Mardis du rêve

Brice Goglin
Services SED & Com

2014/11/18

Software Releases : Why, When and How ?

II

Return of the Revenge

Agenda

- Before we start
- Why?
- When?
- Naming?
- Building?
- Publishing?
- After we end

1 Before we start

Why publishing our code?

- Help the society progress
  - Give the result of our work
- Research results should be reproducible
  - Software freely available online

- You **must** have your **employer agreement**
  - Your code belongs to your employer
  - Ask your employer (STIP, etc.)
    - **Before choosing the license and publishing online!**

2 Why Software Releases?

Why do we need software releases?

- Show we're alive
  - To partners, evaluation committees, competitors
- Doesn't mean they will use our releases...

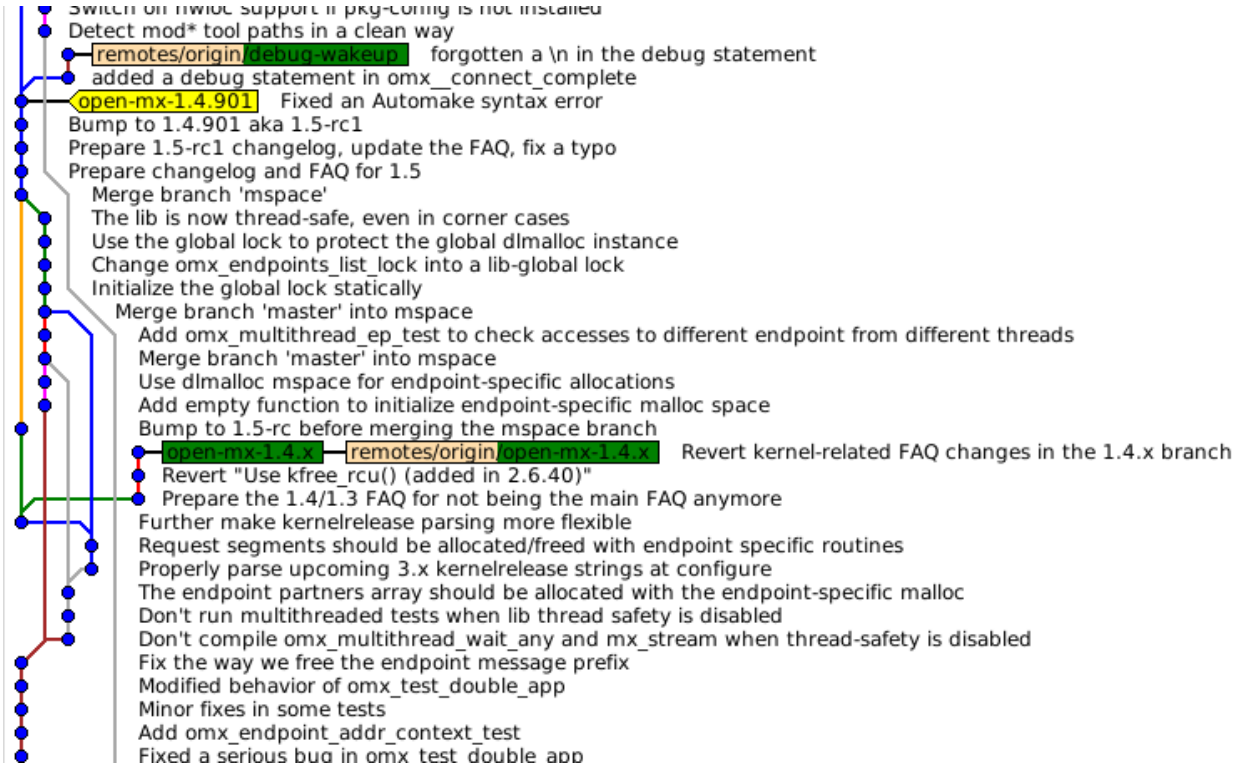
- « Software without users is dead software. »
  – F. P.

Why do we need software releases?

- People want to test your code
  - When you read an article, if interesting, you may want to take a deeper look
- If a reader wants to test but can't, he will cite your work **negatively**
  - « Doesn't work », « Code isn't available », « Not portable », « Not ready for production », …
    - His readers will believe his conclusion without checking
      - Researchers don't have time to double-check everything
- Your code must be available online
  - And easy to use!
    - And it should (somehow) work...

But my SVN is public?!

- A SVN repository isn't easy to use
  - « Which autotools version do I need? »
  - « Which SVN/GIT revision? »
  - « Which branch? »



  - « Is this branch currently broken? »

Releases tell you the right revisions and makes them easy to use

- Releases are ready to use
  - No need for recent autotools, etc.
- Releases are tested before publication
  - Contents won't change anymore
  - Can be marked *development*, *beta*, *experimental*...
    - Users know the code won't work perfectly
    - But they will know whether their required features are available

3 When?

Planning your releases

- Periodic releases every 6 months?
  - Like Ubuntu
  - Good for users
    - They know what's coming and when
  - Hard for developers
    - They don't sleep anymore when the deadline approaches
    - Hardly possible in research
      - We have so many other deadlines...

Planning your releases

- Feature-based releases?
  - Sounds good
  - Except when features are delayed by months or years
    - If you don't release often, your software looks dead
    - Periodic developer meetings
      - Update the list of features that may **really** be included in the next release

Avoid long delays
Release early, Release often

- When huge delays between releases, bug reports are unrelated to the code that developers work on
  - Developers' replies don't appear nice to users
    - « It's fixed in trunk, but it's not stable yes, it will be usable in v1.8 which is planned for release in 2 months »
    - « You use a very old release. Of course this cannot work. »
    - Note: Be nice to users, especially those that report bugs
- New features are tested late (when the new release is published), so you will get bug reports very late
  - Users will never be happy

Be careful with major releases

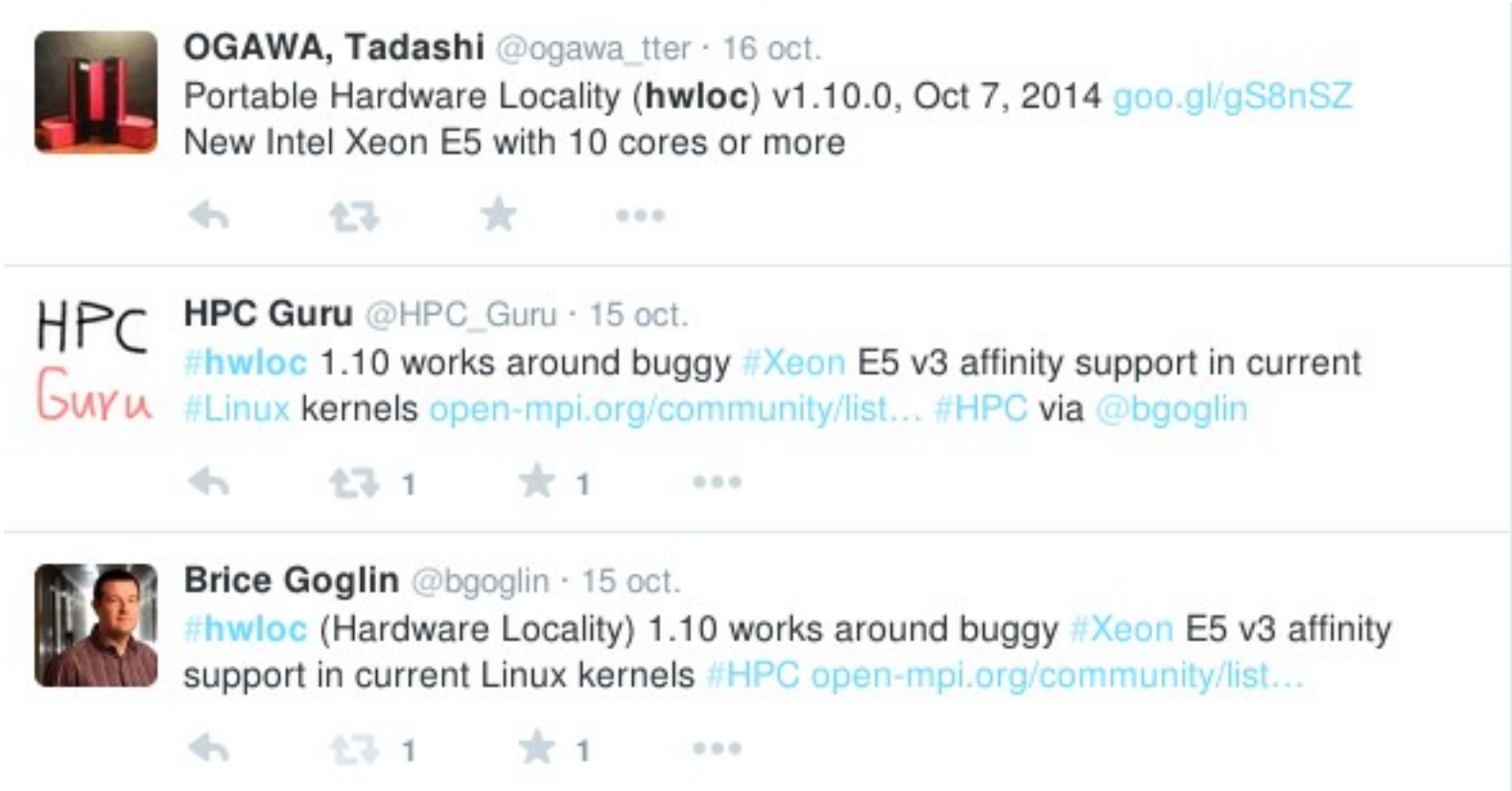
- Try to get feedback/testing early
  - Publish **Release Candidates** as soon as possible?
    - People know RC aren't bug-free
    - Doing multiple RC isn't a bad idea
  - Maybe even Beta releases first? (or nightly snapshots)
- In the meantime, publish stable releases to fix existing bugs
  - Don't wait for a big bug fix before publishing 15 small fixes that have been waiting for months
    - Stable release when there are many bug fixes or a long delay since the last release
  - Nightly snapshots can help **temporarily**

Adapt to events

- Release code **before** the buzz occurs
  - Conference talk
  - Exhibition/Demonstration
  - Press release or any other big news
  - Anything that will cause people to suddenly look at your work
    - They will want to test your wonderful features now
    - If they want but cannot test, or if it fails, they won't come back soon
      - And they will cite you negatively (again)

Adapt to others

- Follow related press releases
  - Ex: when vendors make the buzz about some new products



Talk and plan

- With colleagues
  - Assert branches' stability before releases
    - Is the SVN trunk or git master stable?
    - Which branches are ready to be merged?
  - Synchronize early to avoid late unexpected problems
- With users
  - Assert bug criticalness
    - Grave or not? Rare or not? Any easy workaround?
      - Needs immediate release to fix it?
  - Publish the roadmap for earlier feedback?

How **NOT** to plan releases?

- Too often, too close
  - If you release 1.3.3 and 1.4.0 at the same time, users will only test one of them
    - May be OK if you have many users with different needs
  - Delay between releases so that feedback doesn't arrive too late
    - At least one week between RC and final release?

How **NOT** to plan releases?

- Too early
  - Releasing an immature programming API may cause severe issues later
    - Breaking the ABI (change functions, rename, …) is bad
    - How about marking part of the API as experimental ?
      - Avoid as much as possible, otherwise users will never trust this unstable API
- Right before the week-end?
  - If people don't test right after the release, they won't test until Monday, but they may forget or have something else to do by then

4 Naming?

How to name my releases.

- *Alpha, Beta, Dev, RC, Stable, …*
  - Naming has an impact on usage
    - Very few people actually play with « Alpha » releases
    - Linus names all Linux prereleases « RC » ?
      - This has been widely explained/documented in public
- Use clear release identifiers
  - Number and/or dates
    - NOT mysoft-last.tar.gz !
  - So that bug reports are exploitable
    - Make it easy to locate which version is affected
      - So that you can reproduce and debug

Numbering releases

- Version numbers should match reality
- Don't wait until EVERYTHING is implemented before releasing 1.0
  - Otherwise **1.0 will NEVER appear**
    - Research projects are rarely completed?
- Release a 1.0 release that **works** even if **few features** are available
  - Then do 1.1 or 2.0 when other big features are ready

1st example of (bad) numbering: 1.0 came way too late

1.0.0	October 8th 2012	ac61ddb6ff7fa261c76b3f9172057cad	3598f
Older releases:			
0.9.8	February 21st 2012	9f4ed565a333be68959ae19c41737b38	148db
0.9.7	July 29th 2011	7576a0b991075becc4e43b3fe16227b6	f4360
0.9.6	April 2nd 2011	9ae1a4066cb02cda0adbfede28072019	df1fa
0.9.5	February 1st 2011	794fb4e653948d9008c492d9eef0bd03	a6272
0.9.4	December 16th 2010	ed4aab1b04b1718b3c1ac8daa06a4399	4aad
0.9.3	October 20th 2010	a26b52713da37aba7492e63126643e4d	96de
0.9.2	August 30th 2010	e1e7a3254096aca26559958964d7f005	08789
0.9.1	July 6th 2010	6bf08346536d6e975555925341ab2062	af3a6
0.9.0	June 14th 2010	2b10c1daee8a260023b41d81c19f3f52	21ad5
0.8.901	May 15th 2010	64fce0f69689c5da19100fa70d2fff3e	09b0d
0.8.1	March 22nd 2010	f9849456e624b333f7100602a731de3a	cdee1
0.8.0	March 3rd 2010	8543e2aba5272d8caff1c828bae48ffe	069f9

2nd example of (bad) numbering: 1.0 was late

1.0.1	29.03.2012	[illegible]-1.0.1.tar.gz	
1.0.0	28.03.2012	[illegible]-1.0.0.tar.gz	
0.9.2	06.10.2011	[illegible]-0.9.2.tar.gz	
0.9.1	13.05.2011	[illegible]-0.9.1.tar.gz	
0.9-win32	12.05.2011	[illegible]-win32-build.zip	s
0.9	12.05.2011	[illegible]-0.9.tar.gz	
0.4-win32	25.08.2010	[illegible]-win32-build.zip	s
0.4	25.08.2010	[illegible]-0.4.tar.gz	
0.2	04.07.2009	[illegible]-0.2.tar.gz	
0.1	02.04.2009	[illegible]-0.1.tar.gz	

Numbering releases

- Don't underestimate the psychological impact of version numbers
  - 0.5 = « half ready/useful/stable/... »
  - 0.10 = even worth
    - Looks like it's 10% ready/useful/stable
    - And it will never reach 1.0
      - It will never be ready!
- Think « x.y » is too complicated?
  - Mozilla numbers Firefox 33, 34, 35, … every 6 weeks

5 Building?

How to build a release?

- Must be **easy** to install and use
  - No need to install bleeding-edge dependencies
    - Autotools, Cmake, doxygen...
  - configure && make should be enough
- There are plenty of tools to help
  - make dist and make distcheck with automake
  - Cmake/Cpack
  - Stuff for mobile apps (but I have no clue about these)
- Note that some build systems are not auto-sufficient (CMake)
  - Users must install them before compiling your project
    - Don't depend on specific versions

What to distribute inside releases?

- Including dependencies?
  - Usually, no
  - If a dependency is rarely installed and is hard to compile?
  - If a dependency has an unstable ABI ? Could be useful.
- **Beware of licenses**
  - Note in the COPYING file the licenses of each dependency that you are distributing
  - Ask SED or STIP for help
- Don't reinvent the wheel just to avoid adding yet another dependency!

What to enable by default in releases?

- **Everything !**
  - Most users just use the default configuration
    - Everything else will be rarely tested
- Do not add too many configuration options
  - Users won't know how to configure too many things
    - Linux has more than 15 000 options, very few people configure them, distributions enable almost everything
  - Users will say your software doesn't work well
    - Just because the default configuration isn't very good

What to enable by default in releases?

- What about my « experimental » code ?
  - Enable it by default
    - Use release candidates for more testing
  - If it breaks everything, it's too experimental
    - If shouldn't be in the main SVN/GIT branch
    - What does « experimental » mean?
- Don't distribute old unmaintained code
  - If somebody tests it and reports a failure, you'll have to explain that this code is useless and that nobody will ever fix it
    - This old code is saved in the SVN/GIT history anyway

What to check **BEFORE** a release ?

- Don't break the ABI
  - Except if **really** necessary
    - So don't break the ABI multiple times
      - Group all ABI breaks in one big 2.0 major release?
- What if a bug fix requires an ABI break?
  - Shouldn't happen!
  - Needs a major release?
    - Synchronize it with other big pending changes
- Think about your API/ABI in advance. Think more. Again.
  - Choose your prototypes, names, … wisely
- There exists rules for ABI versioning
  - Libtool documentation, etc.

What to check **BEFORE** a release ?

- **Check that things work!**
  - On real machines (what end users will use)
    - **Not only on your laptop!**

- This slide is where you remember you should really have a look at http://ci.inria.fr
  - See previous talks

What to check AFTER a release ?

- Document releases in SVN/GIT
  - Tags for each release
    - Makes *bissecting* much easier
      - For locating which commit introduced a bug
- Change the trunk version number right after the release?
  - Lets you distinguish between the released code and the SVN trunk that got broken 2 hours later
    - Useful for locating faulty commits again

6 Publishing?

How to publish a release?

- Tarball?
  - Some OS don't like tarballs
    - .zip or .exe for Windows, mobile apps, etc.
  - Some Unix want .tar.gz instead of .tar.bz2
- Clearly show the release date
  - « I don't remember if 1.0 was already available when I checked 2 months ago. »
  - « Not sure if my SVN checkout is more up-to-date than 1.3 »
- Show MD5/SHA1 sums and sign files?
- Fill the form when publishing on Gforge
  - Gives details about the release

Documenting releases

- Don't mix NEWS and ChangeLog
  - ChangeLog is for developers (git log)
  - NEWS is for end users
    - NEWS should be **short and clear**, listing big changes and bug fixes
      - If too long, nobody will read it
    - Thank contributors (bug reports, patchs, tests, etc.)
      - Remember: Be nice to bug reporters
      - It shows your software attracts contributors
    - Insert all previous releases' NEWS in the last release
      - « I don't remember if 1.4.3 bug fixes are included in 1.5.1 »

Documenting releases

- Display the SVN/GIT version in releases
- Publish the NEWS file
  - Inside the tarball, announce mails, web pages, etc.
  - Users should find releases' NEWS without having to be subscribed to foo-announce@mysite.org

Announcing a release?

- Not just adding a new tarball on Gforge
- Mailing lists
  - Check that people are not on foo-devel@ without being on foo-announce@
  - Include NEWS and download link, etc.
- Add a news on the project website
  - With links (again)
  - And the website must show that the project is alive
    - New papers, demo, etc.

Announcing releases (better)

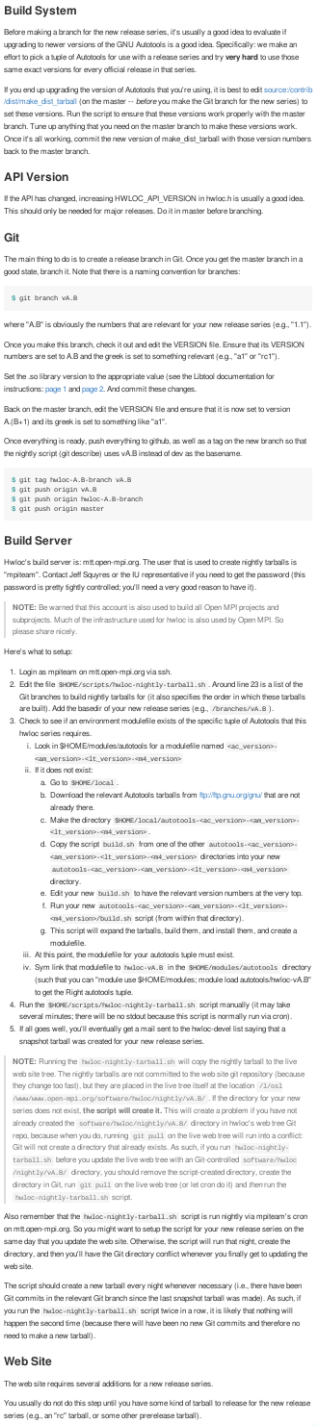
- News on team website, personal website...
  - Or even « *Scotch breaks the 32bits barrier* » on inria.fr
  - Make Google good at finding your software news
- Wider announces for major releases
  - Mailing lists of related projects
  - Tweets, blogs, etc.
    - Teasing for upcoming releases/features?
- Install on local platforms (PlaFRIM, MCIA…) and tell the admins to announce it
  - Neighbors and colleagues are better/easier bug reporters and debuggers

7

After we end

How much time do I waste?

- Spend some time documenting the release process
  - and maintaining it
- Continuous Integration to ease debugging
  - and automatic building of releases



Build System

Before making a branch for the new release series, it's usually a good idea to evaluate if upgrading to newer versions of the GNU Autotools is a good idea. Specifically: we make an effort to pick a tuple of Autotools for use with a release series and try **very hard** to use those same exact versions for every official release in that series.

If you end up upgrading the version of Autotools that you're using, it is best to edit source/contrib/dist/make_dist_tarball (on the master -- *before* you make the Git branch for the new series) to set these versions. Run the script to ensure that these versions work properly with the master branch. Tune up anything that you need on the master branch to make these versions work. Once it's all working, commit the new version of make_dist_tarball with those version numbers back to the master branch.

API Version

If the API has changed, increasing HWLOC_API_VERSION in hwloc.h is usually a good idea. This should only be needed for major releases. Do it in master before branching.

Git

The main thing to do is to create a release branch in Git. Once you get the master branch in a good state, branch it. Note that there is a naming convention for branches:

```
$ git branch vA.B
```

where "A.B" is obviously the numbers that are relevant for your new release series (e.g., "1.1").

Once you make this branch, check it out and edit the VERSION file. Ensure that its VERSION numbers are set to A.B and the greek is set to something relevant (e.g., "a1" or "rc1").

Set the .so library version to the appropriate value (see the Libtool documentation for instructions: page 1 and page 2. And commit these changes.

Back on the master branch, edit the VERSION file and ensure that it is now set to version A.(B+1) and its greek is set to something like "a1".

Once everything is ready, push everything to github, as well as a tag on the new branch so that the nightly script (git describe) uses vA.B instead of dev as the basename.

```
$ git tag hwloc-A.B-branch vA.B
$ git push origin vA.B
$ git push origin hwloc-A.B-branch
$ git push origin master
```

Build Server

Hwloc's build server is: mtt.open-mpi.org. The user that is used to create nightly tarballs is "mpiteam". Contact Jeff Squyres or the IU representative if you need to get the password (this password is pretty tightly controlled; you'll need a very good reason to have it).

> **NOTE:** Be warned that this account is also used to build all Open MPI projects and subprojects. Much of the infrastructure used for hwloc is also used by Open MPI. So please share nicely.

Here's what to setup:

1. Login as mpiteam on mtt.open-mpi.org via ssh.
2. Edit the file `$HOME/scripts/hwloc-nightly-tarball.sh`. Around line 23 is a list of the Git branches to build nightly tarballs for (it also specifies the order in which these tarballs are built). Add the basedir of your new release series (e.g., `/branches/vA.B`).
3. Check to see if an environment modulefile exists of the specific tuple of Autotools that this hwloc series requires.
   i. Look in $HOME/modules/autotools for a modulefile named `<ac_version>-<am_version>-<lt_version>-<m4_version>`
   ii. If it does not exist:
      a. Go to `$HOME/local`.
      b. Download the relevant Autotools tarballs from ftp://ftp.gnu.org/gnu/ that are not already there.
      c. Make the directory `$HOME/local/autotools-<ac_version>-<am_version>-<lt_version>-<m4_version>`.
      d. Copy the script `build.sh` from one of the other `autotools-<ac_version>-<am_version>-<lt_version>-<m4_version>` directories into your new `autotools-<ac_version>-<am_version>-<lt_version>-<m4_version>` directory.
      e. Edit your new `build.sh` to have the relevant version numbers at the very top.
      f. Run your new `autotools-<ac_version>-<am_version>-<lt_version>-<m4_version>/build.sh` script (from within that directory).
      g. This script will expand the tarballs, build them, and install them, and create a modulefile.
   iii. At this point, the modulefile for your autotools tuple must exist.
   iv. Sym link that modulefile to `hwloc-vA.B` in the `$HOME/modules/autotools` directory (such that you can "module use $HOME/modules; module load autotools/hwloc-vA.B" to get the Right autotools tuple.
4. Run the `$HOME/scripts/hwloc-nightly-tarball.sh` script manually (it may take several minutes; there will be no stdout because this script is normally run via cron).
5. If all goes well, you'll eventually get a mail sent to the hwloc-devel list saying that a snapshot tarball was created for your new release series.

> **NOTE:** Running the `hwloc-nightly-tarball.sh` will copy the nightly tarball to the live web site tree. The nightly tarballs are not committed to the web site git repository (because they change too fast), but they are placed in the live tree itself at the location `/l/osl/www/www.open-mpi.org/software/hwloc/nightly/vA.B/`. If the directory for your new series does not exist, **the script will create it.** This will create a problem if you have not already created the `software/hwloc/nightly/vA.B/` directory in hwloc's web tree Git repo, because when you do, running `git pull` on the live web tree will run into a conflict: Git will not create a directory that already exists. As such, if you run `hwloc-nightly-tarball.sh` before you update the live web tree with an Git-controlled `software/hwloc/nightly/vA.B/` directory, you should remove the script-created directory, create the directory in Git, run `git pull` on the live web tree (or let cron do it) and *then* run the `hwloc-nightly-tarball.sh` script.

Also remember that the `hwloc-nightly-tarball.sh` script is run nightly via mpiteam's cron on mtt.open-mpi.org. So you might want to setup the script for your new release series on the same day that you update the web site. Otherwise, the script will run that night, create the directory, and then you'll have the Git directory conflict whenever you finally get to updating the web site.

The script should create a new tarball every night whenever necessary (i.e., there have been Git commits in the relevant Git branch since the last snapshot tarball was made). As such, if you run the `hwloc-nightly-tarball.sh` script twice in a row, it is likely that nothing will happen the second time (because there will have been no new Git commits and therefore no need to make a new tarball).

Web Site

The web site requires several additions for a new release series.

You usually do not do this step until you have some kind of tarball to release for the new release series (e.g., an "rc" tarball, or some other prerelease tarball).

What do I earn?

- Many good releases don't guarantee many users
- Download stats?
  - Gforge isn't good at statistics (robots?)
  - External services? (Google Analytics, Piwik...)
- Mailing list activity is a good hint
  - Even bug reports
    - Users don't report bugs unless they care
      - Be nice to them
- Citations
  - Positive ones...

Now, go back to work!

(and thank you for your attention!)

Brice.Goglin@inria.fr

EPI Runtime - BSO