Labelling Unicon Binary Releases

Don Ward Jafar Al Gharaibeh

June 8^{th} 2020

Different versions of software must be labelled so that they may be readily distinguished. The purpose of any version labelling scheme is to bring some order to a possible tangle of different revisions. Ideally, the scheme should allow the user to determine quickly whether one release is earlier[1] than another.

A Scheme to Label Unicon Releases

We propose using git tags with a structured name to identify and describe Unicon releases. The tag name will have three parts described by three numbers – Major.Minor.Bugfix. Major.Minor is the version of Unicon that is stored in `src/h/version.h`; Bugfix starts at 0 and will increment by one on each binary release of Unicon that has Major.Minor as the first part: it will reset to 0 when Major.Minor changes. Major.Minor.0 releases will normally be labelled as Major.Minor without the trailing .0.

Although "Bugfix" implies that the reason for a subsequent release based on Major.Minor is merely to correct errors, it could also contain new features that are not considered important enough to bump Major.Minor but are desirable to release in advance of the next Major.Minor binary. There is some further discussion later on about the implications of this.

As an aside, we considered using the form Major.Minor.Patch governed by the popular Semantic Versioning specification [1] but decided against it because the Unicon project is unlikely to follow the strict rules about when the component numbers must alter. Our proposal is similar in spirit to [1] but not as prescriptive.

The svn commit history was linear: after the transition from svn to git, some effort is being made currently to ensure that commits made into the

[1] We are not using "earlier" in a strict chronological sense here: one release might be earlier than another in developmental terms, even though it was released afterwards.

git master branch continue to form a linear sequence by rebasing new pull requests onto its tip. Rebasing involves some extra (manual, and possibly error prone) steps during the commit process and, as we become more familar with git, we may prefer to use merge commits instead – which will mean a departure from a strictly linear commit history – although, if we zoom out, the commit history will still appear linear in the large. In any case, the release numbering scheme and the release process itself must be able to deal with a general graph (of commits) as well as a tree, or a simple sequence.

In the absence of maintenance releases, and even though the commit graph itself may not be linear, we think that a zoomed out view that only has releases in it should be linear, which leads to some rules about which commits are eligible to build a release. This is nothing more than expecting the natural taxonomic ordering implied by Major.Minor.Bugfix to be followed in practice – nobody would expect either 10.5.1 or 10.4.4 to be earlier than 10.4.3. The rules are discussed in detail in an appendix.

We believe that "Maintenance releases", be they urgent bug fixes or feature enhancements, will inevitably lead to the development of different revisions of Unicon in parallel. The reason is that it is highly likely, when we want to go back and fix something, that we will want to change just that one thing and not bring along every change that has happened subsequently – i.e. we will not want to fix it by merging the necessary modifications into the tip of the current master branch and releasing that. Our simple linear sequence of releases must become a tree. It *could* become a general graph – and git would happily cope with that – but we are proposing that it does not: maintenance releases should be on "Long Term Branches" (LTBs) that we are **never** going to merge back into the master branch. In time, the release tree will thus come to resemble the iconic Saguaro cactus. Note that the use of LTBs does not preclude cherry-picking commits from the master branch into the LTB or vice-versa.

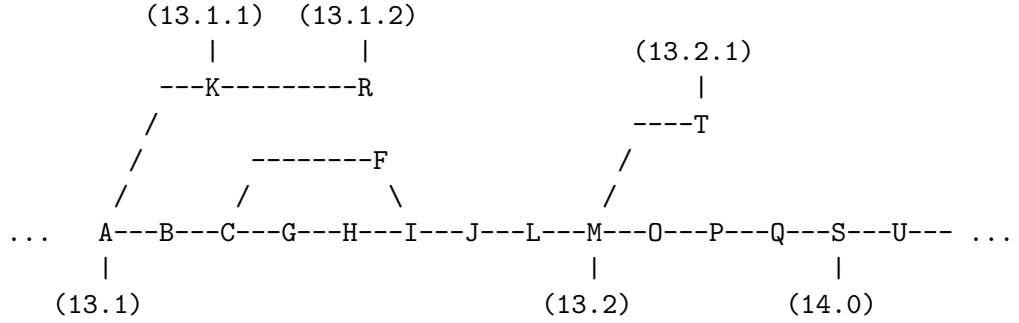


Figure 1: An example release tree

The Major.Minor.Bugfix labelling scheme cannot handle two parallel developments with the same Major.Minor and preserve the natural taxonomic ordering if both of them result in a release. A binary release (of Major.Minor.0) thus defines the *end* of the development of Major.Minor on the master branch: the first act after the release should be to increment Minor on the master branch. So, in the figure above, B (and subsequent commits on the master branch) would have a version of 13.2.

This brings up an interesting question: at what point between O and S may the major version change to 14? The answer is any of them. O (and its descendants) would start off at 13.3 but they are all pre-release, so the version may change at any point: i.e. a change in version number does not trigger a release, whereas a release does trigger a change in version number. Although the Major version changed in the example given above, there is no implication that a new release *must* alter the Major version number.

Displaying the Release Details

The present system displays five separate pieces of information that can help to classify and order different releases: There is Major.Minor and `Version Date`, all contained in the string returned from `&version`, and a "repo-count" combined with a git hash that is returned as the `Revision` string from `&features`. Only the git hash uniquely identifies the revision, but it contains no clue as to which revision is earlier when comparing two arbitrary builds. When the history is largely linear, as it is now, the "repo-count" is a good rough and ready comparator but it might fail in the presence of a release tree (as opposed to a release sequence).

We propose that the version string be derived automatically from the git meta-data, based on the name of the branch – "`master`" or something else – and whether or not the tip of the branch is tagged.

The version string will be derived using the following rules

1. The version date is the date of the HEAD commit.
2. If the HEAD commit has a tag name, that tag name is the version number.
3. If HEAD is untagged the version number depends on the branch name
   (a) The master branch will use a version number of $VERSION (pre-release)

(b) Other branches will use the output of `git describe`, which will be something like TAG-n-hash (for example 13.1-176-g943d2b72)

13.1	is the name of the most recent release tag (starting from HEAD and going backwards towards the root until a release tag is found).
176	is the number of commits between HEAD and the tag.
943d2b7	is the short hash of the HEAD commit.

In the table of example version strings below, the source code was exactly the same: only the git meta-data changed for each build. Each version string begins with "Unicon Version ", which has been omitted from the table.

Version "number"	Version Date	comments
13.2 (pre-release)	07-Jun-2020	Built on the master branch with HEAD untagged.
13.1-176-g943d2b72	07-Jun-2020	Built with the master branch renamed to something else (simulating a non-release build on a LTB).
13.1.1	07-Jun-2020	Built with a tag of 13.1.1 pointing at HEAD (simulating a maintenance release build on a LTB).
13.1.1	07-Jun-2020	Built on the master branch with a tag of 13.1.1 pointing at HEAD (demonstrates that a tag overrides the branch name).

Note that users who use binary releases will only ever see a version string like Unicon Version 13.2 07-Jun-2020. Users who build from the source will usually see version strings like the first example (i.e. containing "(pre-release)") The only time something like 13.1-176-g943d2b72 will ever appear in the version string is if the user has downloaded the latest source and has switched to another branch before making a build. In such a case we assume they know enough to unscramble the output of `git describe`.

When the compiler is in use, the string "(iconc) " is automatically placed between the version number string and the date, as at present.

In future, should we decide to go down the release candidate route, we can tag a release candidate with an appropriately named tag – for example 13.5-RC1 – and the version string will automatically include the information. Note that a release candidate would not be considered to be a suitable starting point for an LTB[2]; if there were a problem, we'd fix it by by releasing

[2]We have chosen to make all Major.Minor releases on the master branch but there is nothing fundamental about this. An alternative would be to start the LTB at 13.5-RC1 and then put RC2 on that branch until, ultimately, 13.5 is released (on the LTB, rather

13.5-RC2 until, ultimately, 13.5 is released and the master branch moves on to 13.6.

A non-release build will not have a release tag associated with it. The user must make sense of the version strings (which include the git hash plus the number of commits back to the last release) to decide which of two builds is newer. But, to be in this situation in the the first place, the user must have used git to retrieve the source to build it: we assume they are capable of also interrogating git to establish the relative ordering of the two builds – if it isn't obvious – should the need arise.

References:

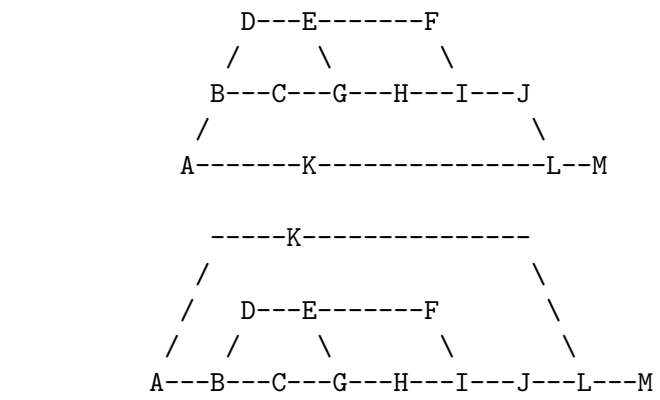
[1] Preston-Werner, Tom (2013). Semantic Versioning 2.0.0. `http://semver.org/spec/v2.0.0.html`.

than the master). In such a scheme, the master branch would go to 13.6 immediately after the release of 13.5.0-RC1.

If a long time elapses between 13.5-RC1 and 13.5 this scheme might be preferable because it allows the development of 13.6 to proceed on the master branch without entangling it in the release of 13.5. It's a decision we can defer until we decide to adopt the release candidate approach to releases.

Appendix: Choosing a release point

If the commit graph is linear then it's simple: any commit will serve as a release and the taxonomic ordering will automatically apply. If there are branches in parallel with a potential release commit then some care must be taken – we should choose only one of the branches to make releases from. Consider the graph below, which is taken from `git log --help`. We have drawn it twice to highlight the point that there is no "main line" between A and L, just descendants of A and parents of L.




```
A < { K, B < { {D < E < F}, { C < G < H} } < I < J } < L < M
E < G

Consistent release combinations are a subset of
    {AKLM}            ( not {BCDRFGHIJ} )
or
    {ABDEGHIJLM}      ( not {CFK} )
or
    {ABDEFIJLM}       ( not {CGHK} )
or
    {ABCGHIJLM}       ( not {DEFK} )
```

Figure 2: A commit graph

To maintain the taxonomic ordering, we must pick release points from only one of the alternatives listed. The easiest and least error-prone policy is to make a release from a part of the commit graph that has nothing in parallel with it and is thus, unambiguously, part of the main line. In the diagram above, that would be one or more from {ALM}. The same

rules apply to a release from a LTB with the proviso that we only consider commits on the LTB, rather than the master. If these rules are followed then the Major.Minor.Bugfix label describes the position of an individual release in the release tree exactly.

Appendix: A Bestiary of Unicon Releases

In this table, we identify the points at which the version details changed and also (as far as is practicable) the points at which a binary release was made. The table also contains suggestions to tag the historical releases. Note that the historical "Bugfix" releases before 13.1 do not correspond with our naming proposals because they are in the main line rather than on LTBs but, since there are no plans to make any maintenance releases before 13.1.1 – and, perhaps, not even then – it doesn't really matter.

Windows Date is the Version Date reported by the Windows binary.

SVN rev	git hash	Version	Version Date	Windows Date	git tag
	a37d2288	13.1	November 6, 2019		
	f22d949a	13.1	August 19, 2019		
5965	dce80670	13.1	March 2, 2019	March 24, 2019	13.1
5920	283f5909	13.1	March 2, 2019		
5888	c27495fd	13.1	January 19, 2019		
5854	5eb096f4	13.0	December 6, 2018		
5599	a6a53187	13.0	August 10, 2017		
5080	380c9ec5	13.0	April 16, 2017	April 16, 2017	13.0.1
5076	2f471f57	13.0	April 16, 2017		
5039	8ddc1399	13.0	April 10, 2017		
4777	37b670df	13.0	Feb 1, 2017	Feb 1, 2017	13.0
4768	fad1e049	13.0	Jan 30, 2017		
4500	3de78ac1	13.0	Aug 30, 2016		
4237	e9bb5587	12.3	Feb 29, 2016	Feb 29, 2016	12.3
4232	8cb66e3a	12.3	Feb 29, 2016		
4166	e717f130	12.3	Sep 12, 2015		
3976	2edead79	12.2	Dec 2, 2014	Dec 2, 2014	12.2
3975	2b1344d4	12.3	Dec 2, 2014		
3883	250626b8	12.1	July 17, 2014		

continued ...

SVN rev	git hash	Version	Version Date	Windows Date	git tag
3420	eee39677	12.1	April 13, 2013	April 13, 2013	12.1.2
3312	a89fd52c	12.1	November 19, 2012		
3231	4ae6595f	12.1	August 06, 2012	August 12, 2012	12.1.1
3211	7f7f54e6	12.1	August 06, 2012	August 6, 2012	12.1
3101	3dc88528	12.1	May 31, 2012		
3090	509e849d	12.1	May 21, 2012		
2983	9db42528	12.0 or 11.9	November 2, 2011	November 2, 2011	12.0.3
2857	6c12273b	12.0 or 11.9	November 2, 2011	November 2, 2011	12.0.2
2788	bdffe5e7	12.0 or 11.9	July 15, 2011	July 15, 2011	11.9.1
2786	39258419	11.7	January 14, 2011	July 13, 2011	12.0.1
2708	547d0670	11.7	January 14, 2011	March 10, 2011	12.0 11.9
2675	11466f2e	11.7	January 14, 2011	February 22, 2011	11.8
2623	39a3e916	11.7	January 14, 2011		
2535	268a2a6a	11.7	January 22, 2010	January 22, 2010	11.7.1
2237	7465a6d4	11.7	January 22, 2010	January 22, 2010	11.7
2094	14bf54ab	11.6	October 12, 2009		
1932	5ee2eb2f	11.5	March 22, 2009		
1636	a683b0ad	11.5	April 27, 2008		
1362	5b279165	11.4	January 10, 2007		
1273	018bbb2a	11.3	March 20, 2006		
1213	5c306bdc	11.3 (beta)	November 30, 2005		
1117	85d2d3f2	11.3 (beta)	June 23, 2005		
1078	95d17114	11.3 (beta)	April 28, 2005		
1051	1eb51620	11.2 (beta)	February 18, 2005		
1018	4e83fa58	11.2 (beta)	January 4, 2005		
922	c73adaed	11.1 (beta)	October 18, 2004		
882	0df3e550	11.1 (beta)	July 6, 2004		
810	794f8018	11.0 (beta)	May 15, 2004		
672	520e9aea	11.0 (beta)	February 1, 2004		
548	63b5dcc1	11.0 (beta)	July 8, 2003		
520	175cc419	11.0 (beta)	June 2, 2003		
492	d7388853	11.0 (beta)	May 24, 2003		
484	11eea964	11.0 (beta)	May 23, 2003		
411	bf6174b2	10.0 (beta-3)	November 25, 2002		
395	f7804ff1	10.0 (beta-3)	October 25, 2002		
369	a5cadd88	10.0 (beta-3)	August 24, 2002		
274	4ea85e6f	10.0 (beta-3)	April 27, 2002		
244	441aad90	10.0 (beta-3)	February 14, 2002		
58	be738a6a	10.0 (beta-2)	August 1, 2001		
3	984fa078	10.0 beta	January 18, 2001		