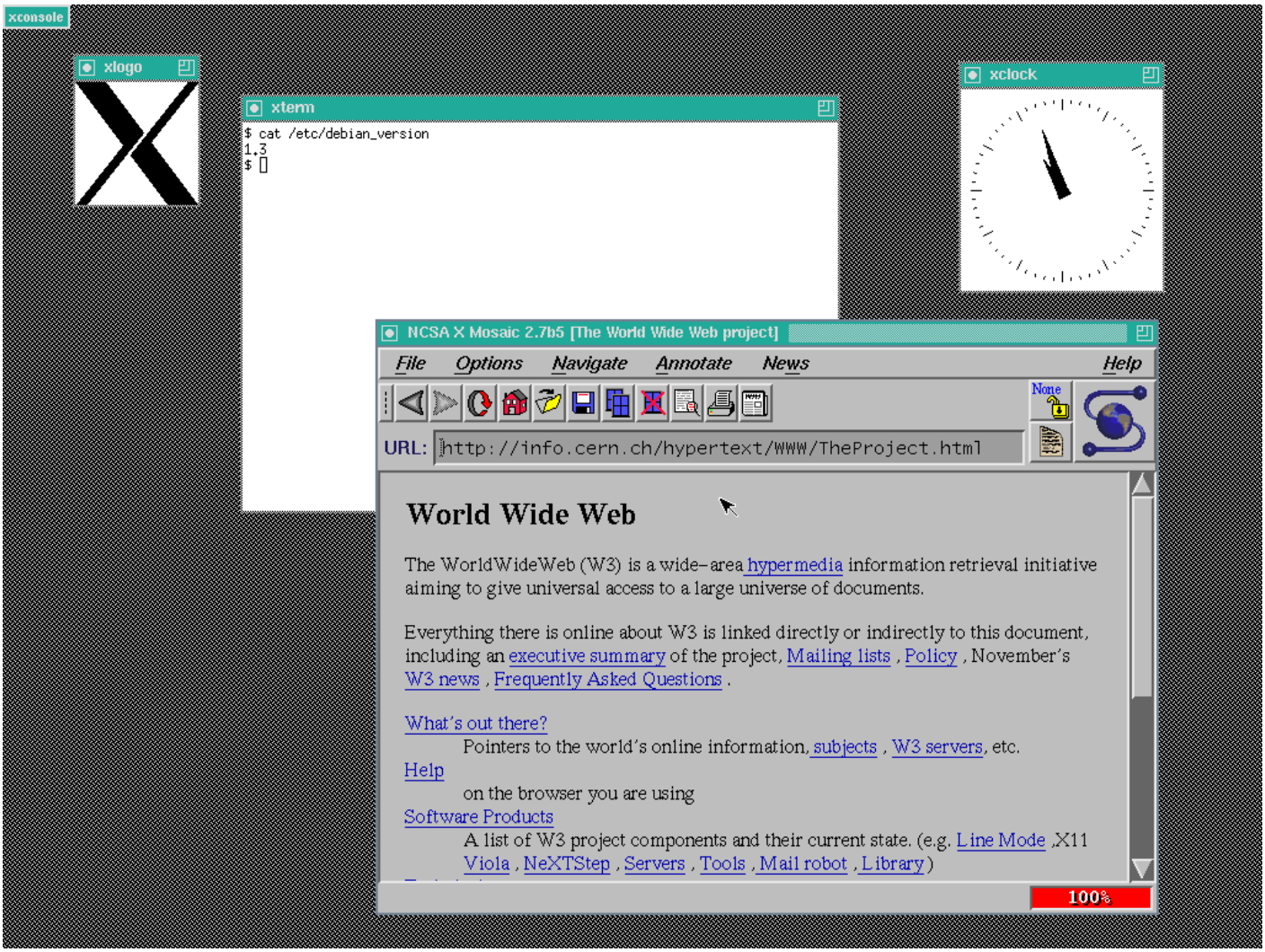
Universal Operating System

Released around the same time as Slackware, another Linux distribution emerged that sought to define simple a little differently. Their focus was to create a simple and accessible system that could function on the widest range of hardware.



Similarly frustrated with SLS, Ian Murdock created Debian as a conscious response to perceived flaws. By contrast, this project took an almost humanist approach that guaranteed a free operating system for everyone, forever.

> Debian codenames come from the *Toy Story* film universe.



Debian v1.3 "Bo"

Debian is a volunteer-operated software project and accompanying non-profit organization that operate entirely over the internet. Guided by an annually elected leader, their operation is governed by three foundational documents that outline the mission and vision for their software:

Counter_1

Social Contract

This document outlines core principles of the project, as well as the expectations and requirements of developers who create on its behalf.

Counter_2

Free Software Guidelines

This document defines "free software" and sets the requirement for software that can be used to build the operating system. While similar to the GNU Manifesto, these guidelines specifically lay out avenues for a relationship with commercial "non-free" software.

Counter_3

Constitution

This document explores formal power structures, lying out the responsibilities of the Debian project leader and other organizational roles.

Debian has been developed according to principles similar to the GNU Project. In fact, they even had a short-lived partnership. In contrast to the Free Software Movement, Debian proactively included important proprietary "non-free" software – such as firmware or drivers. Fundamentally, Debian wanted to remain open to commercial clients and the potential relationships that may follow.

At the same time, Debian was conscious of the corporate Linux distributions forming, like SUSE and Red Hat. Concerned with the potential effects that profit could have on the project's future, the Social Contract was written to ensure the operating system remained open and free.

> The Four Software Freedoms were not known to the Debian project at the time.

From their perspective, the modular nature of Linux allowed people to work together despite differences in project philosophy and organization. They defined open-source software licenses that supported the spirit of the project while explicitly opening the door to commercial "non-free" software. They placed their bet on the contributions of both corporations and communities to propel mutual growth.

Crowdsourcing Security

Security can be slippery defintion that encompasses so many different things. It manifests as a noun: in the moment, when we are safe from harm in our beds at the end of the day. It also takes shape as a verb: occurring over time through our proactive measures to ensure our safety. Security is both a feeling and a reality.

We contribute to our own feeling of safety by locking the door and enacting this simple layer of security. While the locked door may be technically easy to overcome, an ongoing relationship with our neighbors and communities lends weight to this assurance. Our society frowns on the theft of property and our laws create consequences for a trespasser's actions.

Security is a mindset and an ongoing relationship, not a task or a checklist that can be completed. Protective measures shouldn't occur in isolation, but within a holistic system that values and respects security overall. We build ourselves communities who impart onto us the feeling of safety.

Decentralized Defenses

Cybersecurity carries this vigilance into the digital systems we create to connect us. By using proprietary software, people must often tacitly accept the security provided by the developers. The open-source Linux operating system approaches security in a fundamentally different way than "closed-source" ones – like Windows and MacOS.

Proprietary software rely on the concept of security by obscurity – the idea that it is difficult to exploit what you cannot see. This has been proven untrue time and time again. Without the source code available, hackers can still exploit a system but the community cannot always respond.

By contrast, the code powering Linux is openly available on the internet for review. At first thought, this may seem insecure and ill-advised – to advertise potential exploits to would-be hackers. However, a system that relies solely on obscuring vulnerability is not truly secure.

Linus's Law – named after the creator of the Linux kernel – asserts that "given enough eyeballs, all bugs are shallow". The philosophy claims that the more independent people are working on a project, the greater the chances of uncovering bugs early. With regards to emergent "zero-day" exploits, there are numerous communities searching for the same fix.

While there have been no large-scale experiments or peer reviewed studies, empirical evidence can offer some clues. Popular open-source software had a higher ratio of big fixes than less popular projects by the same company. Relatedly, open-source GNU programs have demonstrated they're more reliable than their proprietary Unix counterparts.

We shouldn't allow ourselves to be lulled into false sense of security around open-source software. Simply making source code available to a broader community does not guarantee that vulnerabilities will immediately be found and fixed. While a large community of people may use the software,

there is no promise that they are engaging.

A potent illustration of this debate has been the Heartbleed security bug persisting in a popular project for two years. An accidental vulnerability within openssl – software that securely encrypts Internet connections – was maintained entirely by fifteen unpaid volunteer developers.

> “ In these cases, the eyeballs weren't really looking.
>
> — Linux Foundation Director, Jim Zemlin

Intentional Design

Linux employs a secure by design philosophy that develops a robust architecture from the ground up with conscious planning put into design at every layer. The Open Source Initiative works toward a vendor-neutral ecosystem welcoming contribution from all sectors.

Corporations and communities can lend their voices to the future of open-source software. This offers healthy competition, enabling people with choices about how to use their electronics. The Linux ecosystem has built-in support from a diversity of distributions – such as the community-supported Debian, the corporate-focused Fedora, or a hybrid-approach like Ubuntu.

Crowdsourcing volunteers from open communities requires developers to consider how their software might be exploited from the start. By presenting source code, open communities can engage with software review – which had been repeatedly shown to address security issues.

Despite every precaution, exploits are bound to arise within a software program – especially as the code base grows larger, including code written and maintained by other developers. By the security company Synopsys' estimates, over 80% of codebases have at least one known vulnerability – on average, a codebase may have upwards of 150 vulnerailities with documented solutions available for at least two years.



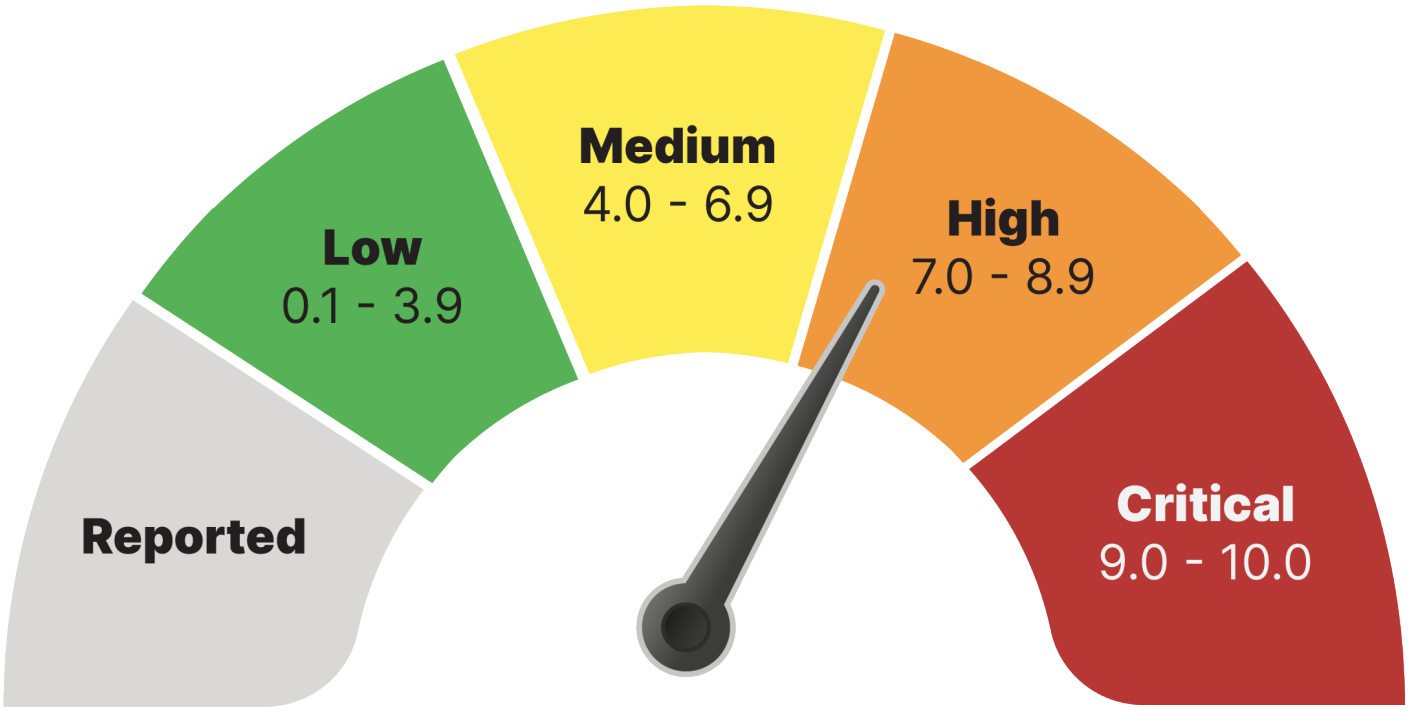
The Open-Source Security Foundation (known as OpenSSF) is dedicated to offering security testing. Through Security Score Cards, developers can quickly test their software for vulnerabilities. By extension, they can advertise their rating to help build community trust with the project.

Coordinated Response

Debian has created systems for crowdsourcing and responding to emergent threats. They employ a public bug tracking service that enables anyone to report inconsistencies with the operating system. Developers, researchers and white hat hackers can identify eminent threats, separating them from relatively minor programming errors.

Bug_report	**Bug** This is a design or coding defect within a program that can lead to undesired effects – such as a crash.
Skull	**Vulnerability** This is a weakness in a system – sometimes the result of a bug or design flaw – that can be leveraged to compromise the security of a system.
Crisis_alert	**Exploit** This is a procedure or program that take advantage of vulnerabilities within a system and allows a malicious actor to use it to their advantage.

Once threats are identified, they are assessed for severity based on technical factors – including ease-of-use and potential impact. This rating, known as the Common Vulnerability Scoring System, offers a simple 1-to-10 metric for exploit severity. Once assessed, an advisory is promptly made public with information about the exploit and any enactable defenses until the exploit is formally fixed.



While an entirely volunteer-operated organization, Debian hosts a security team who proactively ensure the security of the operating system while working with development teams to resolve critical vulnerabilities. They maintain the documentation for further hardening Debian to make it even more difficult to exploit while limiting collateral.

Debian is now one of the most popular Linux distributions and many others have been created from it. As of 2025, there are almost 140 Linux-based operating systems that rely on Debian. It is leveraged almost everywhere – by governments, schools, corporations, non-profit organizations and even laptops in space aboard the ISS.

Revision #30
Created 2 June 2025 18:36:10 by metaphorraccoon
Updated 7 June 2025 20:19:27 by metaphorraccoon