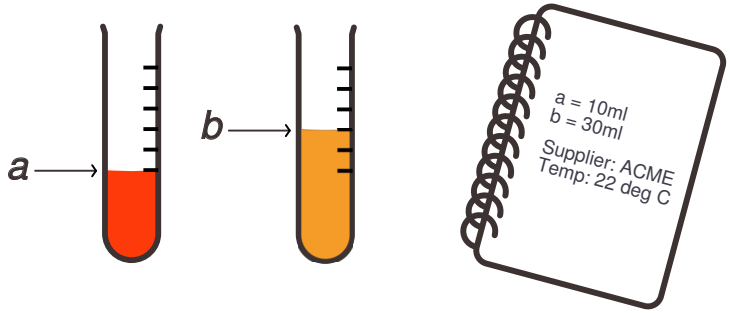
GuixHPC

Reproducible software deployment for high-performance computing.




# Activity Report 2018–2019

GuixHPC

17 February 2020
Ludovic Courtès, Paul Garlick, Konrad Hinsen, Pjotr Prins, Ricardo Wurmus

2.

19.

same vein, we are also interested in adapting Mohammad Akhlaghi's reproducible paper template[64] to take advantage of Guix.

There's a lot we can do and we'd love to hear your ideas[65]!

[64] *https://gitlab.com/makhlaghi/reproducible-paper*

[65] *https://hpc.guix.info/about*



Guix-HPC is a collaborative effort to bring reproducible software deployment to scientific workflows and high-performance computing (HPC). Guix-HPC builds upon the GNU Guix[1] software deployment tool and aims to make it a better tool for HPC practitioners and scientists concerned with reproducible research.

Guix-HPC was launched in September 2017 as a joint software development project involving three research institutes: Inria[2], the Max Delbrück Center for Molecular Medicine (MDC)[3], and the Utrecht Bioinformatics Center (UBC)[4]. GNU Guix for HPC and reproducible science has received contributions from additional individuals and organizations, including CNRS[5], Cray, Inc.[6], the University of Tennessee Health Science Center[7] (UTHSC), and Tourbillion Technology[8].

This report highlights key achievements of Guix-HPC between our previous report[9] a year ago and today, February 2020. This year was marked by a major milestone: the release in May 2019 of GNU Guix 1.0, seven years and more than 40,000 commits after its inception[10].

[1] *https://guix.gnu.org*

[2] *https://www.inria.fr/en/*

[3] *https://www.mdc-berlin.de/*

[4] *https://ubc.uu.nl/*

[5] *https://www.cnrs.fr/en*

[6] *https://www.cray.com*

[7] *https://uthsc.edu/*

[8] *http://tourbillion-technology.com/*

[9] *https://hpc.guix.info/blog/2019/02/guix-hpc-activity-report-2018/*

## Outline

Guix-HPC aims to tackle the following high-level objectives:

- *Reproducible scientific workflows.* Improve the GNU Guix tool set to better support reproducible scientific workflows and to simplify sharing and publication of software environments.
- *Cluster usage.* Streamlining Guix deployment on HPC clusters, and providing interoperability with clusters not running Guix.
- *Outreach & user support.* Reaching out to the HPC and scientific research communities and organizing training sessions.

The following sections detail work that has been carried out in each of these areas.

[10] *https://hpc.guix.info/blog/2019/05/gnu-guix-1.0-foundation-for-hpc-reproducible-science/*



## Perspectives

Making Guix more broadly usable on HPC clusters remains one of our top priorities. Features added this year to `guix pack` are one way to approach it, and we will keep looking for ways to improve it. In addition to this technical approach, we will keep working with cluster administrators to allow them to deploy Guix directly on their cluster. We have seen more cluster administrators deploy Guix this year and we are confident that this trend will continue.

Last year, we advocated for tight integration of reproducible deployment capabilities through Guix in scientific applications. The GNU Guix Workflow Language and Guix-Jupyter have since matured, giving us more insight into the benefits of the approach and opening new perspectives that we will explore. We would additionally like to investigate a complementary approach: adding Guix support to existing tools, such as `jupyter-repo2docker`[62].

For the Guix Workflow Language we will continue to explore its suitability in scheduler-less compute environments, such as ad-hoc clusters of short-lived virtual servers, that are becoming increasingly popular. We think that the properties of bit-reproducible builds and package-level granularity unlock hitherto unavailable sharing among independent parts of workflow environments to an extent that is impossible when using monolithic container images. This increase in storage and deployment efficiency is expected to result in significant cost savings when computations are offloaded to externally hosted and metered resources.

We have witnessed increasing awareness in the scientific community of the limitations of container-based tooling when it comes to building transparent and reproducible workflows. We are happy to be associated with the "Ten Years Reproducibility Challenge"[63] where we plan to demonstrate how Guix can help reproduce computational experiments. In the

[62] *https://repo2docker.readthedocs.io/en/latest/*

[63] *https://rescience.github.io/ten-years/*

## Personnel

GNU Guix is a collaborative effort, receiving contributions from more than 60 people every month—a 50% increase compared to last year. As part of Guix-HPC, participating institutions have dedicated work hours to the project, which we summarize here.

- CNRS: 0.25 person-year (Konrad Hinsen)
- Inria: 2 person-years (Ludovic Courtès, Maurice Brémond, and the contributors to the Guix-HPC channel: Florent Pruvost, Gilles Marait, Marek Felsoci, Emmanuel Agullo, Adrien Guilbaud)
- Max Delbrück Center for Molecular Medicine (MDC): 2 person-years (Ricardo Wurmus and Mădălin Ionel Patrașcu)
- Tourbillion Technology: 0.7 person-year (Paul Garlick)
- Université de Paris: 0.25 person-year (Simon Tournier)
- University of Tennessee Health Science Center (UTHSC): 0.8 person-year (Efraim Flashner and Pjotr Prins)
- Utrecht Bioinformatics Center (UBC): 1 person-year (Roel Janssen)



## Reproducible Scientific Workflows

Supporting reproducible research in general remains a major goal for Guix-HPC. The ability to *reproduce* and *inspect* computational experiments—today's lab notebooks—is key to establishing a rigorous scientific method. We believe that a prerequisite for this is the ability to reproduce and inspect the software environments of those experiments. We have made further progress to ensure Guix addresses this use case.

### Better Support for Reproducible Research

Guix has always supported reproducible computations by design, but there were two obstacles to using Guix for actually doing reproducible computations: the user interface to reproducibility features was a bit clumsy, and documentation, both practical and background, was scarce.

Supporting reproducible computations requires addressing four aspects:

1. Finding the dependencies of a computation.
2. Ensuring that there are no hidden dependencies, such as utility programs from the environment that are "just there".
3. Providing a record of the dependencies from which they can be reconstructed.
4. Reproducing a computation from such a record.

Step 1 is very situation-dependent and can therefore not be fully automatized. Step 2 is supported by `guix environment`[11], step 3 by `guix describe`[12]. Step 4 used to require a rather unintuitive form of `guix pull`[13] (whose main use case is updating Guix), but is now supported in a more

[11] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-environment.html*

[12] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-describe.html*

[13] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-pull.html*

straightforward way by `guix time-machine`[14], which provides direct access to older versions of Guix and all the packages it defines.

A post on the Guix HPC blog[15] explains how to perform the four steps of reproducible computation, and also explains how Guix ensures bit-for-bit reproducibility through comprehensive dependency tracking.

## Reproducible Deployment for Jupyter Notebooks

Jupyter Notebooks[16] have become a tool of choice for scientists willing to share and reproduce computational experiments. Yet, nothing in a notebook specifies which software packages it relies on, which puts reproducibility at risk.

Together with Pierre-Antoine Rouby as part of a four-month internship at Inria in 2018, we started work on Guix-Jupyter[17], a Guix "kernel" for Jupyter Notebook. In a nutshell, Guix-Jupyter allows notebook writers to specify the software environment the notebook depends on: the Guix packages, and the Guix commit. Furthermore, all the code in the notebook runs in an isolated environment (a "container"). This ensures that someone replaying the notebook will run it in the right environment as the author intended.

Guix-Jupyter reached its first release in October 2019[18]. Many on Jupyter fora were enthusiastic about this approach. Compared to other approaches, which revolve around building container images, Guix-Jupyter addresses the deployment problem at its root, providing a maximum level of transparency. These Jupyter notebooks are being used in bioinformatics courses by, for example, the University of Tennessee.

## The Guix Workflow Language

[14] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-time_002dmachine.html*

[15] *https://hpc.guix.info/blog/2020/01/reproducible-computations-with-guix/*

[16] *https://jupyter.org*

[17] *https://hpc.guix.info/blog/2019/02/guix-hpc-activity-report-2018/*

[18] *https://hpc.guix.info/blog/2019/10/towards-reproducible-jupyter-notebooks/*



- ARAMIS Plenary Session on Reproducibility, May 2019[56] (Ludovic Courtès)
- JCAD, Oct. 2019[57] (Ludovic Courtès)
- SciCloj Web Meeting, Jan. 2020[58] (Ludovic Courtès)
- FOSDEM, Feb. 2020[59] (Ludovic Courtès, Efraim Flashner, Pjotr Prins)

We also organised the GNU Guix Days[60], which attracted 35 Guix contributors and ran for two days before FOSDEM 2020.

## Training Sessions

The PRACE/Inria High-Performance Numerical Simulation School[61] that took place in November 2019 contained an introduction to Guix and used it throughout its hands-on sessions. A Guix training session also took place at Inria (Bordeaux) in October 2019.

[56] *https://aramis.resinfo.org/wiki/doku.php?id=pleniaires:pleniere23mai2019*

[57] *https://jcad2019.sciencesconf.org/resource/page/id/6*

[58] *https://scicloj.github.io/pages/web_meetings/*

[59] *https://fosdem.org/2020/*

[60] *https://libreplanet.org/wiki/Group:Guix/FOSDEM2020*

[61] *https://project.inria.fr/hpcschool2019/*

# Outreach and User Support

Guix-HPC is in part about "spreading the word" about our approach to reproducible software environments and how it can help further the goals of reproducible research and high-performance computing development. This section summarizes articles, talks, and training sessions given this year.

## Articles

The book *Evolutionary Genomics*[50], published in July 2019, contains a chapter entitled "Scalable Workflows and Reproducible Data Analysis for Genomics"[51], by Francesco Strozzi *et al.* that discusses workflow and deployment tools, in particular looking at the GNU Guix Workflow Language[52], the Common Workflow Language, Snakemake, as well as Docker, CONDA, and Singularity.

We have published 7 articles on the Guix-HPC blog[53] touching topics such as efficient Open MPI packaging, Guix-Jupyter, Software Heritage integration, and a hands-on tutorial using Guix for reproducible workflows and computations.

## Talks

Since last year, we gave the following talks at the following venues:

- INRA MIA Seminar, Feb. 2019[54] (Ludovic Courtès)
- IN2P3/CNRS ComputeOps Workshop, March 2019[55] (Ludovic Courtès)

[50] *https://link.springer.com/book/10.1007/978-1-4939-9074-0*

[51] *https://link.springer.com/protocol/10.1007%2F978-1-4939-9074-0_24*

[52] *https://www.guixwl.org/*

[53] *https://hpc.guix.info/blog/*

[54] *https://miat.inrae.fr/site/List_of_past_seminars*

[55] *https://indico.in2p3.fr/event/18626/*



The Guix Workflow Language[19] (or GWL), an extension of Guix for the description and execution of scientific workflows, has seen continuous improvements in the past year. The core idea remains unchanged[20]: rather than grafting software deployment onto a workflow language, extend a mature software deployment solution just enough to accomodate the needs of users and authors of scientific workflows.

User testing revealed a desire for a more familiar syntax for users of other workflow systems without compromising the benefits of embedding a domain specific language in a general purpose language, as demonstrated by Guix itself. As a result of these tests and discussions, the Guix Workflow Language now accepts workflow definitions written in a pythonesque syntax called Wisp[21] and provides about a dozen macros and procedures to simplify common tasks, such as embedding of foreign code snippets, string interpolation, file name expansion, etc. Of course, workflows can also be written in plain Scheme or even in a mix of both styles.

One of the benefits of "growing" a workflow language out of Guix is that non-trivial features implemented in Guix are readily available for co-option. For example, the GWL now uses the mature implementation of containers in Guix to provide support for evaluating processes in isolated container environments.

Work has begun to leverage the features of both `guix pack`[22] and `guix deploy`[23] to not only execute workflows on systems that share a Guix installation but also to provision remote Guix systems from scratch to run a distributed workflow without a traditional HPC scheduler. To that end, a first prototype[24] of a Guile library to manage storage and compute

[19] *https://workflows.guix.info*

[20] *https://archive.fosdem.org/2019/schedule/event/guixinfra/*

[21] *https://srfi.schemers.org/srfi-119/srfi-119.html*

[22] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-pack.html*

[23] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-deploy.html*

[24] *https://git.elephly.net/software/guile-aws.git*

resources through Amazon Web Services (AWS) has been developed, which will be integrated with the Guix Workflow Language in future releases.

You can read more about the many changes to the GWL in the release notes of version 0.2.0[25].

## Ensuring Source Code Availability

In April 2019, Software Heritage and GNU Guix announced their collaboration[26] to enable long-term reproducibility. Being able to rely on a long-term source code archive is crucial to support the use cases that matter to reproducible science: what good would it be if `guix time-machine` would fail because upstream source code vanished? Starting from beginning of 2019, Guix is able to fall back to Software Heritage[27] should upstream source code vanish.

We worked to improve coverage of the Software Heritage archive—making sure source code Guix packages refer to is archived. That led to the addition of an `archival` tool to `guix lint`[28], our helper for package developers, which instructs Software Heritage to archive source code it currently lacks, before the package even makes it in Guix itself. We helped review work carried out by NixOS developer "lewo" to further improve archive coverage[29].

## Packaging

The core package collection that comes with Guix went from 9,000 packages a year ago to more than 12,000 as of this writing. This rapid

[25] *https://lists.gnu.org/archive/html/info-gnu/2020-02/msg00011.html*

[26] *https://www.softwareheritage.org/2019/04/18/software-heritage-and-gnu-guix-join-forces-to-enable-long-term-reproducibility/*

[27] *https://hpc.guix.info/blog/2019/03/connecting-reproducible-deployment-to-a-long-term-source-code-archive/*

[28] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-lint.html*

[29] *https://forge.softwareheritage.org/D2025/new/*



ticular the addition of the `-entry-point` option to specify the default entry point, and that of a `-save-provenance` option to save provenance metadata in the container image.

## Cluster Usage

This year Guix has become the deployment tool of choice on more clusters. We are notably aware of new deployments at several academic clusters such as GriCAD[42] (France), CCIPL[43] (France), and UTHSC[44] (USA). Discussions are on-going with other academic and industrial partners who have shown interest in deploying Guix.

In order to improve the availability of binary substitutes[45] for the more than 12,000 packages defined in Guix, the Max Delbrück Center for Molecular Medicine (MDC) in Berlin (Germany) generously provided funds to purchase 30 new servers to replace a number of outdated and failing build nodes in the distributed build farm[46]. These new servers are now hosted at the MDC data center in Berlin and continuously build binaries for several of the architectures supported by Guix. The binaries are archived on a dedicated storage array and offered for download to all users of Guix.

We have further improved `guix pack`[47] to support users who wish to take advantage of Guix while deploying software on machines where Guix is not available. One noteworthy improvement is the addition of the `-RR` option, which we like to refer to as "reliably relocatable": `guix pack -RR` would create a relocatable tarball that automatically falls back to using PRoot[48] for relocation when unprivileged user namespaces are not supported[49], thereby providing a "universal" relocatable archive. The Docker and Singularity back-ends of `guix pack` have also seen improvements, in par-

[42] *https://gricad.univ-grenoble-alpes.fr/*

[43] *https://ccipl.univ-nantes.fr/*

[44] *https://uthsc.edu/*

[45] *https://guix.gnu.org/manual/en/html_node/Substitutes.html*

[46] *https://ci.guix.gnu.org*

[47] *https://guix.gnu.org/manual/devel/en/html_node/Invoking-guix-pack.html*

[48] *https://github.com/proot-me/PRoot*

[49] *https://hpc.guix.info/blog/2017/10/using-guix-without-being-root/*



growth benefits users of all application domains, notably HPC practitioners and scientists.

The message passing interface (MPI) is a key component for our HPC users and an important factor for the performance of multi-node parallel applications. We have worked on improving Open MPI support for a wide range of high-speed network devices, making sure our `openmpi` package achieves peak performance *by default* on each of them—it is all about *portable performance*. This work is described in our blog post entitled *Optimized and portable Open MPI packaging*[30]. It led to improvements in packages for the high-speed network drivers and fabrics, such as UCX, PSM, and PSM2, improvements in the Open MPI package itself, the addition of a package for the Intel MPI Benchmarks[31], and the addition of an MPICH[32] package.

Numerical simulation is one of the key activities on HPC systems. Within GNU Guix a `simulation` module has been established to gather together packages that are used in this field. Popular packages such as OpenFOAM[33] and FEniCS[34] have already been included, with FEniCS having had a recent update. The Gmsh[35] package in the `maths` module allows for sophisticated grid generation and post-processing of results. This year the FreeCAD[36] package was added to the `engineering` module. This allows for the definition of complex two-dimensional and three-dimensional geometries, often needed as the first step in the simulation process. Engineers and scientists using Guix can now conduct simulations and numerical experiments that span a spectacular range of applications. Plans for the near future include

[30] *https://hpc.guix.info/blog/2019/12/optimized-and-portable-open-mpi-packaging/*

[31] *https://software.intel.com/en-us/articles/intel-mpi-benchmarks*

[32] *https://www.mpich.org/*

[33] *https://openfoam.org/*

[34] *https://fenicsproject.org/*

[35] *http://gmsh.info/*

[36] *https://www.freecadweb.org/*

updates to Gmsh and OpenFOAM and the addition of a specialised solver for the shallow water equations.

In HPC environments typically an underlying GNU/Linux distribution is used such as Red Hat, Debian or Ubuntu. In addition user land build systems are used such as Conda which has the downside of not being reproducible because the bootstrap normally depends on the underlying distribution. Guix, however, has support for a reproducible Conda bootstrap. This means that HPC managers can support distro software installs (e.g., through `apt-get`), but in addition users get empowered to install software themselves using thousands of GNU Guix supported packages (and extra through Guix channels, see below) and thousands of Conda packages. In practice, as system administrators, we find we hardly ever have to build packages from source again and system administrators hardly get bothered by their (scientific) users.

Many other key HPC packages have been added, upgraded, or improved, including the SLURM batch scheduler, the HDF5 data management suite, the LAPACK reference linear algebra package, the Julia and Rust programming languages, the PyOpenCL Python interface to OpenCL, and many more.

Statistical and bioinformatics packages for the R programming language in particular have seen regular comprehensive upgrades, closely following updates to the popular CRAN and Bioconductor repositories. At the time of this writing Guix provides a collection of more than 1300 reproducibly built R packages, making R one of the best supported programming environments in Guix.

In addition to the packages in core Guix, we have been developing *channels*[37] providing packages that are closely related to the research work of teams at our institutes. One such example is the Guix-HPC channel[38], developed by HPC research teams at Inria, and which now contains about forty packages. Active bioinformatics channels include that of the BIMSB

[37] *https://guix.gnu.org/manual/devel/en/html_node/Channels.html*

[38] *https://gitlab.inria.fr/guix-hpc/guix-hpc/*



group at the Max Delbrück Center for Molecular Medicine (MDC)[39] (130+ packages), that of the genetics group at UMC Utrecht[40] (400+ packages), and the genomics channel by Erik Garrison[41].

[39] *https://github.com/BIMSBbioinfo/guix-bimsb*

[40] *https://github.com/UMCUGenetics/guix-additions*

[41] *https://github.com/ekg/guix-genomics*