Planet GNU

Planet GNU - https://planet.gnu.org/

Latest 20 News Articles

GNUnet News: Messenger-GTK 0.7.0

Wed, 04 May 2022 22:00:00 +0000

Messenger-GTK 0.7.0 released

We are pleased to announce the release of the Messenger-GTK application.
The application is a convergent GTK messaging application using the GNUnet Messenger service. The goal is to provide private and secure communication between any group of devices. The interface is also designed in a way to scale down to mobile and small screen devices like phones or tablets.

The application provides the following features:

- Creating direct chats and group chats
- Managing your contacts and groups
- Invite contacts to a group
- Sending text messages
- Sending voice recordings
- Sharing files privately
- Deleting messages with any custom delay
- Renaming contacts
- Exchanging contact details physically
- Verifying contact identities
- Switching between different accounts

The application utilizes the previously released library "libgnunetchat" in a convergent graphical user interface. More information about that can be found here .

It is also possible to install and try the application as flatpak. The application is already available on flathub.org . Otherwise you will find the source code ready to compile below as well.

Download links

- messenger-gtk-0.7.0.tar.gz (signature)

The GPG key used to sign is: 3D11063C10F98D14BD24D1470B0998EF86F59B6A

Note that due to mirror synchronization, not all links might be functional early after the release. For direct access try http://ftp.gnu.org/gnu/gnunet/

Noteworthy changes in 0.7.0

- The version iteration will be inherited by cadet-gtk as logical successor.
- It is possible to create direct chats and group chats via physical or virtual exchange.
- Groups and contacts can be named, left, verified or deleted.
- Existing contacts can be invited to any private or public group.
- Chats allow sending text messages, voice recordings or files.
- Messages can be deleted with a custom delete or automatically.
- Switching between different accounts can be done during runtime.

A detailed list of changes can be found in the ChangeLog .

Known Issues

- It is still difficult to get reliable chats between different devices. This might change with the upcoming changes on the GNUnet transport layer though.
- It might happen that the FS service is not connected which might stop any file upload or stall it forever.
- The webcam/camera to scan QR codes might not get picked up properly (for example it doesn't work yet with the Pinephone).
- The application might crash at times. So consider it still being in development.

In addition to this list, you may also want to consult our bug tracker at bugs.gnunet.org .

education @ Savannah: Along: an app to collect students' data for marketing purposes

Tue, 03 May 2022 20:08:26 +0000

The nonfree app *Along*, developed by a company controlled by Zuckerberg, leads students to reveal to their teacher personal information about themselves and their families. Conversations are recorded and the collected data sent to the company, which grants itself the right to sell it.

FSF Blogs: April GNU Spotlight with Amin Bandali: Twenty new GNU releases!

Tue, 03 May 2022 15:15:35 +0000

RSS 2.0 Warning: Element </description> is missing

FSF Events: LibrePlanet workshop - May 23 - Installing Ourselves into LibrePlanet

Mon, 02 May 2022 19:54:32 +0000

RSS 2.0 Warning: Element </description> is missing

www @ Savannah: Along: an app to collect students' data for marketing purposes

Sun, 01 May 2022 10:55:43 +0000

The nonfree app *Along*, developed by a company controlled by Zuckerberg, leads students to reveal to their teacher personal information about themselves and their families. Conversations are recorded and the collected data sent to the company, which grants itself the right to sell it.

www @ Savannah: New free program needed

Fri, 29 Apr 2022 05:59:44 +0000

The world urgently needs a free program that can subtract background music from a field recording.

The purpose is to prevent censorship of people's video recordings of how cops treat the public.

FSF News: FSF job opportunity: Licensing and compliance manager

Wed, 27 Apr 2022 19:06:07 +0000

The Free Software Foundation (FSF), a Massachusetts 501(c)(3) charity with a worldwide mission to protect computer user freedom, seeks a motivated and talented Boston-based individual to be our full-time licensing and compliance manager.

remotecontrol @ Savannah: Insteon finally comes clean about its sudden smart home shutdo

Sat, 23 Apr 2022 21:44:07 +0000

https://arstechnica.com/gadgets/2022/04/insteon-finally-comes-clean-about-its-sudden-smart-home-shutdown/

Another strong reason why proprietary firmware is a bad idea.

parallel @ Savannah: GNU Parallel 20220422 ('Буча') released

Sat, 23 Apr 2022 11:19:46 +0000

GNU Parallel 20220422 ('Буча') has been released. It is available for download at: lbry://@GnuParallel:4

Quote of the month:

Immensely useful which I am forever grateful that it exists.
-- AlexDragusin@ycombinator

New in this release:

- sash is no longer supported as shell.
- --retries 0 is an alias for --retries 2147483647.
- --shell-completion returns shell completion code.
- --ssh-login-file reloads every second.
- --parset is replaced with --_parset because it is only used internally.
- sem --pipe passes STDIN (standard input) to the command.
- Bug fixes and man page updates.

Get the book: GNU Parallel 2018
http://www.lulu.com/shop/ole-tange/gnu-parallel-2018/paperback/product-23558902.html

GNU Parallel - For people who live life in the parallel lane.

If you like GNU Parallel record a video testimonial: Say who you are, what you use GNU Parallel for, how it helps you, and what you like most about it. Include a command that uses GNU Parallel if you feel like it.

About GNU Parallel

GNU Parallel is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. The typical input is a list of files, a list of hosts, a list of users, a list of URLs, or a list of tables. A job can also be a command that reads from a pipe. GNU Parallel can then split the input and pipe it into commands in parallel.

If you use xargs and tee today you will find GNU Parallel very easy to use as GNU Parallel is written to have the same options as xargs. If you write loops in shell, you will find GNU Parallel may be able to replace most of the loops and make them run faster by running several jobs in parallel. GNU Parallel can even replace nested loops.

GNU Parallel makes sure output from the commands is the same output as you would get had you run the commands sequentially. This makes it possible to use output from GNU Parallel as input for other programs.

For example you can run this to convert all jpeg files into png and gif files and have a progress bar:

```
parallel --bar convert {1} {1.}.{2} ::: *.jpg ::: png gif
```

Or you can generate big, medium, and small thumbnails of all jpeg files in sub dirs:

```
find . -name '*.jpg' |
 parallel convert -geometry {2} {1} {1//}/thumb{2}_{1/} :::: - ::: 50 100 200
```

You can find more about GNU Parallel at: http://www.gnu.org/s/parallel/

You can install GNU Parallel in just 10 seconds with:

```
 $ (wget -O - pi.dk/3 || lynx -source pi.dk/3 || curl pi.dk/3/ || \
   fetch -o -  http://pi.dk/3 ) > install.sh
$ sha1sum install.sh | grep 883c667e01eed62f975ad28b6d50e22a
12345678 883c667e 01eed62f 975ad28b 6d50e22a
$ md5sum install.sh | grep cc21b4c943fd03e93ae1ae49e28573c0
cc21b4c9 43fd03e9 3ae1ae49 e28573c0
$ sha512sum install.sh | grep ec113b49a54e705f86d51e784ebced224fdff3f52
```

```
79945d9d 250b42a4 2067bb00 99da012e c113b49a 54e705f8 6d51e784 ebced224
fdff3f52 ca588d64 e75f6033 61bd543f d631f592 2f87ceb2 ab034149 6df84a35
$ bash install.sh
```

Watch the intro video on http://www.youtube.com/playlist?list=PL284C9FF2488BC6D1

Walk through the tutorial (man parallel_tutorial). Your command line will love you for it.

When using programs that use GNU Parallel to process data for publication please cite:

O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014.

If you like GNU Parallel:

- Give a demo at your local user group/team/colleagues
- Post the intro videos on Reddit/Diaspora*/forums/blogs/ Identi.ca/Google+/Twitter/Facebook/Linkedin/mailing lists
- Get the merchandise https://gnuparallel.threadless.com/designs/gnu-parallel
- Request or write a review for your favourite blog or magazine
- Request or build a package for your favourite distribution (if it is not already there)
- Invite me for your next conference

If you use programs that use GNU Parallel for research:

- Please cite GNU Parallel in you publications (use --citation)

If GNU Parallel saves you money:

- (Have your company) donate to FSF https://my.fsf.org/donate/

About GNU SQL

GNU sql aims to give a simple, unified interface for accessing databases through all the different databases' command line clients. So far the focus has been on giving a common way to specify login information (protocol, username, password, hostname, and port number), size (database and table size), and running queries.

The database is addressed using a DBURL. If commands are left out you will get that database's interactive shell.

When using GNU SQL for a publication please cite:

O. Tange (2011): GNU SQL - A Command Line Tool for Accessing Different Databases Using DBURLs, ;login: The USENIX Magazine, April 2011:29-32.

About GNU Niceload

GNU niceload slows down a program when the computer load average (or other system activity) is above a certain limit. When the limit is reached the program will be suspended for some time. If the limit is a soft limit the program will be allowed to run for short amounts of time before being suspended again. If the limit is a hard limit the program will only be allowed to run when the system is below the limit.GNU Parallel 20220422 ('Буча') has been released. It is available for download at: lbry://@GnuParallel:4

Quote of the month:

```
Immensely useful which I am forever grateful that it exists.
 -- AlexDragusin@ycombinator
```

New in this release:

- sash is no longer supported as shell.
- --retries 0 is an alias for --retries 2147483647.
- --shell-completion returns shell completion code.
- --ssh-login-file reloads every second.
- --parset is replaced with --_parset because it is only used internally.
- sem --pipe passes STDIN (standard input) to the command.
- Bug fixes and man page updates.

Get the book: GNU Parallel 2018
http://www.lulu.com/shop/ole-tange/gnu-parallel-2018/paperback/product-23558902.html

GNU Parallel - For people who live life in the parallel lane.

If you like GNU Parallel record a video testimonial: Say who you are, what you use GNU Parallel for, how it helps you, and what you like most about it. Include a command that uses GNU Parallel if you feel like it.

About GNU Parallel

GNU Parallel is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. The typical input is a list of files, a list of hosts, a list of users, a list of URLs, or a list of tables. A job can also be a command that reads from a pipe. GNU Parallel can then split the input and pipe it into commands in parallel.

If you use xargs and tee today you will find GNU Parallel very easy to use as GNU Parallel is written to have the same options as xargs. If you write loops in shell, you will find GNU Parallel may be able to replace most of the loops and make them run faster by running several jobs in parallel. GNU Parallel can even replace nested loops.

GNU Parallel makes sure output from the commands is the same output as you would get had you run the commands sequentially. This makes it possible to use output from GNU Parallel as input for other programs.

For example you can run this to convert all jpeg files into png and gif files and have a progress bar:

```
parallel --bar convert {1} {1.}.{2} ::: *.jpg ::: png gif
```

Or you can generate big, medium, and small thumbnails of all jpeg files in sub dirs:

```
find . -name '*.jpg' |
 parallel convert -geometry {2} {1} {1//}/thumb{2}_{1/} :::: - ::: 50 100 200
```

You can find more about GNU Parallel at: http://www.gnu.org/s/parallel/

You can install GNU Parallel in just 10 seconds with:

```
 $ (wget -O - pi.dk/3 || lynx -source pi.dk/3 || curl pi.dk/3/ || \
   fetch -o - http://pi.dk/3 ) > install.sh
$ sha1sum install.sh | grep 883c667e01eed62f975ad28b6d50e22a
12345678 883c667e 01eed62f 975ad28b 6d50e22a
$ md5sum install.sh | grep cc21b4c943fd03e93ae1ae49e28573c0
```

cc21b4c9 43fd03e9 3ae1ae49 e28573c0
$ sha512sum install.sh | grep ec113b49a54e705f86d51e784ebced224fdff3f52
79945d9d 250b42a4 2067bb00 99da012e c113b49a 54e705f8 6d51e784 ebced224
fdff3f52 ca588d64 e75f6033 61bd543f d631f592 2f87ceb2 ab034149 6df84a35
$ bash install.sh

Watch the intro video on http://www.youtube.com/playlist?list=PL284C9FF2488BC6D1

Walk through the tutorial (man parallel_tutorial). Your command line will love you for it.

When using programs that use GNU Parallel to process data for publication please cite:

O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014.

If you like GNU Parallel:

- Give a demo at your local user group/team/colleagues
- Post the intro videos on Reddit/Diaspora*/forums/blogs/
  Identi.ca/Google+/Twitter/Facebook/Linkedin/mailing lists
- Get the merchandise https://gnuparallel.threadless.com/designs/gnu-parallel
- Request or write a review for your favourite blog or magazine
- Request or build a package for your favourite distribution (if it is not already there)
- Invite me for your next conference

If you use programs that use GNU Parallel for research:

- Please cite GNU Parallel in you publications (use --citation)

If GNU Parallel saves you money:

- (Have your company) donate to FSF https://my.fsf.org/donate/

About GNU SQL

GNU sql aims to give a simple, unified interface for accessing databases through all the different databases' command line clients. So far the focus has been on giving a common way to specify login information (protocol, username, password, hostname, and port number), size (database and table size), and running queries.

The database is addressed using a DBURL. If commands are left out you will get that database's interactive shell.

When using GNU SQL for a publication please cite:

O. Tange (2011): GNU SQL - A Command Line Tool for Accessing Different Databases Using DBURLs, ;login: The USENIX Magazine, April 2011:29-32.

About GNU Niceload

GNU niceload slows down a program when the computer load average (or other system activity) is above a certain limit. When the limit is reached the program will be suspended for some time. If the limit is a soft limit the program will be allowed to run for short amounts of time before being suspended again. If the limit is a hard limit the program will only be allowed to run when the system is below the limit.

FSF Blogs: Find new ways to "live liberation" with LibrePlanet 2022 workshops
Thu, 21 Apr 2022 19:25:00 +0000

health @ Savannah: The Free Software community mourns the loss of Pedro Francisco (Mas

Tue, 19 Apr 2022 08:27:37 +0000

Dear friends

These are very sad days for the Free Software and Social Justice movements. Our beloved friend Pedro Francisco has passed away.
Pedro fought relentlessly for equity in our society. He was also a Free/Libre Software activist.

Pedro created and managed MasGNULinux, a Spanish blog with news about Free Software and GNU/Linux. MasGNULinux was the best reference in the latest Free Software projects for the Spanish speaking community.

Thank you for your integrity, your honesty and your dedication to make this world a better place for this and future generations. Pedro's legacy will live on forever, in every line of code of each Free Software project.

From the GNU Health community, we send our deepest condolences to his family and friends.

PS: Rumor has it that God has switched to GNU/Linux.

parted @ Savannah: parted-3.5 released [stable]

Mon, 18 Apr 2022 20:26:31 +0000

I have released parted v3.5, the only change from the previous alpha was
updating gnulib to the current version.

Here are the compressed sources and a GPG detached signature[*]:
  https://ftp.gnu.org/gnu/parted/parted-3.5.tar.xz
  https://ftp.gnu.org/gnu/parted/parted-3.5.tar.xz.sig

Use a mirror for higher download bandwidth:
  https://www.gnu.org/order/ftp.html

Here are the SHA256 checksums:

4938dd5c1c125f6c78b1f4b3e297526f18ee74aa43d45c248578b1d2470c05a2 parted-3.5.tar.xz
1b4a381f344435baf69616a985fac6f411d740de9eebd91e4cccdf046332366a parted-3.5.tar.xz.sig

[*] Use a .sig file to verify that the corresponding file (without the
.sig suffix) is intact. First, be sure to download both the .sig file
and the corresponding tarball. Then, run a command like this:

  gpg --verify parted-3.5.tar.xz.sig

If that command fails because you don't have the required public key,
or that public key has expired, try the following commands to update
or refresh it, and then rerun the 'gpg --verify' command.

  gpg --locate-external-key bcl@redhat.com

  gpg --recv-keys 117E8C168EFE3A7F

  wget -q -O- '
https://savannah.gnu.org/project/release-gpgkeys.php?group=parted&download=1' | gpg --

import -

This release was bootstrapped with the following tools:
Autoconf 2.71
Automake 1.16.5
Gettext 0.21
Gnulib v0.1-5201-g0cda5beb79
Gperf 3.1

NEWS

- Noteworthy changes in release 3.5 (2022-04-18) [stable]
- - New Features

Update to latest gnulib for 3.5 release

- Noteworthy changes in release 3.4.64.2 (2022-04-05) [alpha]
- - Bug Fixes

usage: remove the mention of "a particular partition"

- Noteworthy changes in release 3.4.64 (2022-03-30) [alpha]
- - New Features

Add --fix to --script mode to automatically fix problems like the backup GPT header not being at the end of a disk.

Add use of the swap partition flag to msdos disk labeled disks.

Allow the partition name to be an empty string when set in script mode.

Add --json command line switch to output the details of the disk as JSON.

Add support for the Linux home GUID using the linux-home flag.

- - Bug Fixes

Decrease disk sizes used in tests to make it easier to run the test suite on systems with less memory. Largest filesystem is now 267MB (fat32). The rest are only 10MB.

Add aarch64 and mips64 as valid machines for testing.

Escape colons and backslashes in the machine output. Device path, model, and partition name could all include these. They are now escaped with a backslash.

Use libdevmapper's retry remove option when the device is BUSY. This prevents libdevmapper from printing confusin output when trying to remove a busy partition.

Keep GUID specific attributes when writing the GPT header. Previously they were set to 0.

GNU Guix: 10 years of stories behind Guix

Mon, 18 Apr 2022 18:45:00 +0000

It’s been ten years today since the very first commit to what was already called Guix—the unimaginative name is a homage to Guile and Nix, which Guix started by blending together. On April 18th, 2012, there was very little to see and no actual “project”. The project formed in the following months and became a collective adventure around a shared vision.

Ten years later, it’s amazing to see what more than 600 people achieved, with 94K commits, countless hours of translation, system administration, web design work, and no less than 175 blog posts to share our enthusiasm at each major milestone. It’s been quite a ride!

What follows is a series of personal accounts by some of the contributors who offered their time and energy and made it all possible. Read their stories and perhaps you too will be inspired to join all the nice folks on this journey?

Alice Brenon

As a conclusion, Guix is a really neat system and I hope you enjoy it as much as I do!

My story with Guix is a bit topsy-turvy so I thought I might as well start by the end :) I first ranked it last among systems I wanted to test, then was a bit puzzled by it when I had the chance to test it after all the others had disappointed me, and we finally got to know each other once I installed it on my work laptop, because when you need a good, stable system you know you can rely on, why not use the most surprising one? Strangely, the alchemy worked and it has never let me down so far.

Like all good computer things, it looked way scarier from a distance than it really was, and seemed to be very much about ethics and theory while it's actually very pragmatic. I had struggled for years with the myriad of incompatible package formats for systems, then for each specific languages, and was flushed to discover at last what seemed to be a reasonable universal format. That's probably what I like best about it: the ability to use potentially any software I want without trashing my system. The welcoming community eager to help and valuing my contributions made it even better, and submitting patches came naturally. I mostly use it for development, and to keep my sanity in spite of all the data science tools I have to use for work. I sometimes wish it were easier to tweak the core of the system, but I blame my lack of free time at least as much as its design. I would absolutely love to see my Guix system using the runit init system one day but it just works and I finally learnt that it was all that mattered if you wanted to get things done in the end.

Andreas Enge

When I think of Guix, I always kid myself into believing that I had the idea — I remember chatting with Ludovic around a GNU hackers' meeting about Nix; I joked that since Guile is the GNU language, Nix should be rewritten in Guile. But it turned out that Ludovic had already started such a project in earnest... Luckily there is the git history to refresh our memories. Apparently I installed Guix, at the time only the package manager, with Ludovic's help in December 2012, and immediately reported a few bugs. My next action was to update the package for my own software. Learning Scheme was quite difficult, but I fondly remember discussing quotes and quasiquotes with Ludovic. After that, I mostly added packages to Guix, which was possible without knowing much of functional programming; the most tricky packages that stayed in my mind were ImageMagick and TeX Live. I came to appreciate the GNU Autotools — with all their shortcomings, having a uniform (and usually reliable) way of compiling and installing a software makes creating a Guix package almost trivial.

The most compelling feature of Guix was (and still is, I think) the ability to roll back package installations, and now complete system installations — no more fear of updating a package to a non-working state! And on a completely different level, the nice and welcoming atmosphere in the community, in no small part thanks to Ludovic's initial efforts of creating an inclusive environment.

Many formidable adventures are attached to working on Guix. Buying our first server for the build farm was difficult, since we wanted to use a machine with Libreboot that would work well with the GNU system. Eventually we succeeded, and it is still hosted with the non-profit Aquilenet in Bordeaux, so we managed to start our own infrastructure in accordance with our values.

Writing the bylaws of the Guix Europe non-profit was another exciting adventure; again we tried to create a structure in line with our values, where the decisions were taken as collectively as possible.

And personally I have fond memories of taking part in Guix meetings at FOSDEM and co-organising the Guix Days in Brussels; these are such nice occasions to meet people passionate about Guix and free software! I will never forget the Guix Days 2020 when I had just returned from a Reproducible Build Summit where I admired their way of facilitating the workshop, which I then tried to copy for our own meeting.

The Guix system meets all my daily needs now, so I have no technical wishes for the future — but I trust the many creative minds working on advancing the project to come up with nice new ideas. And I wish the human adventure and community building around Guix to continue!

Andrew Tropin

It's all lisp, no distraction, pure consistency! Every few years I migrate to a different workhorse and it has always been a pain to bring my software setup with me: forgot a pam/udev rule here, package here and some tiny hack here and everything is messed up, easier to reinstall everything from ground up. With declarative and reproducible nature of Guix System, Guix Home and rde project it's just a pure pleasure: write the configuration once, use it everywhere! Daemon configurations, package build phases, cron tasks, everything described in Guile Scheme. The one language to rule them all! I look forward for a little more: wider adoption of Guix for managing development environments and infrastructures.

GNU Guix respects you and your freedoms: anyone can explore and study, tweak and adjust, and share everything they want, every program available. Moreover every package is bootstrapped, what a miracle! Yes, some hardware isn't supported, but for a good reason, for this price you get a lot. I look forward for a little more: decentralized substitutes and RISC-V laptops running GNU Guix.

Thank you the community for all the hard work, already enjoy Guix for more than an year and look forward for more exciting things to appear!

Arun Isaac

I was introduced to Guix in 2016. That was around the time I was learning lisp and having my mind blown. As soon as I heard that Guix was written in a lisp and that it was a FSDG compliant distro, I knew that this was going to be the best distro ever and immediately jumped ship.

While I was immediately smitten by the perfection and elegance of Guix, I have stayed this long not for the technical excellence but for the people. The Guix community is the warmest and most caring free software community I have ever been a part of. I honestly did not believe such people could exist online until I saw Guix. In the future, I would love for this community to grow and thrive with a smoother issue tracking and contribution process so that potential contributors, especially newcomers, don't turn away frustrated.

Björn Höfling

In 2016, I searched a GNU/Linux distribution for a friend's laptop and chose GuixSD (now Guix System), inspired by a LibrePlanet video from MediaGoblin. The laptop failed to digest this OS, as it demanded binary, non-free drivers. In contrast, my wetware is 100% GNU FSDG-compatible, and so, generation 1 of Guix burned successfully into my brain, without any headaches. Or at least, they went away with the help from the very friendly, supportive, tolerant (only to the tolerant, thanks to the code of conduct) community.

My contributions started with not understanding Guix, and asking stupid questions on the mailing lists. Sometimes during that process I found bugs and analyzed them further, which helped Ludovic and other developers to fix them. I reviewed mostly Java packages and added some on my own. I very much enjoyed co-mentoring for Outreachy, and learned more on automated video/screencast generation. I should be really more active in the community again!

For the future of Guix, I would like to see more Java and Go packages (hem, this comes not from alone, I should review and contribute more). Internally I wish a more intuitive bug- and patch-tracker instead of Debbugs+ Mumi. Externally I wish a much bigger awareness in the commercial "Open Source" world about software freedom, bootstrapping your environment and dependencies from free source code in a real reproducible way, instead of relying on opaque, binary containers. I wish people would much more take care of their dependencies (with Guix, of course!), put much more thoughts in their usage of dependencies, break-up dependency hell and support third parties in building their sources with free software (instead of relying on binary dependencies and opaque containerized dev-environments).

Blake Shaw

New media artists and designers suffer from following dilemma: our work, with its primary medium being code, is at once perhaps the simplest medium to *distribute* — requiring little more than copying a directory of text files from one hard drive to another — yet the works themselves remain a total nightmare to faithfully *reproduce* across varying machines at different points in time. Among other reasons, this is because our works are often composed of disparate parts with accompanying technical debt: an audio-visual installation may use the C++ library **openFrameworks** for high-performance interactive graphics, Haskell's **TidalCycles** for realtime sequencing, the fantastic **FAUST** signal processing language for zero-delay audio DSP, plus the usual dependencies; openCV, libfreenect, cairo, gstreamer, ffmpeg and so on. Time and its corresponding ABI changes intensify our predicament; not only is it often an error-prone and laborious task to get all of this running correctly across many machines, the nature of technical debt means that getting an installation from 2014 up and running in 2022 is often more trouble than its worth. Sadly, these seemingly immaterial works of art that are the simplest to copy are simultaneously some of the most difficult to reproduce and the quickest to depreciate.

Guix, on the other hand, offers its users provenance-preserving *bit-reproducible builds* of their entire operating systems: using Guix's implementation of the functional software deployment model, I should be able to reproduce, bit-for-bit, the exact same results across equivalent hardware. Suddenly our artworks can be deterministically produced not only at the level of the source code's changelog but also as the level of the build, offering the guarantee that our usually glued-together towers of systems that power our installations can be revisited and reproduced in the future, and the deployment processes becomes as simple as packing your system into a container to be deployed to a remote target. These guarantees means that the scaling of our works become simplified: if you want to do an installation that involves 100 Raspberry Pis communicating with one another in a dynamic system, you can focus on working on just a small parameterized subset, and then generate their varying configurations using the infrastructure that Guix provides. I'm currently in the early phases of developing re::producer, a "creative plumber's toolkit" that seeks to simplify this process for artists, tailoring the tools provided by Guix to the domain-specific needs of media art and allowing artists to declaratively define complex media art systems using one of the greatest programming language of all time, Scheme.

This is still new terrain, so there is plenty of work to do. But that’s no excuse to keep up with your old habits, so roll up your sleeves and come hack the good hack!

Cyril Roelandt

Back in early 2013, I was lucky enough to be unemployed for a few months. This gave me a lot of time to try out GNU Guix. Ludovic had told me about it a few months earlier, while we were still working at the same place. It was so easy to add new packages that I naturally ended up submitting a few patches and quickly trolled the whole project by adding my editor of choice, vim. Debugging package definitions was also very simple since the builds were reproducible by default.

I also had some fun writing the first version of the linter and improving the importer/updater for Python packages. I even hacked tox to make it use Guix instead of virtualenv and gave a talk about this at FOSDEM. Even though I left the project a few years ago, I'm glad to see it's doing well and is used in science and has joined forces with Software Heritage.

Efraim Flashner

Back in 2015 or so I had been using GNU/Linux on the desktop for a number of years and I wanted to contribute somehow. I had just finished a course in University using Lisp and Prolog and then I heard about Guix having its 0.8.3 (or so) release and it looked like something that I

could try to contribute to. I certainly made a number of mistakes in the beginning; I didn't know that
`git revert`
was an actual command and I tried to revert a commit by hand, leaving a dangling parenthesis and breaking the repo. Another time I added Java as a dependency to an image library and broke the graphics stack for half the architectures until I reverted that! I even had a stint as a failed GSoC student. I was working on Bournish, a Gash/Gash-utils like utility to make debugging in the early boot process far easier by providing common CLI utilities. I had some issues with time management and ended up spending more time than I should have updating packages in the repository, as a result I didn't spend enough time working on Bournish and it's languished since then.

Currently, I enjoy working on troublesome packages and expanding the number of packages available on non-popular architectures. Sometimes it's removing compiler flags or 'ifdef gating' architecture-specific includes and other times certain parts of programs need to be disabled. Then everything needs to be double-checked for cross-compiling. Right now I'm working on riscv64-linux support in Guix, it has a lot of potential but powerful boards are hard to come by. Also there are some lingering bugs with
`guix show`
showing different supported-systems for packages depending on which architecture you run it from; on x86_64-linux only two are shown, from aarch64-linux all 9 architectures are shown.

Ekaitz Zarraga

A friend of mine introduced me to Nix and Guix a while ago but I was hesitant to try it because I hate configuring stuff and this didn't look as an easy distribution to use. Once I discovered we could have separate environments and how easy is to write a package (despite all other difficulties Guix has) I was completely into it. I installed Guix in my laptop and never looked back. In the 10 years I've been using a GNU/Linux distribution I never interacted that directly with my packages: creating custom ones, sending them upstream, making fixes... That's also freedom! Now, a couple of years later, I am working on improving the bootstrap process for RISC-V and using Guix as a mechanism that provides reproducible builds and an easy way to manage all the actors I have to deal with: very old software with old dependencies, colliding libraries, environment variables, custom patches in source code... This would be a pain to build in any other environment, but Guix makes hard things easy. Guix also makes easy things hard sometimes, but we are working on that!

Eric Bavier

As a young(ish) computer programmer, I had been running GNU/Linux systems for about 7 years but wanted to find a project I could contribute back to. Fortunately, I came upon a release announcement for Guix after having poked around the GNU Hurd and Guile spheres. To me at the time Guix had the exact mix of upstart energy, optimism, and long-term vision that I was hoping to find. Over the years I've been able to contribute packages I use in both my personal and work lives, and I'm proud to have implemented the first version of `guix refresh --list-dependents`. I've really loved how Guix allows me to easily move my environments around to different systems, and "rollback" gives me much peace of mind knowing that I can tinker with the system and recover should something go wrong. But probably my favorite part of Guix is the fantastic community I've seen grow around the project. It exemplifies the sort of caring, kind, supportive group I wish many other projects had. Together I know we'll be able to make advances on many fronts. In particular, I'd like to see further work on peer-to-peer substitutes delivery, a native

build daemon, additional tools for managing relocatable pack collections, and continued leadership in bootstrapping.

Florian Pelz (pelzflorian)

GNU Guix to me is a group that cares about its software and about the people involved. I've got to know Guix by reading discussions on how to do sandboxing properly. But actually, Guix convinced me with its clarity and approachability and principles. Guix opened my eyes on how parts of GNU fit together. Thanks to all who give us this freedom to understand and decide. By contributing to the translation, I hope to make it reach more users and developers.

Guillaume Le Vaillant

Before I started using Guix in 2019, I was using Gentoo because I liked how I could easily package software and make package variants with some custom patches. However one day an upgrade didn't go well and many packages ended in a bad state. I realized I would have to reinstall the whole system to get things to work again. Before recompiling the whole system, I tried Nix and Guix, because I had read somewhere that they used functional package management, which gives the possibility to roll back to a working state when an upgrade causes problems. I chose Guix because I thought it was going in the right direction by using only free software and trying to get reproducible builds. The fact that package definitions use the Scheme language was a bonus point as I like Lisp languages. And there was even a build system for Common Lisp packages, which is rarely the case in the GNU/Linux distributions I tried over time. So I started using Guix, and packaging software I wanted that were not in Guix yet. One day someone asked me if I would be interested in having commit access, and I accepted. I also found a way to improve the build system for Common Lisp packages that simplified package definitions. In the future, I think it would be nice to add an importer fetching information from Quicklisp, as it would make packaging Common Lisp software even easier.

Hartmut Goebel

Christian Grothoff (GNU Taler) pointed me to Guix early 2016, saying "This will become the new Debian!" and asking me to look at it for GNU Taler. Well, quickly I was attracted by the ideas of reproducible build and the ease of packaging software. I also love the one-time usage of programs without littering my system.

Curiously, even as I'm a Python developer, my first contributions have been about Java packaging. And I spend quite some time trying to build maven. This challenge I gave up after two (or three? can't remember) attempts. Glad Julien Lepiller continued the endeavor and created the Maven build system.

Nowadays I still use Guix on a foreign distro only, as KDE desktop and some of my main applications are still not here. Guix keeps my main system tidy, while I can have different development environments without dependency conflicts.

As you can imagine, I'd like to see KDE desktop in Guix as well as some
`guix compose`
for managing compound containers.

Jan (janneke) Nieuwenhuizen

At FOSDEM 2016 there were seven talks about GNU Guix: A talk about the Hurd by Manolis Ragkousis, about functional package management by Ricardo Wurmus and that was just what I needed to hear: Finally a viable promise for the GNU System and much more innovative than I could have hoped for. At the time I also worked on a project where building binary releases was

becoming more unmanageable with every release because of conflicting requirements. We were slowly migrating away from C++ to GNU Guile, so while not directly applicable the talk "Your distro is a Scheme library" by Ludovic Courtès also made me feel: Go Guix!

Using Guix, my intricate dependency problems building binary packages quickly and easily disappeared. That gave me the confidence that I needed and I wanted to get involved. My first contributions where a programming game called Laby and its dependencies and a few more packages that I missed. After running Guix on top of Debian GNU/Linux for three months I switched to what we now call Guix System. Guix did not have log rotation yet in those days, so I created a package.

This is how I found how amazingly helpful and friendly the community was. I created the MinGW cross build for Guile 2.0 and then "found out" about the bootstrap binaries: The only packages in Guix that are not built from source. Something just did not feel right. The manual said: *"These big chunks of binary code are practically non-auditable which breaks the source to binary transparency that we get in the rest of the package dependency graph."* So, I wrote GNU Mes and started working on solving this problem. Twice we halved the size of the bootstrap binaries and the work is still ongoing.

What possibly started somewhat as a April fools joke in 2020 about the Hurd—this is still unclear—was (mis?)taken by some as a real project and led to a fun hacking frenzy of several months finally producing the "Childhurd": A Guix Shepherd service that gives access to the GNU/Hurd in a VM. My wish for the near future would be see an up-to-date Hurd including the Debian rumpkernel patches that may finally enable running the Hurd on real hardware again.

John Kehayias

All I wanted to do was to try out a new status bar, but the author only officially supported Nix for building. That started me to finally look at Nix after hearing about it in passing before. I was intrigued by the focus on reproducible and declarative builds. The language, not so much. My brother mentioned another project in the same vein but built on a Lisp. As a lover of all things Lisp, that was basically enough for me to dive right in. Beyond the banner features of the powerful package and system management, reproducible builds, system configuration, and, of course, Guile, I quickly found perhaps the biggest and most important: the GNU Guix community. They have been nothing short of amazing: helpful, intelligent, supportive, and fun to hang out with on the
`#guix`
channel. In less than a year, my journey so far has taken me through the (is it infamous yet?) recent big core-updates branch and merge, submitting patches for random libraries and key desktop features I use, and participating in the motivating Guix Days 2022. Looking to the future, I hope we can better harness the energy and resources of the growing Guix community. It is already a great project to get involved with and make your own, but with better and quicker patch review, further building out and using our development tools and ecosystem, and continuing to smooth out the rough edges for new users/contributors, I'm sure the next 10 years of GNU Guix will be very bright indeed.

Konrad Hinsen

In my work as a computational scientist, my first encounter with reproducibility issues happened in 1995, when a quantum chemistry package produced different results on two almost identical Silicon Graphics workstations. This was the beginning of a long quest for better computational reproducibility, in the course of which I discovered in 2014 Nix and Guix as two implementations of the same promising idea: the fully automated construction of a complete reproducible software stack. Of the two, Guix was more aligned with my lispy past, and already had a burgeoning computational science user community. I started playing with Guix in 2016, in

a virtual machine under macOS, but only fully adopted Guix for my everyday work in 2021, when I left macOS for Linux. During those five years, I also learned to appreciate the Guix community, which is friendly, competent, and refreshingly low-ceremony in spite of continuous growth. That makes for an easy transition from newbie to contributor (mostly contributing packages, but also the time-machine command that matters for reproducibility). The anniversary is a good occasion to express my thanks to all those who answered my many questions, ranging from conceptual to technical, and to the maintainer team that does an important but not very visible work by critically examining all submitted packages and code enhancements. My main wish for the future is a lower barrier to adoption for my colleagues in computational science, and I hope to contribute to making this happen.

Lars-Dominik Braun

Around the end of 2019 we were looking for a way to provide reproducible software environments to researchers in(?) psychology and I was researching software to accomplish that. Binder/repo2docker was the obvious and most mature solution at that time and a colleague of mine had set up a proof of concept server already. But it could only handle public projects out-of-the-box and setting up an entire Kubernetes cluster didn't seem particularly appealing at that time, because no other project was moving in that direction yet. So I set out to look for alternatives. Another idea was based around OpenStack and one virtual machine per project with preinstalled software, which we would keep for eternity. Also not ideal and OpenStack is very hard to master too. So I looked further at Nix, which – at that time – lacked an obvious way to spawn ad-hoc environments with a certain set of packages. Thankfully I stumbled upon GNU Guix by mere accident, which had exactly that feature. And so in December 2019 my first code contribution was merged.

Prior to that I had never written a single line of Scheme or Lisp and even now it's still a steep hill. GNU Guix still powers our project and allows us to easily share software environments while providing excellent application startup times. I also started contributing software that I run on my own machines, but I'm not running Guix System, because compared to systemd, Shepherd is quite limited on the desktop and Guix' lack of first-class support for non-free drivers/firmware, which I need to even boot my machine.

Ludovic Courtès

It all started as a geeky itch-scratching experiment: a tiny bit of Guile code to make remote procedure calls (RPCs) to the Nix build daemon. Why? As I was involved in and excited about Guile and Nix, it felt natural to try and bridge them. Guile had just had its 2.0.0 release, which broadened its scope, and I wanted to take advantage of it. Whether to go beyond the mere experiment is a decision I made sometime after a presentation at the 2012 GNU Hackers Meeting.

It was far from obvious that this would lead us anywhere—did the world *really* need another package manager? The decisive turn of event, for me, was to see that, at the time Guix officially became part of GNU in November 2012, it had already become a group effort; there was, it seems, a shared vision of why such a crazy-looking project made sense not just technically but also socially—for GNU, for user freedom. I remember Nikita Karetnikov as the first heroic contributor at a time when Guix could barely install packages.

One of my "ah ha!" moments was when I built the first bootable image a year later. G-expressions, the service framework, and checkout authentication are among my favorite hacks. What's mind-blowing to me though is what others have achieved over the years: the incredible bootstrapping work, Disarchive, Emacs-Guix, the installer, Hurd support, Guix Home, supporting tools like Cuirass, the Data Service, and mumi. There's also the less visible but crucial work: Outreachy and GSoC mentoring, support on IRC and the mailing lists,

build farm administration, translation, dealing with the occasional incident on communication channels, organizing events such as the Guix Days or FOSDEM, and more.

As much as I love *hacking the good hack*, I think Guix's main asset is its community: a friendly, productive, and creative group with a sense of attention to the other. I started clueless about what it means "to build a community" and learned a lot from everyone met on the way. We did it, *we built this*! Thumbs up, Guix!

Luis Felipe

When I found that Guix existed, I saw it could make it easier for GNU to release its Operating System and reach a wider audience. I intended to propose some graphic designs related to this, and sent a message to GNU in order to test the waters. Things didn't go as I expected, so, instead, I decided to direct my contributions towards GNU Guix and its distribution of GNU.

Since then, I've contributed with graphics (Guix and Guile logos and website designs, desktop backgrounds, release and promotional artwork), testing, bug reporting, packaging, and Spanish translations.

It's been about 8 years of Guix for me (the heck!). I started using the package manager on Debian, gradually switched the provenance of my software from Debian to Guix, and, once GNOME became available, I moved to Guix's distribution of the GNU operating system, which I've been using as my main system for about 3 years now (and I don't see that changing anytime soon).

Right now, I'm enjoying developing software using Guix's reproducible environments and containers, and using one single package manager for every dependency.

I hope this system reaches a wider audience and brings science to home computing along the way. Homes should be capable of producing scientific work too.

Manolis Ragkousis

When I think how I started with Guix, I use one word to describe it, luck! It was early 2014 when I encountered Guix by luck, while I was still a student at Crete, Greece. I remember there was a strike during that time and I had plenty of free time for a week, so I decided that I will try to start working on this. Then an idea came in mind, why not try porting Guix to GNU/Hurd and build a system with it? One thing led to another and it also became a GSoC project in 2015 and 2016. In 2016 I also gave a FOSDEM talk about this, which somehow ended up being the start of me helping out with the GNU Guile devroom in 2017 and 2018, and then what became the Minimalistic Languages until today. When I am thinking about Guix is like thinking about the story of me growing up and the people I met through all these years I consider family! Guix is a big part of my life, I use it everywhere and even though I am not able to help much nowadays I am following the project as much as I can. Here's hoping to another 10 year!

Marius Bakke

I originally got interested in Guix after facing shortcomings in traditional configuration management tools. A fully declarative and immutable system which cleans out old user accounts and packages, that also offers reproducibility, rollbacks, and the ability to generate virtual machines and containers from the same code. Where do I sign up?

It turns out, signing up was easy, and I soon found myself contributing the pieces I needed to make it a daily driver. Watching the community grow from a handful of contributors to 100 monthly has been astonishing. I have learned a lot from this community and am proud to be a part of it. Can't wait to see what the next decade brings. Happy birthday Guix!

Mathieu Othacehe

I was introduced to GNU Guix by a colleague, Clément Lassieur in 2016. At first I found the concept overwhelming. Writing Guile wrappers for each and every Linux service out there and keeping them up to date seemed like impossible. However, I quickly fell in love with the package manager, the distribution and the community behind. A few months later, GNU Guix was running on all my machines and I started hacking on the continuous integration tool: Cuirass.

Since then GNU Guix has been an important part of my life. I wrote most of the Guix System installer while traveling by bike to China in 2018. During the 2020 lockdown, I worked with janneke on the new image API and the Hurd port. At that time, I was proposed a co-maintainer position of the project. In 2021, thanks to an NGI sponsorship, I dedicated 6 months to improving our continuous integration process and overall substitutes coverage.

Recently it has been harder to dedicate as much efforts on the project but I'm sure this is a transient phase. I can't wait to start working again with the incredibly talented people making this piece of software so special to me.

Paul Garlick

I began using and contributing to the Guix project in 2016. I had been searching for a way to preserve software environments that are used for numerical simulation. The applications that run in these environments often comprise a combination of specialised code and building blocks drawn from an underlying framework. There are many moving parts and changes to a low-level library can block the operation of the high- level application. How much better things would be if one could specify the exact components of the environment and re-create it whenever it is needed. I discovered that Guix provides the machinery to do just that. Scheme was new to me then so I had some learning to do before contributing. This included a detour via Vonnegut/Cat's Cradle, of course, to discover the meaning of ice-9. Suitably informed I returned to add a number of finite volume and finite element frameworks to the Guix package collection. Keeping these packages up- to-date and welcoming new simulation-related packages is the next target. Looking ahead to the next ten years, an important task is to popularise the use of the Guix tools. Many more engineers and scientists stand to benefit from the use of the dependable software environments that are now made possible.

Ricardo Wurmus

In 2014 I became responsible for building and installing scientific software at the Max Delbrück Centre, a research institute in Berlin. We used CentOS, so I built custom RPMs, installing applications to ad-hoc prefix directories. After a few weeks I took a minute to consider the horrific implications of maintaining a growing collection of custom software with RPM. As I tried to remember what life choices had led me to this moment, I recalled an announcement email of a quirky GNU package manager written in Scheme. A short web search later I was playing around with Guix.

After an encouraging chat on IRC I realized that I could probably replace our custom RPM repository and build different variants of scientific software on much more solid ground—all the while contributing to a project that felt like a new and exciting take on GNU. We're building the GNU system!

Guix only had very few of the packages I needed, so I got busy. I packaged and bootstrapped the JDK because I was under the mistaken assumption that I would need it for R (turns out Java is optional). Many more foolish adventures followed, and some of them have actually been useful for others.

I had found my tribe of fellow hackers who cared about the vision of the GNU system, encouraged playful experimentation, and were rooting for each other to succeed in building a better system that made software freedom a practical reality, blurring the lines between developers and users. In the decades to come I hope many more people will get to experience what I did and end up calling this community their home.

Simon Tournier

Back in 2014, I watched the video "Growing a GNU with Guix" at FOSDEM but the real revelation had been in 2015 with "GNU Guix: The Emacs of Distros", again at FOSDEM. Then, I was following the development but not using Guix yet. 2016, new job where I was spending my time to fight against dependencies and Modulefiles. Then I have totally jumped in Guix in December 2018. My first interaction with the project — and not yet running Guix — was a in-person event in Paris before the Reproducible Builds workshop. Back to home, I proofread cover to cover the French manual — my first contribution — and installed Guix on the top of my Debian GNU/Linux system. So amazing! Guix fixes many issues I had at work — and introduce new ones^W challenges. Plus, thanks to people around, I am learning a lot, both about technical details and about inter-personal interactions. My wish for the near future is a community more structured: more events and meetups, more process for smoothing the contributions ("teams" for improving the reviewing by sharing the load, RFC for discussing new features, regular releases, etc.), and more materials for using Guix in various configurations.

In scientific context, transparency — being able to audit the whole computational environment from the source codes to the production of binaries — is one of the keys for a true reproducible research. Since Guix is transparent by design, it appears to me one part for a solution in tackling the computational side of the replication crisis. For the near future, I wish more scientific practitioners will employ Guix.

Thiago Jung Bauermann

I learned about Guix when I was looking for alternative, safe ways of installing an up-to-date Rust toolchain on my machine (at the time rustup didn't verify signatures of downloaded binaries, and it still doesn't do the full job). Guix is a great way to have the latest and greatest software on top of your slower-moving Linux distribution. I love how easy it makes to create instant, ad hoc environments with the packages you need for a specific task. Or to temporarily try out some new app or tool, leaving Guix to garbage-collect it and its dependencies. The Guix community is amazing as well! It's a pleasure to participate on the mailing lists. And I've been enjoying learning Scheme! For the future, I hope Guix can get even better test coverage so that every update of the master branch is guaranteed to not introduce regressions. And that the project gets more committers, to help with the constant influx of patches.

raingloom

There are multiple reasons I started using Guix. On the tech side, I'd been playing around with 9front for a while at that time, but kept running into issues where the imperative structure of namespaces was getting in my way. I like Haskell a lot and heard about the many benefits of a pure functional approach to build systems, et cetera. I ran Guix on top of Arch for a while and liked it a lot. Using package transformations still feels magical. But yall already know about this cool stuff from Ambrevar's blog post. On the social side, I saw that one of my favorite compsci people — Christine Lemmer Webber — was involved with the project, so I knew it probably has a nice community, which turned out to be very true. This is one of the best communities centered around a piece of tech that I've been in and yall inspire me to be better with each interaction. Huge thank you for that. My favorite Guix memory is when someone CC'd me in a patch for the egg importer, which was built on top of the Chicken Scheme build system I contributed. Seeing

others build on top of my work is an amazing feeling. For the future, I hope the service management improvements will keep coming, but what I'd like to see the most is Guix running on old and slow devices. There is a lot of work to be done to make it more bandwidth and space efficient and to support development on systems with little RAM. If I could use it instead of PostmarketOS/Alpine, I'd be elated. On the human side, I hope we can keep contributors from burnout, while increasing the software's quality. I think the way Blender development is structured could be a source of inspiration. On that note, using Guix for reproducible art workflows would be rad. Okay that's it, bye yall lovely people.

Vagrant Cascadian

I think I first heard of Guix in 2016, triggering a late-night session trying to wrap my head around the crazy symlink farms at the heart of Guix. By late 2017 I was filing bug reports and eventually patches!

I am deeply fascinated that Guix has Reproducible Builds built right in, with normalized, containerized build environments and the "guix challenge" tool to verify reproducibility. I had heard of Nix as an interesting model, but valued the strong commitment to Free Software with Guix.

Eventually I even grew crazy enough to package Guix in Debian... which indirectly lead to one of my most creative contributions to a Free Software project, a typo poem embedded in!

I really appreciate the community around Guix and the process, values and thoughtfulness that work proactively to maintain a healthy community, even in the face of inevitable and occasional conflict. Guix balances formal and informal in a way that works for me.

I look forward to the day when Guix has a full source bootstrap!

> 10 Years of Guix artwork by Luis Felipe.

About GNU Guix

GNU Guix is a transactional package manager and an advanced distribution of the GNU system that respects user freedom. Guix can be used on top of any system running the Hurd or the Linux kernel, or it can be used as a standalone operating system distribution for i686, x86_64, ARMv7, AArch64 and POWER9 machines.

In addition to standard package management features, Guix supports transactional upgrades and roll-backs, unprivileged package management, per-user profiles, and garbage collection. When used as a standalone GNU/Linux distribution, Guix offers a declarative, stateless approach to operating system configuration management. Guix is highly customizable and hackable through Guile programming interfaces and extensions to the Scheme language.

---

unifont @ Savannah: Unifont 14.0.03 Released

Sun, 17 Apr 2022 20:13:14 +0000

17 April 2022 Unifont 14.0.03 is now available. This release adds the new **hex2otf** program, which can convert Unifont .hex format files into OpenType fonts, as well as TrueType and other formats. See the hex2otf documentation for details.

The font files just add several new Under ConScript Unicode Registry (UCSUR) scripts: Xaîni (U+E2D0..U+E2FF), Ophidian (U+E5E0..U+E5FF), Niji (U+ED40..U+ED5F), Sitelen Pona (U+F1900..U+F19FF), and Shidann (U+F1B00..U+F1C3F).

Download this release from GNU server mirrors at:

https://ftpmirror.gnu.org/unifont/unifont-14.0.03/

or if that fails,

https://ftp.gnu.org/gnu/unifont/unifont-14.0.03/

or, as a last resort,

ftp://ftp.gnu.org/gnu/unifont/unifont-14.0.03/

These files are also available on the unifoundry.com website:

https://unifoundry.com/pub/unifont/unifont-14.0.03/

Font files are in the subdirectory

https://unifoundry.com/pub/unifont/unifont-14.0.03/font-builds/

A more detailed description of font changes is available at

https://unifoundry.com/unifont/index.html

and of utility program changes at

http://unifoundry.com/unifont/unifont-utilities.html

---

mailutils @ Savannah: Version 3.15

Sun, 17 Apr 2022 19:22:59 +0000

Version 3.15 is released today. New in this version:

- mbox format: don't count terminating empty line as part of the message
- Improve performance of the Sieve fileinto action
- Improve efficiency of operations on flat mailboxes in append mode
- Bugfixes in quoted-printable and fromrd filters
- Variois fixes in mbox and dotmail format libraries
- Fix compilation with flex version 2.6.1

---

coreutils @ Savannah: coreutils-9.1 released [stable]

Fri, 15 Apr 2022 22:34:37 +0000

This is to announce coreutils-9.1, a stable release.
See the NEWS below for details.

Thanks to everyone who has contributed!
There have been 210 commits by 10 people in the 29 weeks since 9.0

Bernhard Voelker (3)	Max Filippov (1)
Bruno Haible (1)	Paul Eggert (136)
Christian Hesse (1)	Pádraig Brady (64)
Daniel Knittl-Frank (1)	Rohan Sable (1)
Jim Meyering (4)	Ville Skyttä (1)

Pádraig [on behalf of the coreutils maintainers]

==========

Here is the GNU coreutils home page:
  https://gnu.org/software/coreutils/

For a summary of changes and contributors, see:
  https://git.sv.gnu.org/gitweb/?p=coreutils.git;a=shortlog;h=v9.1
or run this command from a git-cloned coreutils directory:
  git shortlog v9.0..v9.1

To summarize the 259 gnulib-related changes, run these commands
from a git-cloned coreutils directory:
  git checkout v9.1
  git submodule summary v9.0

==========

Here are the compressed sources:
  https://ftp.gnu.org/gnu/coreutils/coreutils-9.1.tar.gz   (14MB)
  https://ftp.gnu.org/gnu/coreutils/coreutils-9.1.tar.xz   (5.5MB)

Here are the GPG detached signatures[*]:
  https://ftp.gnu.org/gnu/coreutils/coreutils-9.1.tar.gz.sig
  https://ftp.gnu.org/gnu/coreutils/coreutils-9.1.tar.xz.sig

Use a mirror for higher download bandwidth:
  https://www.gnu.org/order/ftp.html

Here are the SHA1 and SHA256 checksums:

cab498ddc655fd3c7da553d80436d28bc9b17283 coreutils-9.1.tar.gz
YFXfkmhgPoI5pcnB1kyyW5qZJTDfZuM7jXimYO2zezU coreutils-9.1.tar.gz
aa7bf0be95eef29d98eb5c76d4455698b3b705b3 coreutils-9.1.tar.xz
YaH0ENeLp+fzelpPUObRMgrKMzdUhKMlXt3xejhYBCM coreutils-9.1.tar.xz

The SHA256 checksum is base64 encoded, instead of the
hexadecimal encoding that most checksum tools default to.

[*] Use a .sig file to verify that the corresponding file (without the
.sig suffix) is intact. First, be sure to download both the .sig file
and the corresponding tarball. Then, run a command like this:

  gpg --verify coreutils-9.1.tar.gz.sig

If that command fails because you don't have the required public key,
or that public key has expired, try the following commands to update
or refresh it, and then rerun the 'gpg --verify' command.

  gpg --locate-external-key P@draigBrady.com

  gpg --recv-keys DF6FD971306037D9

```
  wget -q -O- 'https://savannah.gnu.org/project/release-gpgkeys.php?
group=coreutils&download=1' | gpg --import -

This release was bootstrapped with the following tools:
  Autoconf 2.71
  Automake 1.16.4
  Gnulib v0.1-5194-g58c597d13
  Bison 3.7.4

NEWS

* Noteworthy changes in release 9.1 (2022-04-15) [stable]

** Bug fixes

  chmod -R no longer exits with error status when encountering symlinks.
  All files would be processed correctly, but the exit status was incorrect.
  [bug introduced in coreutils-9.0]

  If 'cp -Z A B' checks B's status and some other process then removes B,
  cp no longer creates B with a too-generous SELinux security context
  before adjusting it to the correct value.
  [bug introduced in coreutils-8.17]

  'cp --preserve=ownership A B' no longer ignores the umask when creating B.
  Also, 'cp --preserve-xattr A B' is less likely to temporarily chmod u+w B.
  [bug introduced in coreutils-6.7]

  On macOS, 'cp A B' no longer miscopies when A is in an APFS file system
  and B is in some other file system.
  [bug introduced in coreutils-9.0]

  On macOS, fmt no longer corrupts multi-byte characters
  by misdetecting their component bytes as spaces.
  [This bug was present in "the beginning".]

  'id xyz' now uses the name 'xyz' to determine groups, instead of xyz's uid.
  [bug introduced in coreutils-8.22]

  'ls -v' and 'sort -V' no longer mishandle corner cases like "a..a" vs "a.+"
  or lines containing NULs.  Their behavior now matches the documentation
  for file names like ".m4" that consist entirely of an extension,
  and the documentation has been clarified for unusual cases.
  [bug introduced in coreutils-7.0]

  On macOS, 'mv A B' no longer fails with "Operation not supported"
  when A and B are in the same tmpfs file system.
  [bug introduced in coreutils-9.0]

  'mv -T --backup=numbered A B/' no longer miscalculates the backup number
  for B when A is a directory, possibly inflooping.
  [bug introduced in coreutils-6.3]

** Changes in behavior
```

cat now uses the copy_file_range syscall if available, when doing
simple copies between regular files. This may be more efficient, by avoiding
user space copies, and possibly employing copy offloading or reflinking.

chown and chroot now warn about usages like "chown root.root f",
which have the nonstandard and long-obsolete "." separator that
causes problems on platforms where user names contain ".".
Applications should use ":" instead of ".".

cksum no longer allows abbreviated algorithm names,
so that forward compatibility and robustness is improved.

date +'%-N' now suppresses excess trailing digits, instead of always
padding them with zeros to 9 digits. It uses clock_getres and
clock_gettime to infer the clock resolution.

dd conv=fsync now synchronizes output even after a write error,
and similarly for dd conv=fdatasync.

dd now counts bytes instead of blocks if a block count ends in "B".
For example, 'dd count=100KiB' now copies 100 KiB of data, not
102,400 blocks of data. The flags count_bytes, skip_bytes and
seek_bytes are therefore obsolescent and are no longer documented,
though they still work.

ls no longer colors files with capabilities by default, as file-based
capabilties are very rarely used, and lookup increases processing per file by
about 30%. It's best to use getcap [-r] to identify files with capabilities.

ls no longer tries to automount files, reverting to the behavior
before the statx() call was introduced in coreutils-8.32.

stat no longer tries to automount files by default, reverting to the
behavior before the statx() call was introduced in coreutils-8.32.
Only `stat --cached=never` will continue to automount files.

timeout --foreground --kill-after=... will now exit with status 137
if the kill signal was sent, which is consistent with the behavior
when the --foreground option is not specified. This allows users to
distinguish if the command was more forcefully terminated.

** New Features

dd now supports the aliases iseek=N for skip=N, and oseek=N for seek=N,
like FreeBSD and other operating systems.

dircolors takes a new --print-ls-colors option to display LS_COLORS
entries, on separate lines, colored according to the entry color code.

dircolors will now also match COLORTERM in addition to TERM environment
variables. The default config will apply colors with any COLORTERM set.

** Improvements

cp, mv, and install now use openat-like syscalls when copying to a directory.
This avoids some race conditions and should be more efficient.

On macOS, cp creates a copy-on-write clone if source and destination
are regular files on the same APFS file system, the destination does
not already exist, and cp is preserving mode and timestamps (e.g.,
'cp -p', 'cp -a').

The new 'date' option --resolution outputs the timestamp resolution.

With conv=fdatasync or conv=fsync, dd status=progress now reports
any extra final progress just before synchronizing output data,
since synchronizing can take a long time.

printf now supports printing the numeric value of multi-byte characters.

sort --debug now diagnoses issues with --field-separator characters
that conflict with characters possibly used in numbers.

'tail -f file | filter' now exits on Solaris when filter exits.

root invoked coreutils, that are built and run in single binary mode,
now adjust /proc/$pid/cmdline to be more specific to the utility
being run, rather than using the general "coreutils" binary name.

** Build-related

AIX builds no longer fail because some library functions are not found.
[bug introduced in coreutils-8.32]

FSF Events: LibrePlanet workshop - May 16 - Software localization (translation) of Web-b

Wed, 13 Apr 2022 17:45:00 +0000

RSS 2.0 Warning: Element </description> is missing

FSF Events: LibrePlanet workshop - May 09 - IRCNow: Of the users, by the users, for the u

Wed, 13 Apr 2022 17:45:00 +0000

RSS 2.0 Warning: Element </description> is missing

FSF Events: LibrePlanet workshop - May 02 - Translators and free software, a practical int

Wed, 13 Apr 2022 17:45:00 +0000

RSS 2.0 Warning: Element </description> is missing

FSF Events: LibrePlanet workshop - RESCHEDULED: May 23 - Installing Ourselves into L

Wed, 13 Apr 2022 17:45:00 +0000

RSS 2.0 Warning: Element </description> is missing