Software versioning

From Wikipedia, the free encyclopedia

Software 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 electronic information, whether or not this information is computer software.

Contents

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.

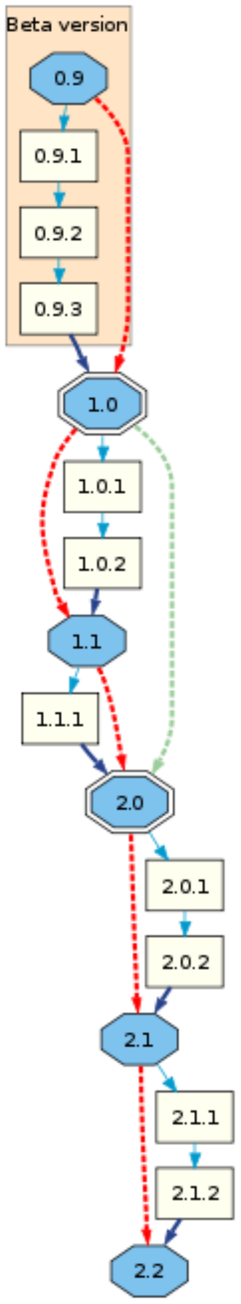
Sequence-based identifiers

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; however, 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.

For instance, in a scheme that uses a four-sequence identifier, the first sequence may be incremented only when the code is completely rewritten, while a change to the user interface or the documentation may only warrant a change to the fourth sequence.



This practice permits users (or potential adopters) to evaluate how much real-world testing a given software release has undergone. If changes are made between, say, 1.3rc4 (a release candidate) and the production release of 1.3, then that release, which asserts that it has had a production-grade level of testing in the real world, in fact contains changes which have not necessarily been tested in the real world *at all*. This approach commonly permits the third level of numbering ("change"), but does not apply this level of rigor to changes in that number: 1.3.1, 1.3.2, 1.3.3, 1.3.4... 1.4.1, etc.

In principle, in subsequent releases, the *major* number is increased when there are significant jumps in functionality such as changing the framework which could cause incompatibility with interfacing systems, the *minor* number is incremented when only minor features or significant fixes have been added, and the *revision* number is incremented when minor bugs are fixed. A typical product might use the numbers 0.9 (for beta software), 0.9.1, 0.9.2, 0.9.3, 1.0, 1.0.1, 1.0.2, 1.1, 1.1.1, 2.0, 2.0.1, 2.0.2, 2.1, 2.1.1, 2.1.2, 2.2, etc. 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 Drupal version 7.12).

A different approach is to use the *major* and *minor* numbers, along with an alphanumeric string denoting the release type, i.e. "alpha", "beta" or "release candidate". 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*) == 1.0. If 1.0rc1 turns out to have bugs which must be fixed, it turns into 1.0rc2, and so on. The important characteristic of this approach is that the first version of a given level (beta, RC, production) *must be identical* to the last version of the release below it: you cannot make any changes at all from the last beta to the first RC, or from the last RC to production. If you do, you must roll out another release at that lower level.

However, since version numbers are human-generated, not computer-generated, there is nothing that prevents arbitrary changes that violate such guidelines: for example, the first sequence could be incremented between versions that differ by not even a single line of code, to give the (false) impression that very significant changes were made.

Other schemes impart meaning on individual sequences:

> *major.minor[.build[.revision]]*

or

> *major.minor[.maintenance[.build]]*

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* [1] format where [2]

> *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.

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.

Designating development stage

Some schemes use a zero in the first sequence to designate alpha or beta status for releases that are not stable enough for general or practical deployment and are intended for testing or internal use only.

It can be used in the third position:

- 0 for alpha (status)
- 1 for beta (status)
- 2 for release candidate

- 3 for (final) release

For instance:

- 1.2.0.1 instead of 1.2-a1
- 1.2.1.2 instead of 1.2-b2 (beta with some bug fixes)
- 1.2.2.3 instead of 1.2-rc3 (release candidate)
- 1.2.3.0 instead of 1.2-r (commercial distribution)
- 1.2.3.5 instead of 1.2-r5 (commercial distribution with many bug fixes)

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 does *not* represent a decimal point, and the sequences do *not* have positional significance. An identifier of *2.5*, for instance, is not "two and a half" or "half way to version three", it is the *fifth second-level revision of the second first-level revision*.

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 1-2-3 Release 1a.

Incrementing sequences

There are two schools of thought regarding how numeric version numbers are incremented: Most free software packages treat numbers as a continuous stream, therefore a free software or open source product may have version numbers 1.7.0, 1.8.0, 1.8.1, 1.9.0, 1.10.0, 1.11.0, 1.11.1, 1.11.2, etc. An example of such a software package is MediaWiki. However, other programs treat version numbers in another way, generally as decimal numbers, and may have version numbers such as 1.7, 1.8, 1.81, 1.82, 1.9, etc. In software packages using this way of numbering 1.81 is the next minor version after 1.8. Maintenance releases (i.e. bug fixes only) would generally be denoted as 1.81a, 1.81b, etc.

The standard GNU version numbering scheme is major.minor.revision,[3] 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.[4] Similarly, Debian package numbers are prefixed with an optional "epoch", which is used to allow the versioning scheme to be changed.[5]

Using negative numbers

There are some projects that use negative version numbers. One example is the SmartEiffel compiler which started from -1.0 and counted upwards to 0.0.[4]

Degree of compatibility

Some projects use the major version number to indicate incompatible releases. Two examples are Apache APR[6] and the FarCry CMS.[7]

Semantic Versioning[8] 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 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.

Date of release

The Wine project formerly used a date versioning scheme, which uses the year followed by the month followed by the day of the release; for example, "Wine 20040505". Ubuntu Linux uses a similar versioning scheme—Ubuntu 11.10, for example, was released October 2011.

When using dates in versioning, for instance, file names, it is common to use the ISO 8601 scheme:[9] YYYY-MM-DD, as this is easily string sorted to increasing/decreasing order. The hyphens are sometimes omitted.

Microsoft Office build numbers are an encoded date:[10] the first two numbers is the number of months passed from the January of the year the project started (with each major Office release being a different project), and the last two numbers are 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 date 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, Microsoft Windows 2000 Server is internally versioned as Windows NT 5.0 ("NT" being a reference to the original product name).

Alphanumeric codes

Examples:

- Macromedia Flash MX

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 the fact that TeX is now very stable, and only minor updates are anticipated. TeX developer Donald Knuth has stated that the "absolutely final change (to be made after my death)" will be to change the version number to π, at which point all remaining bugs will become permanent features.[11]

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

Apple

Apple has a formalised version number structure based around the NumVersion struct, which specifies a one- or two-digit major version, a one-digit minor version, a one-digit "bug" (i.e. revision) version, a stage indicator (drawn from the set development/prealpha, alpha, beta and final/release), and a one-byte (i.e. having values in the range 0–255) pre-release version, which is only used at stages prior to final. In writing these version numbers as strings, the convention is to omit any parts after the minor version whose value are zero (with "final" being considered the zero stage), thus writing 1.0.2 (rather than 1.0.2b12), 1.0.2 (rather than 1.0.2f0), and 1.1 (rather than 1.1.0f0).

Other schemes

Some software producers use different schemes to denote releases of their software. For example, the Microsoft Windows operating system was first labelled with standard numerical version numbers (Windows 1.0 through Windows 3.11). Later, Microsoft started using separate version names for marketing purposes, first using years (Windows 95 (4.0), Windows 98 (4.10), Windows 2000 (5.0)), then using alphanumeric codes (Windows Me (4.90), Windows XP (5.1)), then using brand names (Windows Vista (6.0)). With the release of Windows 7 it appears that Microsoft has returned to using numerical version numbers, although the internal version number for Windows 7 is 6.1.[12]

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.

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 2011 is 140000.[13][14][15]

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 and NT 6.3. Note, however, that Windows NT is only on its fourth major revision, as its first release was numbered 3.1 (to match the then-current Windows release number).

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. For initial development, alpha and beta versions are often given numerical versions less than 1 (such as 0.9), to suggest their approach toward a final "1.0" release. 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. 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 version is released, the "rc" tag is removed.

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 GNOME.

Apple

Apple had their own twist on this habit during the era of the classic MacOS: although there were minor releases, they rarely went beyond 1, and when they did, they twice jumped straight to 5, suggesting a change of magnitude intermediate between a major and minor release (thus, 8.5 really means 'eight and a half', and 8.6 is 'eight and a half point one'). The complete sequence of versions (neglecting revision releases) 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, 8.6, 9.0, 9.1, 9.2.

Mac OS X has departed from this trend, in large part because "X" (the Roman numeral for 10) is in the name of the product. As a result, all versions of OS X begin 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 named version 10.1, followed by 10.2, 10.3, and so on for each subsequent major release (currently at 10.10).

In this system, the third number (instead of the second) denotes a minor release, and a fourth number (instead of the third) denotes bug-fix/revision releases.

Because the first number is always 10, and because the subsequent numbers are not decimal, but integer values, it is that the 11th major version of OS X is labeled "10.10" rather than "11.0".

Political and cultural significance of version numbers

Version 1.0 as a milestone

Proprietary software developers often start at version 1 for the first release of a program and increment the major version number with each significant update.

In contrast to this, the free-software community tends to use version 1.0 as a major milestone, indicating that the software is "complete", that it has all major features, and is considered reliable enough for general release.

In this scheme, the version number slowly approaches 1.0 as more and more bugs are fixed in preparation for the 1.0 release. The developers of MAME do not intend to release a version 1.0 of their emulator program. The argument is that it will never be truly "finished" because there will always be more arcade games. Version 0.99 was simply followed by version 0.100 (minor version 100 > 99). In a similar fashion Xfire 1.99 was followed by 1.100. After 8 years of development, eMule reached version 0.50a.

To describe program history

Winamp released an entirely different architecture for version 3 of the program. Due to lack of backward compatibility with plugins and other resources from the major version 2, a new version was issued that was compatible with both version 2 and 3. The new version was set to 5 (2+3), skipping version 4.[16] A similar situation occurred with UnixWare 7, which was the combination of UnixWare 2 and OpenServer 5.

Keeping up with competitors

There is a common habit in the proprietary software industry to make major jumps in numeric major or minor version numbers for reasons which do not seem (to many members of the program's audience) to merit the "marketing" version numbers.

This can be seen in several Microsoft and America Online products, as well as Sun Solaris and Java Virtual Machine numbering, SCO Unix version numbers, and Corel WordPerfect, as well as the filePro DB/RAD programming package, which went from 2.0 to 3.0 to 4.0 to 4.1 to 4.5 to 4.8 to 5.0, and is about to go to 5.6, with no intervening release. A slightly different version can be seen in AOL's PC client software, which tends to have only major releases (5.0, 6.0, 7.0, etc.). Likewise, 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 browser 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.[17]

Apple

Apple has a particular form of version number skipping, in that it has leveraged its use of the Roman numeral X in its marketing across multiple product lines. Both Quicktime and Final Cut Pro jumped from versions 7 directly to version 10. Like with Mac OS X, the products were not upgrades to previous versions, but brand new programs, branded as Quicktime X and Final Cut Pro X, but unlike Apple's desktop operating systems, there were no major versions 8 or 9. As with OS X, however, minor releases are denoted using a third digit, rather than a second digit. Presumably, major releases for these programs will also employ the second digit, as Apple does with OS X.

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 three times 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 ("Java 5")
- Java 1.6.0 ("Java 6")

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 has recently taken place with the Asterisk open-source PBX construction kit, whose project leads announced that the current version 1.8.x will soon be followed by version 10, which will be followed by version 11, and then 12, and so forth. [18]

Superstition

- The Office 2007 release of Microsoft Office has an internal version number of 12. The next version Office 2010 has an internal version of 14, due to superstitions surrounding the number 13.[19]
- 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".
- Nokia decided to jump directly from S60 3rd Edition to S60 5th Edition, skipping the fourth edition due to the tetraphobia of their Asian customers.
- 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).

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.[20]

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. (It should be noted 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, by being able to compare their copy of the software product against another copy, such as the newest version released by the developer. For the programmer team or company, versioning is often used on a file-by-file basis, where individual parts or sectors of the software code are compared and contrasted with newer or older revisions, often in a collaborative version control system. There is no absolute and definite software version schema; it can often vary from software genre to genre, and is very commonly based on the programmer's personal preference.

Significance in technical support

Version numbers allow people providing support to ascertain *exactly* what code a user is running, so that they know what bugs might affect a problem, and the like. This occurs 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.

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 will 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 will pack each segment into a fixed bit width, for example, **5.8.7** could be represented in 24 bits: (**5** << 16 **|** **8** << 8 **|** **7**; hexadecimal: 050807; for version 12.34.56 in hexadecimal: 0C2238). The floating-point scheme will break down if any segment of the version number exceeds 1,000; a packed-binary scheme employing 8 bits apiece after 256.

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, *Dungeons & Dragons* 3.5, where the rules were revised from the third edition, but not so much as to be considered the fourth), but 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 the World Wide Web as used in collaborative projects such as wikis and social networking websites.

See also

- Continuous Data Protection
- Maintenance release
- Microsoft Version Number
- Point release
- Product life cycle management
- Revision control
- Software engineering
- Software release life cycle

References

1. ^ "Library Interface Versioning in Solaris and Linux" (http://static.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/browndavid/browndavid_html/).
2. ^ "Versioning" (http://gnuwin32.sourceforge.net/versioning.html).
3. ^ "GNU Coding Standards: Releases" (https://www.gnu.org/prep/standards/html_node/Releases.html#index-version-numbers_002c-for-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."

4. ^ *a* *b* "Advogato: Version numbering madness" (http://www.advogato.org/article/40.html). 2000-02-28. Retrieved 2009-04-11.
5. ^ Debian Policy Manual, 5.6.12 Version (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version)
6. ^ "Versioning Numbering Concepts - The Apache Portable Runtime Project" (http://apr.apache.org/versioning.html). Retrieved 2009-04-11.
7. ^ "Daemonite: The science of version numbering" (http://blog.daemon.com.au/archives/000276.html). 2004-09-14. Retrieved 2009-04-11.
8. ^ Preston-Werner, Tom (2013). Semantic Versioning 2.0.0. Creative Commons. Retrieved from http://semver.org/spec/v2.0.0.html.
9. ^ Markus Kuhn (2004-12-19). "International standard date and time notation" (http://www.cl.cam.ac.uk/~mgk25/iso-time.html). University of Cambridge. Retrieved 2009-04-11.
10. ^ Jeff Atwood (2007-02-15). "Coding Horror: What's In a Version Number, Anyway?" (http://www.codinghorror.com/blog/archives/000793.html). Retrieved 2009-04-11.
11. ^ Donald E. Knuth. *The future of TeX and METAFONT* (http://www.ntg.nl/maps/05/34.pdf), NTG journal MAPS (1990), 489. Reprinted as chapter 30 of *Digital Typography*, p. 571.
12. ^ Enter the "VER" command in Windows 7 CMD
13. ^ "BLAG Linux And GNU" (http://www.blagblagblag.org/download/index.html). *DistroWatch.com*. Retrieved 29 September 2011.
14. ^ "News and Updates: BLAG" (http://distrowatch.com/index.php?distribution=blag). *DistroWatch.com*. Retrieved 29 September 2011.
15. ^ "blag download" (http://www.blagblagblag.org/download/index.html). *blag*. Retrieved 29 September 2011.
16. ^ "Winamp Media Player FAQ" (http://www.winamp.com/help/FAQ#10).
17. ^ "Slackware FAQ" (http://www.slackware.com/faq/do_faq.php?faq=general#0).
18. ^ Kevin P. Fleming (July 21, 2011). "The Evolution of Asterisk (or: How We Arrived at Asterisk 10) | Inside the Asterisk" (http://blogs.digium.com/2011/07/21/the-evolution-of-asterisk-or-how-we-arrived-at-asterisk-10/). Digium, Inc. Retrieved 2014-05-25.
19. ^ Paul Thurrott (2009-05-14). "Office 2010 FAQ" (http://www.winsupersite.com/office/office2010_faq.asp). Retrieved 2009-12-30.
20. ^ Finnie, Ryan (2010-10-23). "I'm sorry" (http://lists.colobox.com/pipermail/finnix/2010-October/000121.html). Retrieved 2012-02-09.

External links

- 3 Effective Techniques For Software Versioning (http://10kloc.wordpress.com/2013/01/05/3-effective-techniques-for-software-versioning/)
- Software Release Practice Howto (http://tldp.org/HOWTO/Software-Release-Practice-HOWTO/index.html)
- Software version numbering (http://www.everything2.com/index.pl?node_id=1128644) at Everything2
- Semantic Versioning Specification (http://semver.org/) (SemVer)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Software_versioning&oldid=628185551"

Categories: Revision control | Software version histories | Software release