Software versioning

Software upgrade versioning is the process of assigning either unique *version names* or unique *version numbers*to unique states of computer software. Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software. At a fine-grained level, revision control is often used for keeping track of incrementally different versions of information, whether or not this information is computer software.

Modern computer software is often tracked using two different software versioning schemes—internal version number that may be incremented many times in a single day, such as a revision control number, and a *release version* that typically changes far less often, such as semantic versioning[1] or a project code name.

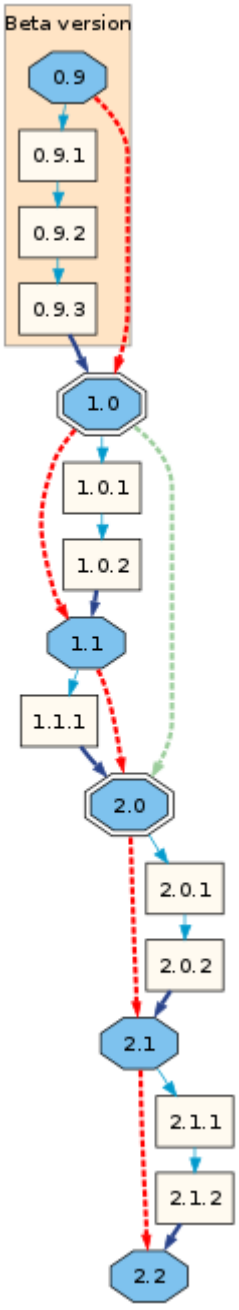
Contents

External links

Schemes

A variety of version numbering schemes have been created to keep track of different versions of a piece of software. The ubiquity of computers has also led to these schemes being used in contexts outside computing.

Sequence-based identifiers



In sequence-based software versioning schemes, each software release is assigned a unique identifier that consists of one or more sequences of numbers or letters. This is the extent of the commonality; schemes vary widely in areas such as the quantity of sequences, the attribution of meaning to individual sequences, and the means of incrementing the sequences.

Change significance

In some schemes, sequence-based identifiers are used to convey the significance of changes between releases. Changes are classified by significance level, and the decision of which sequence to change between releases is based on the significance of the changes from the previous release, whereby the first sequence is changed for the most significant changes, and changes to sequences after the first represent changes of decreasing significance.

Depending on the scheme, significance may be assessed by lines of code changed, function points added or removed, potential impact on customers in terms of work required to adopt a newer version, risk of bugs or undeclared breaking changes, degree of changes in visual layout, quantity of new features, or almost anything the product developers or marketers deem to be significant, including marketing desire to stress the "relative goodness" of the new version.

Semantic versioning (aka SemVer),[1] is a widely adopted version scheme[2] that uses a sequence of three digits (Major.Minor.Patch), an optional pre-release tag and optional build meta tag. In this scheme, risk and functionality are the measures of significance. Breaking changes are indicated by increasing the major number (high risk), new non-breaking features increment the minor number (medium risk) and all other non-breaking changes increment the patch number (lowest risk). The presence of a pre-release tag (-alpha, -beta) indicates substantial risk, as does a major number of zero (0.y.z), which is used to indicate a work-in-progress that may contain any level of potentially breaking changes (highest risk).

Developers may choose to jump multiple minor versions at a time to indicate significant features have been added, but are not enough to warrant incrementing a major version number; for example Internet Explorer 5 from 5.1 to 5.5, or Adobe Photoshop 5 to 5.5. This may be done to emphasize the value of the upgrade to the software user, or, as in Adobe's case, to represent a release halfway between major versions (although levels of sequence based versioning are not limited to a single digit, as in Blender version 2.91 or Minecraft Java Edition after 1.10).

A different approach is to use the *major* and *minor* numbers, along with an alphanumeric string denoting the release type, e.g. "alpha" (a), "beta" (b), or "release candidate" (rc). A software release train using this approach might look like 0.5, 0.6, 0.7, 0.8, 0.9 → 1.0b1, 1.0b2 (with some fixes), 1.0b3 (with more fixes) → 1.0rc1

(which, if it is stable *enough*), 1.0rc2 (if more bugs are found) → 1.0. It is a common practice in this scheme to lock-out new features and breaking changes during the release candidate phases and for some teams, even betas are lock-down to bug fixes only, in order to ensure convergence on the target release.

Other schemes impart meaning on individual sequences:

> *major.minor[.build[.revision]]* (example: *1.2.12.102*)
> *major.minor[.maintenance[.build]]* (example: *1.4.3.5249*)

Again, in these examples, the definition of what constitutes a "major" as opposed to a "minor" change is entirely subjective and up to the author, as is what defines a "build", or how a "revision" differs from a "minor" change.

Shared libraries in Solaris and Linux may use the *current.revision.age* format where:[3][4]

> *current*: The most recent interface number that the library implements.
> *revision*: The implementation number of the current interface.
> *age*: The difference between the newest and oldest interfaces that the library implements. This use of the third field is specific to libtool: others may use a different meaning or simply ignore it.

A similar problem of relative change significance and versioning nomenclature exists in book publishing, where edition numbers or names can be chosen based on varying criteria.

In most proprietary software, the first released version of a software product has version 1.

Degree of compatibility

Some projects use the major version number to indicate incompatible releases. Two examples are Apache Portable Runtime (APR)[5] and the FarCry CMS.[6]

Semantic versioning three-part version number

Semantic versioning[1] is a formal convention for specifying compatibility using a three-part version number: major version; minor version; and patch. The patch number is incremented for minor changes and bug fixes which do not change the software's application programming interface (API). The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

Often programmers write new software to be backward compatible, i.e., the new software is designed to interact correctly with older versions of the software (using old protocols and file formats) and the most recent version (using the latest protocols and file formats). For example, IBM z/OS is designed to work properly with 3 consecutive major versions of the operating system running in the same sysplex. This enables people who run a high availability computer cluster to keep most of the computers up and running while one machine at a time is shut down, upgraded, and restored to service.[7]

Often packet headers and file format include a version number – sometimes the same as the version number of the software that wrote it; other times a "protocol version number" independent of the software version number. The code to handle old deprecated protocols and file formats is often seen as cruft.

Designating development stage

Software in the experimental stage (alpha or beta) often use a zero in the first ("major") position of the sequence to designate its status. However, this scheme is only useful for the early stages, not for upcoming releases with established software where the version number has already progressed past 0.[1]

A number of schemes are used to denote the status of a newer release:

- *Alphanumeric suffix* is a common scheme adopted by semantic versioning.[1] In this scheme, versions are affixed a dash plus some alphanumeric characters to indicate the status.
- *Numeric status* is a scheme that uses numbers to indicate the status as if it's part of the sequence. A typical choice is the third position for the four-position versioning.
- *Numeric 90+* is another scheme that uses numbers, but apparently under a number of a previous version. A large number in the last position, typically 90 or higher, is used. This is commonly used by older open source projects like GNOME and Fontconfig.

Comparison of development stage indicators

Stage	Semver	Num. Status	Num 90+
Alpha	1.2.0-a.1	1.2.0.1	1.1.90
Beta	1.2.0-b.2	1.2.1.2	1.1.93
Release candidate	1.2.0-rc.3	1.2.2.3	1.1.97
Release	1.2.0	1.2.3.0	1.2.0
Post-release fixes	1.2.5	1.2.3.5	1.2.5

The two purely numeric forms removes the special logic required to handle the comparison of "alpha < beta < rc < no prefix" as found in semantic versioning, at the cost of clarity. (semantic versioning actually does not specify specific terms for development stages; the comparison is simply in lexicographical order.)

Incrementing sequences

There are two schools of thought regarding how numeric version numbers are incremented. Most free and open-source software packages, including MediaWiki, treat versions as a series of individual numbers, separated by periods, with a progression such as 1.7.0, 1.8.0, 1.8.1, 1.9.0, 1.10.0, 1.11.0, 1.11.1, 1.11.2, and so on.

On the other hand, some software packages identify releases by decimal numbers: 1.7, 1.8, 1.81, 1.82, 1.9, etc. Decimal versions were common in the 1980s, for example with NetWare, DOS, and Microsoft Windows, but even in the 2000s have been for example used by Opera[8] and Movable Type.[9] In the decimal scheme, 1.81 is the minor version following 1.8, while maintenance releases (i.e. bug fixes only) may be denoted with an alphabetic suffix, such as 1.81a or 1.81b.

The standard GNU version numbering scheme is major.minor.revision,[10] but Emacs is a notable example using another scheme where the major number (1) was dropped and a *user site* revision was added which is always zero in original Emacs packages but increased by distributors.[11] Similarly, Debian package numbers are prefixed with an optional "epoch", which is used to allow the versioning scheme to be changed.[12]

Resetting

In some cases, developers may decide to reset the major version number. This is sometimes used to denote a new development phase being released. For example, *Minecraft* Alpha ran from version 1.0.0 to 1.2.6, and when Beta was released, it reset the major version number, and ran from 1.0 to 1.8. Once the game was fully released, the major version number again reset to 1.0.0.[13]

Separating sequences

When printed, the sequences may be separated with characters. The choice of characters and their usage varies by scheme. The following list shows hypothetical examples of separation schemes for the same release (the thirteenth third-level revision to the fourth second-level revision to the second first-level revision):

- A scheme may use the same character between all sequences: 2.4.13, 2/4/13, 2-4-13
- A scheme choice of which sequences to separate may be inconsistent, separating some sequences but not others: 2.413
- A scheme's choice of characters may be inconsistent within the same identifier: 2.4_13

When a period is used to separate sequences, it *may* or *may not* represent a decimal point, — see "Incrementing sequences" section for various interpretation styles.

Number of sequences

There is sometimes a fourth, unpublished number which denotes the software build (as used by Microsoft). Adobe Flash is a notable case where a four-part version number is indicated publicly, as in 10.1.53.64. Some companies also include the build date. Version numbers may also include letters and other characters, such as Lotus 1-2-3Release 1a.

Using negative number

Some projects use negative version numbers. One example is the SmartEiffel compiler which started from -1.0 and counted upwards to 0.0.[11]

Date of release

Street Fighter EX splash screenshowing release number in CalVerformat

Many projects use a date-based versioning scheme called **Calendar Versioning** (aka **CalVer**[14]).

Ubuntu Linux is one example of a project using calendar versioning; Ubuntu 18.04, for example, was released April 2018. This has the advantage of being easily relatable to development schedules and support timelines. Some video games also use date as versioning, for example the arcade game Street Fighter EX. At startup it displays the version number as a date plus a region code, for example *961219 ASIA*.

When using dates in versioning, for instance, file names, it is common to use the ISO 8601 scheme:[15] YYYY-MM-DD, as this is easily string sorted to increasing or decreasing order. The hyphens are sometimes omitted. The Wine project formerly used a date versioning scheme, which used the year followed by the month followed by the day of the release; for example, "Wine 20040505".

Microsoft Office build numbers are an encoded date:[16] the first two digits indicate the number of months that have passed from the January of the year in which the project started (with each major Office release being a different project), while the last two digits indicate the day of that month. So 3419 is the 19th day of the 34th month after the month of January of the year the project started.

Other examples that identify versions by year include Adobe Illustrator 88 and WordPerfect Office 2003. When a year is used to denote version, it is generally for marketing purposes, and an actual version number also exists. For example, Microsoft Windows 95 is internally versioned as MS-DOS 7.00 and Windows 4.00;

likewise, Microsoft Windows 2000 Server is internally versioned as Windows NT 5.0 ("NT" being a reference to the original product name).

Python

The Python Software Foundation has published PEP 440 -- Version Identification and Dependency Specification,[17] outlining their own flexible scheme, that defines an epoch segment, a release segment, pre-release and post-release segments and a development release segment.

TeX

TeX has an idiosyncratic version numbering system. Since version 3, updates have been indicated by adding an extra digit at the end, so that the version number asymptotically approaches π; this is a form of unary numbering – the version number is the number of digits. The current version is 3.14159265. This is a reflection of TeX being very stable, and only minor updates are anticipated. TeX developer Donald Knuth has stated that the *"absolutely final change (to be made after [his] death)"* will be to change the version number to π, at which point all remaining bugs will become permanent features.[18]

In a similar way, the version number of METAFONT asymptotically approaches e.

Apple

During the era of the classic Mac OS, minor version numbers rarely went beyond ".1". When they did, they usually jumped straight to ".5", suggesting the release was "more significant".[a] Thus, "8.5" was marketed as its own release, representing "Mac OS 8 and a half", and 8.6 effectively meant "8.5.1".

Mac OS X departed from this trend, in large part because "X" (the Roman numeral for 10) was in the name of the product. As a result, all versions of OS X began with the number 10. The first major release of OS X was given the version number 10.0, but the next major release was not 11.0. Instead, it was numbered 10.1, followed by 10.2, 10.3, and so on for each subsequent major release. Thus the 11th major version of OS X was labeled "10.10". Even though the "X" was dropped from the name as of macOS 10.12, this numbering scheme continued through macOS 10.15. Under the "X"-based versioning scheme, the third number (instead of the second) denoted a minor release, and additional updates below this level, as well as updates to a given major version of OS X coming after the release of a new major version, were titled Supplemental Updates.[19]

The Roman numeral X was concurrently leveraged for marketing purposes across multiple product lines. Both QuickTime and Final Cut Pro jumped from version 7 directly to version 10, QuickTime X and Final Cut Pro X. Like Mac OS X itself, the products were not upgrades to previous versions, but brand-new programs. As with OS X, major releases for these programs incremented the second digit and minor releases were denoted using a third digit. The "X" was dropped from Final Cut's name with the release of macOS 11.0 (see below), and QuickTime's branding became moot when the framework was deprecated in favor of AVFoundation in 2011 (the program for playing QuickTime video was only named QuickTime Player from the start).

Apple's next macOS release, provisionally numbered 10.16,[20] was officially announced as macOS 11.0 at WWDC in June 2020.[21]

Microsoft Windows

The Microsoft Windows operating system was first labelled with standard version numbers for Windows 1.0through Windows 3.11. After this Microsoft excluded the version number from the product name. For Windows 95(version 4.0), Windows 98 (4.10) and Windows 2000 (5.0), year of the release was included in the product title. After Windows 2000, Microsoft created the Windows Server family which continued the year-based style with a difference: For minor releases, Microsoft suffixed "R2" to the title, e.g., Windows Server 2008 R2 (version 6.1). This style had remained consistent to this date. The client versions of Windows however did not adopt a consistent style. First, they received names with arbitrary alphanumeric suffixes as with Windows ME (4.90), Windows XP(5.1) and Windows Vista (6.0). Then, once again Microsoft adopted incremental numbers in the title, but this time, they were not version numbers; the version numbers of Windows 7, Windows 8 and Windows 8.1 are respectively 6.1, 6.2 and 6.3. In Windows 10, the version number leaped to 10.0[22] and subsequent updates to the OS only incremented build number and update build revision (UBR) number.

Other schemes

Some software producers use different schemes to denote releases of their software. The Debian project uses a major/minor versioning scheme for releases of its operating system, but uses code names from the movie *Toy Story*during development to refer to stable, unstable and testing releases.[23]

BLAG Linux and GNU features very large version numbers: major releases have numbers such as 50000 and 60000, while minor releases increase the number by 1 (e.g. 50001, 50002). Alpha and beta releases are given decimal version numbers slightly less than the major release number, such as 19999.00071 for alpha 1 of version 20000, and 29999.50000 for beta 2 of version 30000. Starting at 9001 in 2003, the most recent version as of 2011is 140000.[24][25][26]

Internal version numbers

Software may have an "internal" version number which differs from the version number shown in the product name (and which typically follows version numbering rules more consistently). Java SE 5.0, for example, has the internal version number of 1.5.0, and versions of Windows from NT 4 on have continued the standard numerical versions internally: Windows 2000 is NT 5.0, XP is Windows NT 5.1, Windows Server 2003 and Windows XP Professional x64 Edition are NT 5.2, Windows Server 2008 and Vista are NT 6.0, Windows Server 2008 R2 and Windows 7 are NT 6.1, Windows Server 2012 and Windows 8 are NT 6.2, and Windows Server 2012 R2 and Windows 8.1 are NT 6.3, however the first version of Windows 10 was 10.0 (10.0.10240). Note, however, that Windows NT is only on its fifth major revision, as its first release was numbered 3.1 (to match the then-current Windows release number) and the Windows 10 launching made a version leap from 6.3 to 10.0.

Pre-release versions

In conjunction with the various versioning schemes listed above, a system for denoting pre-release versions is generally used, as the program makes its way through the stages of the software release life cycle.

Programs that are in an early stage are often called "alpha" software, after the first letter in the Greek alphabet. After they mature but are not yet ready for release, they may be called "beta" software, after the second letter in the Greek alphabet. Generally alpha software is tested by developers only, while beta software is distributed for community testing.

Some systems use numerical versions less than 1 (such as 0.9), to suggest their approach toward a final "1.0" release. This is a common convention in open source software.[27][28] However, if the pre-release version is for an existing software package (e.g. version 2.5), then an "a" or "alpha" may be appended to the version number. So the alpha version of the 2.5 release might be identified as 2.5a or 2.5.a.

An alternative is to refer to pre-release versions as "release candidates", so that software packages which are soon to be released as a particular version may carry that version tag followed by "rc-#", indicating the number of the release candidate; when the final version is released, the "rc" tag is removed.

Release train

A **software release train** is a form of software release schedule in which a number of distinct series of versioned software releases for multiple products are released as a number of different "trains" on a regular schedule. Generally, for each product line, a number of different release trains are running at a given time, with each train moving from initial release to eventual maturity and retirement on a planned schedule. Users may experiment with a newer release train before adopting it for production, allowing them to experiment with newer, "raw", releases early, while continuing to follow the previous train's point releases for their production systems prior to moving to the new release train as it becomes mature.

Cisco's IOS software platform used a release train schedule with many distinct trains for many years. More recently, a number of other platforms including Firefox and Fenix for Android,[29] Eclipse,[30] LibreOffice,[31]Ubuntu,[32] Fedora,[33] Python,[34] digiKam[35] and VMware[36] have adopted the release train model.

Modifications to the numeric system

Odd-numbered versions for development releases

Between the 1.0 and the 2.6.x series, the Linux kernel used odd minor version numbers to denote development releases and even minor version numbers to denote stable releases; see Linux kernel § Version numbering. For example, Linux 2.3 was a development family of the second major design of the Linux kernel, and Linux 2.4 was the stable release family that Linux 2.3 matured into. After the minor version number in the Linux kernel is the release number, in ascending order; for example, Linux 2.4.0 → Linux 2.4.22. Since the 2004 release of the 2.6 kernel, Linux no longer uses this system, and has a much shorter release cycle.

The same odd-even system is used by some other software with long release cycles, such as Node.js up to version 0.12 as well as GNOME and WineHQ.[37]

Political and cultural significance of version numbers

Version 1.0 as a milestone

The free-software and open source communities tend to release software early and often. Initial versions are numbers less than 1, with these 0.x version used to convey that the software is incomplete and not reliable enough for general release or usable in its current state. Version 1.0 is used as a major milestone, indicating that

the software has at least all major features plus functions the developers wanted to get into that version, and is considered reliable enough for general release.[27][28] A good example of this is the Linux kernel, which was first released as version 0.01 in 1991,[38] and took until 1994 to reach version 1.0.0.[39]

The developers of the arcade game emulator MAME do not ever intend to release a version 1.0 of the program because there will always be more arcade games to emulate and thus the project can never be truly completed. Accordingly, version 0.99 was followed by version 0.100.[40]

Since the internet has become widespread, most commercial software vendors no longer follow the maxim that a major version should be "complete" and instead rely on patches with bugfixes to sort out the known issues which a solution has been found for and could be fixed.

Version numbers as marketing

A relatively common practice is to make major jumps in version numbers for marketing reasons. Sometimes software vendors just bypass the 1.0 release or quickly release a release with a subsequent version number because 1.0 software is considered by many customers too immature to trust with production deployments. For example, as in the case of dBase II, a product is launched with a version number that implies that it is more mature than it is.

Other times version numbers are increased to match those of competitors. This can be seen in many examples of product version numbering by Microsoft, America Online, Sun Solaris, Java Virtual Machine, SCO Unix, WordPerfect. Microsoft Access jumped from version 2.0 to version 7.0, to match the version number of Microsoft Word.

Microsoft has also been the target of 'catch-up' versioning, with the Netscape browsers skipping version 5 to 6, in line with Microsoft's Internet Explorer, but also because the Mozilla application suite inherited version 5 in its user agent string during pre-1.0 development and Netscape 6.x was built upon Mozilla's code base.

Another example of keeping up with competitors is when Slackware Linux jumped from version 4 to version 7 in 1999.[41]

Dropping the most significant element

Sun's Java has at times had a hybrid system, where the internal version number has always been 1.*x* but has been marketed by reference only to the *x*:

- JDK 1.0.3
- JDK 1.1.2 through 1.1.8
- J2SE 1.2.0 ("Java 2") through 1.4.2
- Java 1.5.0, 1.6.0, 1.7.0, 1.8.0 ("Java 5, 6, 7, 8")

Sun also dropped the first digit for Solaris, where Solaris 2.8 (or 2.9) is referred to as Solaris 8 (or 9) in marketing materials.

A similar jump took place with the Asterisk open-source PBX construction kit in the early 2010s, whose project leads announced that the current version 1.8.x would soon be followed by version 10.[42]

This approach, panned by many because it breaks the semantic significance of the sections of the version number, has been adopted by an increasing number of vendors including Mozilla (for Firefox).

Superstition

- The Office 2007 release of Microsoft Office had an internal version number of 12. The next version, Office 2010, has an internal version of 14, due to superstitions surrounding the number 13.[43] Visual Studio 2013 is Version number 12.0 of the product, and the new version, Visual Studio 2015 has the Version number 14.0 for the same reasons.[44]
- Roxio Toast went from version 12 to version 14, likely in an effort to skip the superstitions surrounding the number 13.
- Corel's WordPerfect Office, version 13 is marketed as "X3" (Roman number 10 and "3"). The procedure has continued into the next version, X4. The same has happened with Corel's Graphic Suite (i.e. CorelDRAW, Corel Photo-Paint) as well as its video editing software "Video Studio".
- Sybase skipped major versions 13 *and* 14 in its Adaptive Server Enterprise relational database product, moving from 12.5 to 15.0.
- ABBYY Lingvo Dictionary uses numbering 12, x3 (14), x5 (15).
- SUSE Linux Enterprise skipped version 13 and 14 after version 12 and directly released SLES 15 in July 2018.

Geek culture

- The SUSE Linux distribution started at version 4.2, to reference 42, "the answer to the ultimate question of life, the universe and everything" mentioned in Douglas Adams' *The Hitchhiker's Guide to the Galaxy*.
- A Slackware Linux distribution was versioned 13.37, referencing leet.
- Finnix skipped from version 93.0 to 100, partly to fulfill the assertion, "There Will Be No Finnix '95", a reference to Windows 95.[45]
- The Tagged Image File Format specification has used 42 as internal version number since its inception, its designers not expecting to alter it anymore during their (or its) lifetime since it would conflict with its development directives.

Overcoming perceived marketing difficulties

In the mid-1990s, the rapidly growing CMMS, Maximo, moved from Maximo Series 3 directly to Series 5, skipping Series 4 due to that number's perceived marketing difficulties in the Chinese market, where the number 4 is associated with "death" (see tetraphobia). This did not, however, stop Maximo Series 5 version 4.0 being released. (The "Series" versioning has since been dropped, effectively resetting version numbers after Series 5 version 1.0's release.)

Significance in software engineering

Version numbers are used in practical terms by the consumer, or client, to identify or compare their copy of the software product against another copy, such as the newest version released by the developer. For the programmer or company, versioning is often used on a revision-by-revision basis, where individual parts of the software are compared and contrasted with newer or older revisions of those same parts, often in a collaborative version control system.

In the 21st century, more programmers started to use a formalized version policy, such as the semantic versioning policy.[1] The purpose of such policies is to make it easier for other programmers to know when code changes are likely to break things they have written. Such policies are especially important for software

libraries and frameworks, but may also be very useful to follow for command-line applications (which may be called from other applications) and indeed any other applications (which may be scripted and/or extended by third parties).

Versioning is also a required practice to enable many schemes of patching and upgrading software, especially to automatically decide what and where to upgrade to.

Significance in technical support

Version numbers allow people providing support to ascertain *exactly* which code a user is running, so that they can rule out bugs that have already been fixed as a cause of an issue, and the like. This is especially important when a program has a substantial user community, especially when that community is large enough that the people providing technical support are *not* the people who wrote the code. The semantic meaning[1] of version.revision.change style numbering is also important to information technology staff, who often use it to determine how much attention and research they need to pay to a new release before deploying it in their facility. As a rule of thumb, the bigger the changes, the larger the chances that something might break (although examining the Changelog, if any, may reveal only superficial or irrelevant changes). This is one reason for some of the distaste expressed in the "drop the major release" approach taken by Asterisk et alia: now, staff must (or at least should) do a full regression test for every update.

Version numbers for files and documents

Some computer file systems, such as the OpenVMS Filesystem, also keep versions for files.

Versioning amongst documents is relatively similar to the routine used with computers and software engineering, where with each small change in the structure, contents, or conditions, the version number is incremented by 1, or a smaller or larger value, again depending on the personal preference of the author and the size or importance of changes made.

Version number ordering systems

Version numbers very quickly evolve from simple integers (1, 2, ...) to rational numbers (2.08, 2.09, 2.10) and then to non-numeric "numbers" such as 4:3.4.3-2. These complex version numbers are therefore better treated as character strings. Operating systems that include package management facilities (such as all non-trivial Linux or BSD distributions) will use a distribution-specific algorithm for comparing version numbers of different software packages. For example, the ordering algorithms of Red Hat and derived distributions differ to those of the Debian-like distributions.

As an example of surprising version number ordering implementation behavior, in Debian, leading zeroes are ignored in chunks, so that 5.0005 and 5.5 are considered as equal, and 5.5 < 5.0006. This can confuse users; string-matching tools may fail to find a given version number; and this can cause subtle bugs in package management if the programmers use string-indexed data structures such as version-number indexed hash tables.

In order to ease sorting, some software packages represent each component of the *major.minor.release* scheme with a fixed width. Perl represents its version numbers as a floating-point number; for example, Perl's 5.8.7 release can also be represented as 5.008007. This allows a theoretical version of 5.8.10 to be represented as 5.008010. Other software packages pack each segment into a fixed bit width; for example, on Microsoft Windows, version number 6.3.9600.16384 would be represented as hexadecimal 0x0006000325804000. The floating-point scheme breaks down if any segment of the version number exceeds 999; a packed-binary scheme employing 16 bits apiece breaks down after 65535.

Use in other media

Software-style version numbers can be found in other media.

In some cases, the use is a direct analogy (for example: Jackass 2.5, a version of Jackass Number Two with additional special features; the second album by Garbage, titled *Version 2.0*; or *Dungeons & Dragons* 3.5, where the rules were revised from the third edition, but not so much as to be considered the fourth).

More often it's used to play on an association with high technology, and doesn't literally indicate a 'version' (e.g., *Tron 2.0*, a video game followup to the film *Tron*, or the television series *The IT Crowd*, which refers to the second season as Version 2.0). A particularly notable usage is Web 2.0, referring to websites from the early 2000s that emphasized user-generated content, usability and interoperability.

Phish 1.0, 2.0, 3.0 and possibly 4.0 after the Covid 19 forced hiatus.

See also

- Continuous data protection
- Maintenance release
- Product life cycle management
- Software engineering

Notes

a. The complete sequence of classic Mac OS versions (not including patches) is: 1.0, 1.1, 2.0, 2.1, 3.0, 3.2 (skipping 3.1), 4.0, 4.1, 5.0, 5.1, 6.0, 7.0, 7.1, 7.5, 7.6, 8.0, 8.1, 8.5 (jumped), 8.6, 9.0, 9.1, 9.2.

References

1. Preston-Werner, Tom (2013). Semantic Versioning 2.0.0. Creative Commons. Retrieved from https://semver.org/spec/v2.0.0.html.
2. Lam, Patrick; Dietrich, Jens; Pearce, David J. (2020-08-16). "Putting the Semantics into Semantic Versioning". arXiv:2008.07069 [cs.SE].
3. "Library Interface Versioning in Solaris and Linux".
4. "Libtool's versioning system". *Libtool documentation*.
5. "Versioning Numbering Concepts - The Apache Portable Runtime Project". Retrieved 2009-04-11.
6. "Daemonite: The science of version numbering". 2004-09-14. Retrieved 2009-04-11.
7. Frank Kyne, Bert de Beer, Luis Martinez, Harriet Morril, Miha Petric, David Viguers, Suzi Wendler."System z Parallel Sysplex Best Practices". 2011. p. 6.
8. "Opera Changelogs for Windows". Opera Software. 2014. Retrieved November 6, 2014.
9. "Home". *Movable Type Documentation Wiki*. June 25, 2013. Retrieved November 6, 2014.
10. "GNU Coding Standards: Releases". GNU Project. 2014-05-13. Retrieved 2014-05-25. "You should identify each release with a pair of version numbers, a major version and a minor. We have no objection to using more than two numbers, but it is very unlikely that you really need them."