Reproducible computational environment, when?

How to redeploy later and overthere
what had be deployed today and here?

Simon Tournier

Inserm US53 - UAR CNRS 2030
simon.tournier@inserm.fr

November 26th, 2024
https://hpc.guix.info

Replication and reproducibility crisis

More than 70% of researchers have tried and **failed to reproduce** another scientist's experiments, and more than half have failed to reproduce their own experiments.

1,500 scientists lift the lid on reproducibility (Nature, 2016) (link)

Many causes... one solution?
at least, *Open Science* helps

$$\begin{pmatrix} \text{reproductibility} & = & \text{verification} \\ \text{replicability} & = & \text{validation} \end{pmatrix}$$

1905: *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen*
by A. Einstein

- Only one author, verbal reasoning
- Motivated students are able to check by themselves that all the computations are correct

1905: *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen*
by A. Einstein

- Only one author, verbal reasoning
- Motivated students are able to check by themselves that all the computations are correct

2022: *Evolutionary-scale prediction of atomic level protein structure with a language model*
by Z. Zin & al.

- 15 authors, references to software
- "[...] we scale language models from 8 million parameters up to ***15 billion*** **parameters**."
- Code and data seems available... but ~~impossible~~^W hard to check that all is correct

1905: *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen*
by A. Einstein

- Only one author, verbal reasoning
- Motivated students are able to check by themselves that all the computations are correct

2022: *Evolutionary-scale prediction of atomic level protein structure with a language model*
by Z. Zin & al.

- 15 authors, references to software
- "[...] we scale language models from 8 million parameters up to ***15 billion*** **parameters**."
- Code and data seems available... but ~~impossible~~^W hard to check that all is correct

Among several questions*, scientific research is evolving,

what does it mean *scientific research* now?

**is 15 billion parameters explanatory?*

~~Open Science~~ Reproducible Research

Science = Transparent and Collective
Scientific result = Experiment + Numerical treatment

Science at the digital age:

1. Open Article — HAL, BioArxiv
2. Open Data — Data Repositories, Zenodo
3. Open Source — Forges, GitLab, Software Heritage

open science, a tautology?

~~Open Science~~ Reproducible Research

Science = Transparent and Collective

Scientific result = Experiment + *Numerical treatment*

Science at the digital age:

1. Open Article — HAL, BioArxiv
2. Open Data — Data Repositories, Zenodo
3. Open Source — Forges, GitLab, Software Heritage

how to *glue* all that?

open science, a tautology?

~~Open Science~~ Reproducible Research

Science = Transparent and Collective

Scientific result = Experiment + *Numerical treatment*

Science at the digital age:

1. Open Article — HAL, BioArxiv
2. Open Data — Data Repositories, Zenodo
3. Open Source — Forges, GitLab, Software Heritage
4. Computational env. — ?

how to ***glue*** all that?

open science, a tautology?

~~Open Science~~ Reproducible Research

Science = Transparent and Collective

Scientific result = Experiment + *Numerical treatment*

Science at the digital age:

1. Open Article — HAL, BioArxiv
2. Open Data — Data Repositories, Zenodo
3. Open Source — Forges, GitLab, Software Heritage
4. Computational env. — ?

how to ***glue*** all that?

today's topic
considering long-term (1-5 years)

open science, a tautology?

Redo (reproduce or replicate) a result?

		audit		opaque		depend?
result	⟵	paper	+	data	+	analysis
data	⟵	protocol	+	instruments	+	materials
analysis	⟵	script	+	data	+	environment

- audit is the « tractable » part
- opaque is generally the hard part

Redo (reproduce or replicate) a result?

		audit		opaque		depend?
result	⟵	paper	+	data	+	analysis
data	⟵	protocol	+	instruments	+	materials
analysis	⟵	script	+	data	+	environment

- **audit** is the « tractable » part
- opaque is generally the hard part
- how to evacuate **depend?** from the equations

Redo (reproduce or replicate) a result?

		audit		opaque		depend?
	result	⟵ paper	+	data	+	analysis
	data	⟵ protocol	+	instruments	+	materials
★	analysis	⟵ script	+	data	+	environment

- **audit** is the « tractable » part
- opaque is generally the hard part
- how to evacuate **depend?** from the equations

★ *our issue*

Redo (reproduce or replicate) a result?

		audit		opaque		depend?
	result	⟵ paper	+	data	+	analysis
	data	⟵ protocol	+	instruments	+	materials
★	analysis	⟵ script	+	data	+	environment

- **audit** is the « tractable » part
- opaque is generally the hard part
- how to evacuate **depend?** from the equations...

...try to turn **environment** into **audit**

★ *our issue*

Redo (reproduce or replicate) a result?

		audit		opaque		depend?
	result	paper	+	data	+	analysis
	data	protocol	+	instruments	+	materials
★	analysis	script	+	data	+	environment

- **audit** is the « tractable » part
- opaque is generally the hard part
- how to evacuate **depend?** from the equations...

...try to turn **environment** into **audit**

★ *our issue*

(« computer » ≈ instrument computationnal env. and « computation » ≈ measurement experimental setup)

Challenges about reproducible research in science

From the « scientific method » viewpoint:

controlling the source of variations

⇒ transparent — as with instrument ≈ computer

From the « scientific knowledge » viewpoint: (universal?)

- Independant observer must be able to observe the same result.
- The observation must be sustainable (to some extent).

⇒ collective

Challenges about reproducible research in science

From the « scientific method » viewpoint:

controlling the source of variations

⇒ transparent — as with instrument ≈ computer

From the « scientific knowledge » viewpoint: (universal?)

- Independant observer must be able to observe the same result.
- The observation must be sustainable (to some extent).

⇒ collective

In a world where (almost) all is *data*

how to redeploy later and elsewhere what has been deployed today and here?

(implicitely using a « computer »)

We will speak about...

(some examples from `C` programming language but all apply equally to any other computational stack)

Questions (1/2)

Bessel function J_0 using C programming language

```
#include <stdio.h>
#include <math.h>

int main(){
   printf("%E\n", j0f(0x1.33d152p+1f));
}
```

Questions (1/2)

Bessel function J_0 using C programming language

```c
#include <stdio.h>
#include <math.h>

int main(){
   printf("%E\n", j0f(0x1.33d152p+1f));
}
```

Alice sees: 5.643440E-08
Blake sees: 5.963430E-08

Determine if the difference is significant or not is let to experts, scientific field by scientific field

Questions (1/2)

Bessel function J_0 using C programming language

```
#include <stdio.h>
#include <math.h>

int main(){
   printf("%E\n", j0f(0x1.33d152p+1f));
}
```

Alice sees: 5.643440E-08
Blake sees: 5.963430E-08

Why? In spite of everything being available (*open*)

Determine if the difference is significant or not is let to experts, scientific field by scientific field

Questions (2/2)

Alice and Blake both run « GCC at version 11.2.0 »

Questions (2/2)

Alice and Blake both run « GCC at version 11.2.0 »

still different*

```
alice@laptop$
                5.643440E-08
blake@desktop$
                5.963430E-08
```

*Not an issue with floating-point computations

Questions (2/2)

Alice and Blake both run « GCC at version 11.2.0 »

still different*

```
alice@laptop$  gcc bessel.c                    && ./a.out
               5.643440E-08
blake@desktop$ gcc bessel.c  -lm -fno-builtin  && ./a.out
               5.963430E-08
```

(due to *constant folding***)

* *Not an issue with floating-point computations*

** *C language is an example, other source but similar issues with Python, R, Perl, etc.*

Questions (2/2)

Alice and Blake both run « GCC at version 11.2.0 »

still different*

```
alice@laptop$  gcc bessel.c                    && ./a.out
               5.643440E-08
blake@desktop$ gcc bessel.c  -lm -fno-builtin  && ./a.out
               5.963430E-08
```

(due to *constant folding***)

Alice and Blake are running **two different computationnal environments**

* *Not an issue with floating-point computations*

** *C language is an example, other source but similar issues with Python, R, Perl, etc.*

Questions (2/2)

Alice and Blake both run « GCC at version 11.2.0 »

still different*

```
alice@laptop$  gcc bessel.c                    && ./a.out
               5.643440E-08
blake@desktop$ gcc bessel.c  -lm -fno-builtin  && ./a.out
               5.963430E-08
```

(due to *constant folding***)

Alice and Blake are running **two different computationnal environments**

More than version number is required

* *Not an issue with floating-point computations*

** *C language is an example, other source but similar issues with Python, R, Perl, etc.*

Questions about a computational environment

- What is the code source?
- What are the tools required for building?
- What are the tools required for running?
- And recursively for each tool...

Questions about a computational environment

- What is the code source?
- What are the tools required for building?
- What are the tools required for running?
- And recursively for each tool...

Answering these questions enables **control over sources of variations**

Questions about a computational environment

- What is the code source?
- What are the tools required for building?
- What are the tools required for running?
- And recursively for each tool...

Answering these questions enables **control over sources of variations**

How to capture the answer of these questions?

Usually: package manager (Conda, APT, Brew, ...); Modulefiles; container; etc. $\Rightarrow$ **not enough!**

Questions about a computational environment

- What is the code source?
- What are the tools required for building?
- What are the tools required for running?
- And recursively for each tool...

Answering these questions enables **control over sources of variations**

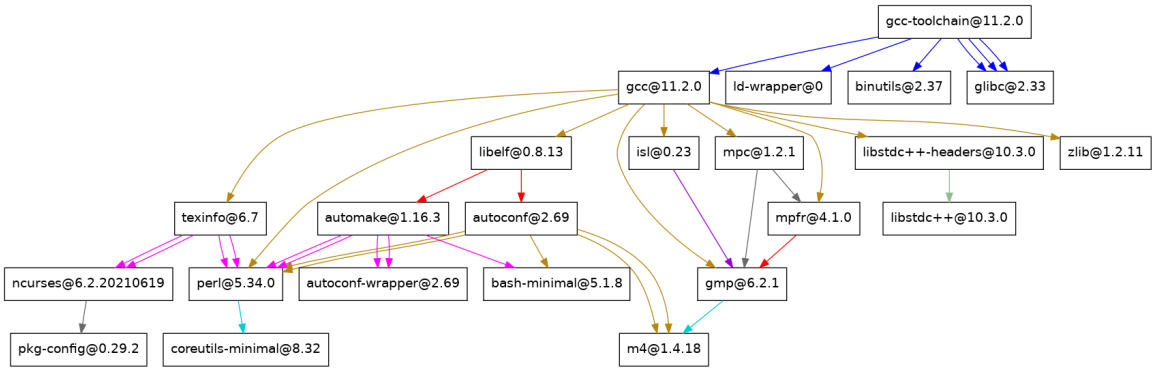
How to capture the answer of these questions?

Usually: package manager (Conda, APT, Brew, ...); Modulefiles; container; etc. $\Rightarrow$ **not enough!**

toward a solution: Guix

When Alice says « GCC at version 11.2.0 »

guix graph



Is it the same "version" of GCC if `mpfr` is replaced by version 4.0 ?

complete graph: 43 ou 104 ou 125 ou 218 nodes
(depending what we consider as *binary seed* for *bootstrapping*)

What does reproducing a computational environment mean?

Alice says "GCC at version 11.2.0"

All the tools (node of the graph) must be captured!

Remember

```c
#include <stdio.h>
#include <math.h>

int main(){
  printf("%E\n", j0f(0x1.33d152p+1f));
}
```

```
      alice@laptop$  gcc bessel.c                    && ./a.out
                     5.643440E-08
    carole@desktop$  gcc bessel.c  -lm -fno-builtin  && ./a.out
                     5.963430E-08
```

(due to *constant folding*)

What is my version of Guix?

`guix describe` = state

```
$ guix describe
Generation 76 Apr 25 2022 12:44:37 (current)
  guix eb34ff1
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: eb34ff16cc9038880e87e1a58a93331fca37ad92

$ guix --version
guix (GNU Guix) eb34ff16cc9038880e87e1a58a93331fca37ad92
```

What is my version of Guix?

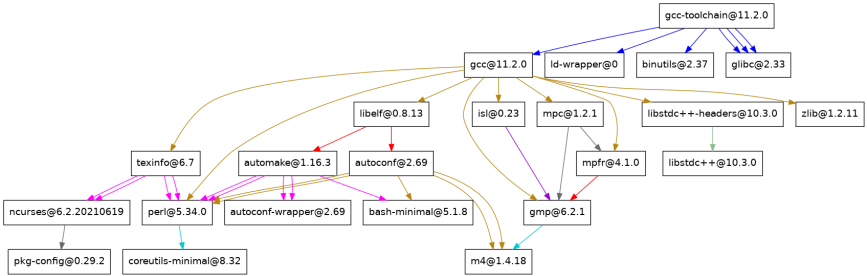
guix describe = state

```
$ guix describe
Generation 76 Apr 25 2022 12:44:37 (current)
  guix eb34ff1
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: eb34ff16cc9038880e87e1a58a93331fca37ad92

$ guix --version
guix (GNU Guix) eb34ff16cc9038880e87e1a58a93331fca37ad92
```

one state **pins** the complete collection of packages and Guix itself

A state can refer to several channels (= Git repository), pointing to URL, branches or commits different
A channel contains a list of recipes (code source, how to build the packages, etc.)

State = Directed Acyclic Graph(*DAG*)



Each node specifies a recipe defining:

- code source — and potentially some *ad-hoc* modifications (`patch`)
- build-time tools — compilers, build automation, configuration flags etc.
- dependencies — other packages ($\rightarrow$recursive $\rightsquigarrow$ graph)

Complete graph : Python = 137 nodes, Numpy = 189, Matplotlib = 915, Scipy = 1439 nodes

Recipe for defining a package

one node of the graph

```
(define python                          ;definition of the node python
  (package
    (name "python")
    (version "3.9.9")
    (source ... )                       ;points to URI source code
    (build-system gnu-build-system)     ;./configure & make
    (arguments ... )                    ; configure flags, etc.
    (inputs (list bzip2                 ;other nodes -> graph (DAG)
                  expat gdbm libffi sqlite ...))))
```

- Each `inputs` is similarly defined (recursion → graph)
- There is no cycle (bzip2 or its `inputs` cannot refer to `python`)

What are the roots of the graph? Part of the broad *bootstrapping* (link) problem

Package manager = graph manager

How to capture this information?

- What is the source code ? `source`
- What are the tools required for building? } `inputs`, `propagated-`, `native-inputs`
- What are the tools required for running? }
- How is each tool produced? `build-system`, `arguments`

Package manager = graph manager

How to capture this information?

- What is the source code ? — `source`
- What are the tools required for building? }
- What are the tools required for running? } `inputs`, `propagated-`, `native-inputs`
- How is each tool produced? — `build-system`, `arguments`

```
(define python                          ;definition of the node python
  (package
    (name "python")
    (version "3.9.9")
    (source ... )                       ;points to URI source code
    (build-system gnu-build-system)     ;./configure & make
    (arguments ... )                    ; configure flags, etc.
    (inputs (list bzip2                 ;other nodes -> graph (DAG)
```

Revision = one specific graph

« GCC at version 11.2.0 » = one pinned graph

```
$ guix describe
Generation 76 Apr 25 2022 12:44:37 (current)
  guix eb34ff1
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: eb34ff16cc9038880e87e1a58a93331fca37ad92
```

this revision `eb34ff1` captures the **complete** graph

- Alice says « I used Guix at revision `eb34ff1` »
- Blake knows all for reproducing the same environment

Collaboration in action

Guix is helping

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

Collaboration in action

Guix is helping

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

- the revision (Guix itself and potentially all the other channels)

```
guix describe -f channels > state-alice.scm
```

Collaboration in action

Guix is helping

collaborate = share one computational environment

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

- the revision (Guix itself and potentially all the other channels)

```
guix describe -f channels > state-alice.scm
```

Collaboration in action

Guix is helping

collaborate = share one computational environment ⇒ share one specific graph

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

- the revision (Guix itself and potentially all the other channels)

```
guix describe -f channels > state-alice.scm
```

Collaboration in action

Guix is helping

collaborate = share one computational environment ⇒ share one specific graph

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

- the revision (Guix itself and potentially all the other channels)

```
guix describe -f channels > state-alice.scm
```

- then **shares these two files**: `state-alice.scm` and `manifest.scm`

Collaboration in action

Guix is helping

collaborate = share one computational environment ⇒ share one specific graph

Alice

describes her environment:

- the list of the tools using the file `manifest.scm`, spawns her environment e.g.,

```
guix shell -m manifest.scm
```

- the revision (Guix itself and potentially all the other channels)

```
guix describe -f channels > state-alice.scm
```

- then **shares these two files**: `state-alice.scm` and `manifest.scm`

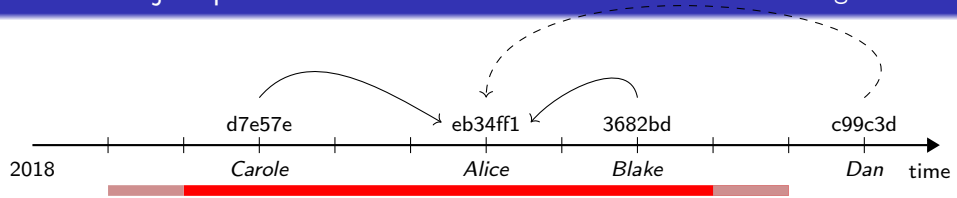
Blake

spawns the same computational environment **from these two files**

```
guix time-machine -C state-alice.scm -- shell -m manifest.scm
```

Reproducible = jump to different states

`guix time-machine`



Requirements for being reproductible with the passing of time using Guix:

- Preservation of the **all** source code
- *Backward* compatibility of the Linux kernel
- Compatibility of *hardware* (to some extent)
- (No time-bomb!)

What is the size of this temporal window where these 3 conditions are satisfied?

To my knowledge, the Guix project is quasi-unique by experimenting since v1.0 in 2019.

how to redeploy later and elsewhere what has been deployed today and here?

Traceability and transparency

being collectively able to study bug-to-bug

Guix should manage everything about the **environment**

```
guix time-machine -C state.scm -- cmd -m list-software.scm
```

if it is specified

« how to build » `channels.scm` (*state*)

« what to build » `manifest.scm` (*software list*)

how to redeploy later and elsewhere what has been deployed today and here?

Traceability and transparency

being collectively able to study bug-to-bug

Guix should manage everything about the **environment**

```
guix time-machine -C state.scm -- cmd -m list-software.scm
```

if it is specified

« how to build » `channels.scm` (*state*)

« what to build » `manifest.scm` (*software list*)

What is required in addition to these 2 files?

Guix cannot fix all the broken world, isn't it? *(opinionated)*

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient?

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient? It depends on efficient... fast? torque? weight?

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient? It depends on efficient... fast? torque? weight?
- Which one is robust?

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient? It depends on efficient... fast? torque? weight?
- Which one is robust? I know which one I choose for going through the unknown.

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient? It depends on efficient... fast? torque? weight?
- Which one is robust? I know which one I choose for going through the unknown.

Easy	vs	Complicated
vs		vs
Simple	vs	Complex

Easy: near to our skill, familiar (≈relative)

Simple: one task, one concept (≈objective)

Guix cannot fix all the broken world, isn't it? *(opinionated)*

- Which one is efficient? It depends on efficient... fast? torque? weight?
- Which one is robust? I know which one I choose for going through the unknown.

Easy	vs	Complicated
vs		vs
Simple	vs	Complex

Easy: near to our skill, familiar (≈relative)

Simple: one task, one concept (≈objective)

Rule of thumb

- Composing simple systems builds complex and robust systems
- Complex and easy systems are complicated thus fragile
- If you have no idea where to start for auditing a tool, it's suspicous!

(Un)Reproducible research *(opinionated)*

the main issue is more about our collective practises
than about technical limitations of our tools

Technical Roadblocks

1. What is the size of the *binary* seed rooting the graph of dependencies?
   language: Haskell, OCaml, Rust, etc.
2. Computational environment (deployment) *bit-for-bit* reproducible is reachable!
   Bit-for-bit reproducible computation is more difficult. Does it make sense?
3. How to audit pre-trained Machine Learning models?
4. Hardware evolution over project duration (2-10 years)

What the time will eat is unknown.

consider efficient as robust (and frugal) then the rest, eventually

ACM REP 24: Conference on Reproducibility and Replicability

- **The Impact of Hardware Variability on Applications Packaged with Docker and Guix: a Case Study in Neuroimaging** (link)
  we study the effect of nine different CPU models using two software packaging systems (Docker and Guix), and we compare the resulting hardware variability to numerical variability measured with random rounding.
- **Longevity of Artifacts in Leading Parallel and Distributed Systems Conferences: a Review of the State of the Practice in 2023** (link)
  By reviewing the methods and tools used to create and share artifacts in a technical, in-depth, and article content-agnostic manner, we found that the state of practice does not address reproducibility in terms of artifact longevity and we expose eight observations that support this finding.
- **Embracing Deep Variability For Reproducibility and Replicability** (link)

Reproducible deployment (ideally)

- Alice says the tool `r-harmony` from Guix revision `eb34ff1`.
- Blake runs on a different machine or at a different point in time:

```
guix time-machine --commit=eb34ff1 -- install r-harmony
```

and Blake deploys the *exact same software environment*, bit-for-bit.

Under the assumptions

- All the source code is still publicly available. (e.g., more than 477)
- All the intermediary builds are deterministic.

"Link rot" empirical evaluation = the problem *(scythe)*

	May 2019 v1.0.0	Apr. 2020 v1.1.0	Nov. 2020 v1.2.0	May 2021 v1.3.0	Dec. 2022 v1.4.0
#sources	8 794	11 659	13 609	15 520	20 184
avail.	91.5%	92.4%	95.0%	95.7%	96.4%
missing	8.5%	7.6%	5.0%	4.3%	3.6%
hash mis.	87	63	69	66	52

"Link rot" empirical evaluation = the problem *(scythe)*

	May 2019 v1.0.0	Apr. 2020 v1.1.0	Nov. 2020 v1.2.0	May 2021 v1.3.0	Dec. 2022 v1.4.0
#sources	8 794	11 659	13 609	15 520	20 184
avail.	91.5%	92.4%	95.0%	95.7%	96.4%
missing	8.5%	7.6%	5.0%	4.3%	3.6%
hash mis.	87	63	69	66	52

- `openjdk-9.181.tar.bz2` is unavailable
  from its original upstream URL as it appears in Guix v1.4.0.

"Link rot" empirical evaluation = the problem *(scythe)*

	May 2019 v1.0.0	Apr. 2020 v1.1.0	Nov. 2020 v1.2.0	May 2021 v1.3.0	Dec. 2022 v1.4.0
#sources	8 794	11 659	13 609	15 520	20 184
avail.	91.5%	92.4%	95.0%	95.7%	96.4%
missing	8.5%	7.6%	5.0%	4.3%	3.6%
hash mis.	87	63	69	66	52

- `openjdk-9.181.tar.bz2` is unavailable
  from its original upstream URL as it appears in Guix v1.4.0.
- `openjdk@9.181` had 184 dependents
  loosing it $\Longrightarrow$ loosing 185 packages, not one.

Software Heritage comes in!

Like all digital information, source code is fragile

link rot: projects are created, moved around, removed

"too big to fail": e.g., Gitorious, Google Code, Bitbucket

Software Heritage comes in!

Like all digital information, source code is fragile

link rot: projects are created, moved around, removed

"too big to fail": e.g., Gitorious, Google Code, Bitbucket

If a website disappears, you go to the Internet Archive...

Where do you do if (a repository on) GitHub or GitLab goes away?

Software Heritage comes in!

Like all digital information, source code is fragile

link rot: projects are created, moved around, removed

"too big to fail": e.g., Gitorious, Google Code, Bitbucket

If a website disappears, you go to the Internet Archive...

Where do you do if (a repository on) GitHub or GitLab goes away?

Answer: **Software Heritage**

Software Heritage comes in!

Like all digital information, source code is fragile

link rot: projects are created, moved around, removed

"too big to fail": e.g., Gitorious, Google Code, Bitbucket

collect, preserve and share source code

If a website disappears, you go to the Internet Archive...

Where do you do if (a repository on) GitHub or GitLab goes away?

Answer: **Software Heritage**

The SWH archive is the largest publicly available archive of software source code.

```
(define-public r-harmony
  (package
    (name "r-harmony")
    (version "1.2.0")
    (source
     (origin
       (method url-fetch)
       (uri (string-append
             "http://cran.r-project.org/src/contrib/harmony_" version ".tar.gz"))
       (sha256
        (base32 "1df7bb9ba3m0c44fhmh8cs4hlkh4fffjwm8rz7l87lf5pdy7sg56"))))
    ;; various fields omitted

(define-public python-scikit-learn
  (package
    (name "python-scikit-learn")
    (version "1.4.2")
    (source
     (origin (method git-fetch)
             (uri (git-reference
                   (url "https://github.com/scikit-learn/scikit-learn")
                   (commit version)))
             (sha256
              (base32 "0pdd508c9540x9qimq83b8kspb6mb98w7w7i7lnb1jqj7rijal6f"))))
    ;; various fields omitted
```

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

source code is *essentially* **content-addressed**

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

source code is *essentially* **content-addressed**

If `uri` becomes stale, e.g., URL no longer available or tempered (hash mismatch)

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

source code is *essentially* **content-addressed**

If `uri` becomes stale, Then Blake can work around.

e.g., URL no longer available or tempered (hash mismatch) (if a copy is available elsewhere)

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

source code is *essentially* **content-addressed**

If `uri` becomes stale, Then Blake can work around.

e.g., URL no longer available or tempered (hash mismatch) (if a copy is available elsewhere)

Elsewhere might be

- a new URL

`guix download` finds the source with the expected hash and proceeds.

Source code identification

`origin` specifies:

- `method`: *tarball*, VCS as Git, Mercurial, Subversion, etc.
- `uri`: upstream location (URL)
- `sha256`: cryptographic hash

source code is *essentially* **content-addressed**

If `uri` becomes stale, Then Blake can work around.

e.g., URL no longer available or tempered (hash mismatch) (if a copy is available elsewhere)

Elsewhere might be

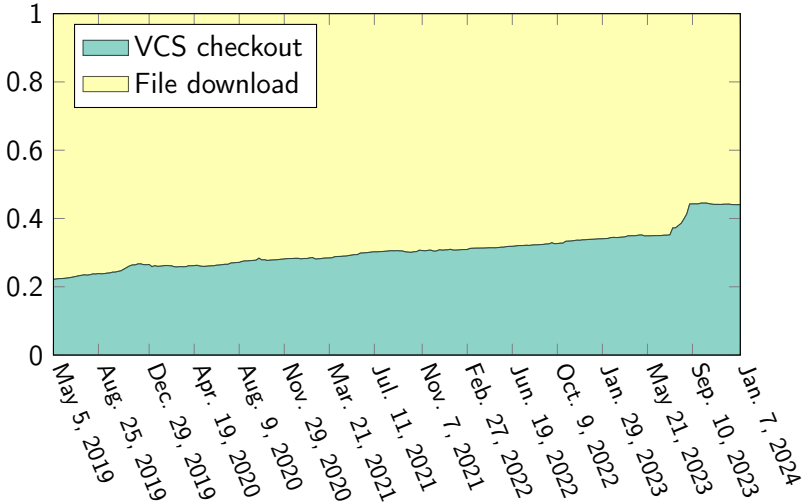
- a new URL

  `guix download` finds the source with the expected hash and proceeds.

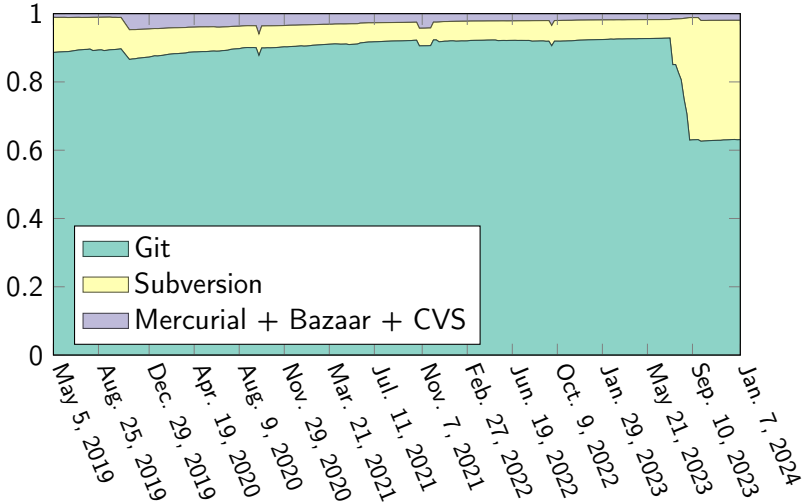
- a **content-addressed server**

  as served by the Guix project or the Nix project, or the Software Heritage initiative.

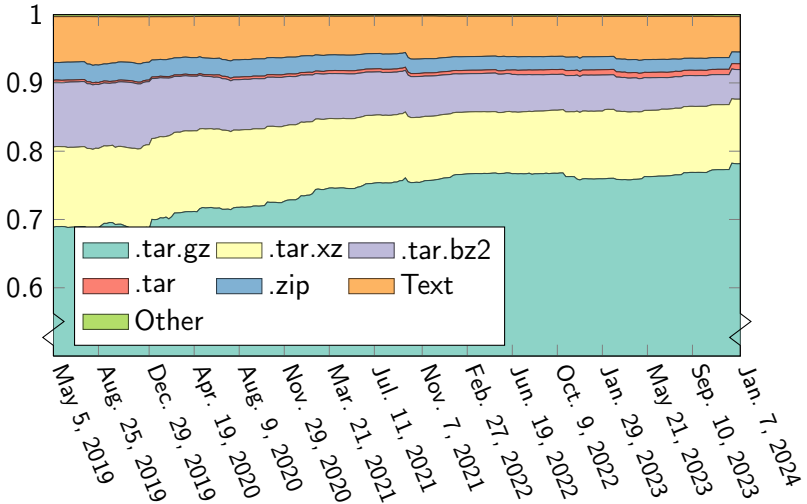
Source type by sampled Guix revision



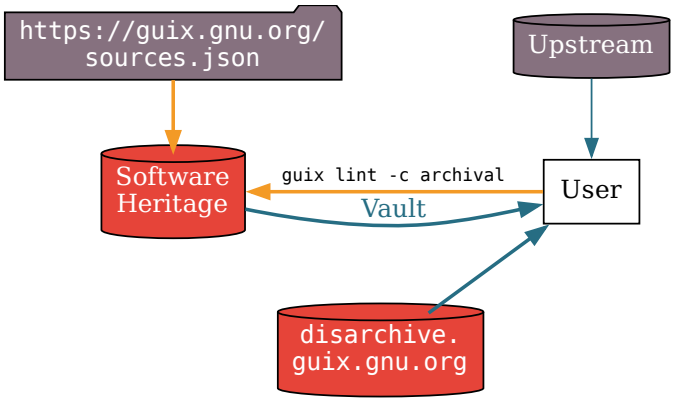
VCS source types by sampled Guix revision



File download types by sampled Guix revision (truncated at 50%)



Architecture = connecting Guix and Software Heritage



Why Disarchive? = issue with compressed *tarballs*

```
(sha256
   (base32 "1df7b..."))
```

```
$ guix hash harmony_1.2.0.tar.gz
1df7b...

$ guix hash \                  # Fast compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -1 -c)
03v29...

$ guix hash \                  # Best compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -9 -c)
10j87...
```

```
# Extract the compressed tarball
$ guix hash harmony-1.2.0 \
    --serializer=nar
b7900...

$ guix hash harmony-1.2.0 \
    --serializer=git
3a46b...

$ guix hash harmony-1.2.0 \
    --serializer=git --hash=sha1
75b43...
```

Why Disarchive? = issue with compressed *tarballs*

```
(sha256
   (base32 "1df7b...")
```

```
$ guix hash harmony_1.2.0.tar.gz
1df7b...

$ guix hash \                  # Fast compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -1 -c)
03v29...

$ guix hash \                  # Best compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -9 -c)
10j87...
```

Process of creating compressed tarballs might vary.
(compression, timestamps, file properties, etc.)

```
# Extract the compressed tarball
$ guix hash harmony-1.2.0 \
    --serializer=nar
b7900...

$ guix hash harmony-1.2.0 \
    --serializer=git
3a46b...

$ guix hash harmony-1.2.0 \
    --serializer=git --hash=sha1
75b43...
```

Why Disarchive? = issue with compressed *tarballs*

```
(sha256
   (base32 "1df7b..."))
```

```
$ guix hash harmony_1.2.0.tar.gz
1df7b...

$ guix hash \                  # Fast compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -1 -c)
03v29...

$ guix hash \                  # Best compression
  <(gzip -dc harmony_1.2.0.tar.gz | gzip -9 -c)
10j87...
```

Process of creating compressed tarballs might vary.
(compression, timestamps, file properties, etc.)

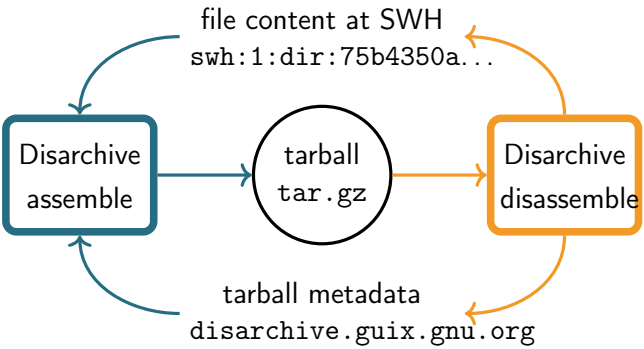
```
# Extract the compressed tarball
$ guix hash harmony-1.2.0 \
    --serializer=nar
b7900...

$ guix hash harmony-1.2.0 \
    --serializer=git
3a46b...

$ guix hash harmony-1.2.0 \
    --serializer=git --hash=sha1
75b43...
```

Cryptographic hash algorithm and data serializer are important for retrieving.

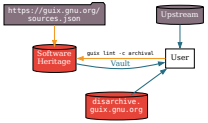
Disarchive disassemble output = description of metadata

```
(disarchive
  (version 0)
  (gzip-member
    (name "harmony_1.2.0.tar.gz")
    (digest (sha256 "a63c7d7..."))
    (header (mtime 1701246604) (extra-flags 0) (os 3))
    (footer (crc 2567676087)
            (isize 6225920))
    (compressor gnu)
    (input
     (tarball
       (name "harmony_1.2.0.tar")
       (digest (sha256 "6c50a34..."))
       (default-header
         (owner "root")
         (group "root"))
       (headers
        ("harmony/"
         (mode 493)
         (mtime 1701246604)
         (chksum 5084)
         (typeflag 53))
        ;; many headers omitted
        ("harmony/inst/doc/Seura
         (size 4130)
         (mtime 1701214143)
         (chksum 6701)))
       (padding 0)
       (input
         (directory-ref
```

Separate storage



Retrieving source code = SWH Vault + Disarchive

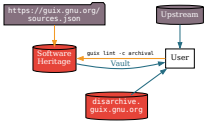


content-address

Guix: "*normalized archived*" (`nar`) + `sha256`

SWH: SWHID = Git compatible `sha1`

Retrieving source code = SWH Vault + Disarchive



content-address

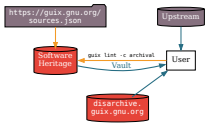
Guix: "*normalized archived*" (`nar`) + `sha256`

SWH: SWHID = Git compatible `sha1`

Case: VCS checkouts

- SWH addresses content as SWHID and associates the `nar-sha256` as *external identifier*.
- Guix queries using `nar-sha256` and gets back SWHID.
- Guix asks SWH Vault to "cook" the files and fetch them.

Retrieving source code = SWH Vault + Disarchive



content-address

Guix: "*normalized archived*" (`nar`) + `sha256`

SWH: SWHID = Git compatible `sha1`

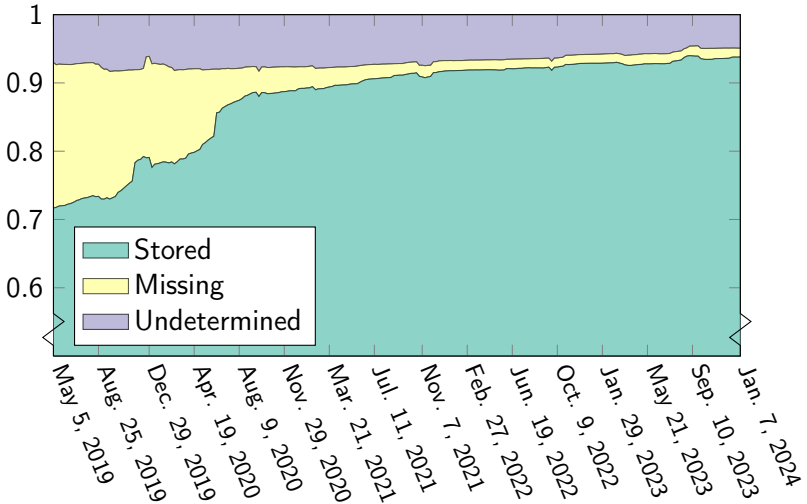
Case: VCS checkouts

- SWH addresses content as SWHID and associates the `nar-sha256` as *external identifier*.
- Guix queries using `nar-sha256` and gets back SWHID.
- Guix asks SWH Vault to "cook" the files and fetch them.

Case: *tarballs*

- Using Disarchive disassemble output, from `nar-sha256`, Guix gets SWHID.
- Guix asks SWH Vault to "cook" the files and fetch them.
- Using Disarchive disassemble output, Guix assembles bit-identical *compressed tarball*.

Coverage by sampled Guix revision



Work in progress

```
guix time-machine --commit=v1.0.0 -- install r-harmony
```

Installs (and potentially rebuilds) Harmony defined in Guix 1.0.0 from 2019.

- This command exploits SWH support as it was in 2019: in its infancy.
- Recovery mechanism is itself improving over time.
- Mitigations:
  - Delegate downloading to the Guix build daemon.
    (special and dedicated "builders" as `builtin:download` or `builtin:git-download`)
  - Recover source code referenced by past revisions using present-day techniques.

- Support more archive formats including lzip, Zip, unusual gzip compression.
- Deal with (long) *cooking* time by SWH Vault,
  from minutes to days depending on artifact size and service load.

Rebuilding the whole only from SWH

My attempts:

- June 2023 redoing paper from 2020 (link)
- December 2023 redoing paper from 2022 (link)

Two main difficulties remain:

- Bootstrapping — binary seed rooting the graph of dependencies
  - storing the seed itself
  - rebuilding from the seed, if needed
- Time bomb — deterministic build depends on date
  - We can fix the future not the past.
  - March 2024: Adventures on the quest for long-term reproducible deployment (link)

Rebuilding the whole only from SWH

My attempts:

- June 2023 redoing paper from 2020 (link)
- December 2023 redoing paper from 2022 (link)

Two main difficulties remain:

- Bootstrapping — binary seed rooting the graph of dependencies
  - storing the seed itself
  - rebuilding from the seed, if needed
- Time bomb — deterministic build depends on date
  - We can fix the future not the past.
  - March 2024: Adventures on the quest for long-term reproducible deployment (link)

Conclusion: tool still missing

Exploration of ideas
https://simon.tournier.info/posts/2024-04-11-rewrite-drv.html

Reproducible computational environment, when? *(opinionated)*

when collective practises will stop to promote *engineering methods*

engineering method
what do we **gain** compared to current? vs *science method*
what do we **understand** compared to current?

redo the past = already hard tasks

Thanks Guix, the situation is improving over the years

Mnemosyne
pearls
memory

Cronus
scythe
time

We cannot predict beforehand
what the scythe will cut

Pearls

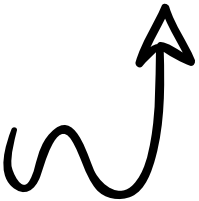
- simple made easy
- efficient means robust
- content-addressed, intrinsic identifier, inherent reference
- transparent and auditable computational environment
- focus on user-autonomy

Is Guix one pearl against the scythe?
small cookie for thought: Is Guix Simple or Easy?

The vision to *reach*

Software Heritage

The ReScience Journal



Questions?

guix-science@gnu.org

dedicated Mattermost (chat) (link)

https://hpc.guix.info/events/2024-2025/café-guix/

Appendix

Example of VCS retrieval

```
building /gnu/store/agsi5ynwvmyfscc2avxkf7i3089m1p0i-scons-3.0.4-checkout.drv...
Initialized empty Git repository in /gnu/store/2p3cb96q8zk2pnarcnkwaifqw3l8gc70-scons-3.0.4-checkout/.git
fatal: unable to access 'https://github.com/SCons/scons.git/': Could not resolve host: github.com
Failed to do a shallow fetch; retrying a full fetch...
fatal: unable to access 'https://github.com/SCons/scons.git/': Could not resolve host: github.com
git-fetch: '/gnu/store/lcygm0p2d59acvwi12lwldg5c0d4czpr-git-minimal-2.41.0/bin/git fetch origin'
failed with exit code 128
Trying content-addressed mirror at bordeaux.guix.gnu.org...
Unable to fetch from bordeaux.guix.gnu.org, getaddrinfo-error: (-2)
Trying content-addressed mirror at ci.guix.gnu.org...
Unable to fetch from ci.guix.gnu.org, getaddrinfo-error: (-2)
Trying content-addressed mirror at bordeaux.guix.gnu.org...
Unable to fetch from bordeaux.guix.gnu.org, getaddrinfo-error: (-2)
Trying to download from Software Heritage...
SWH: found directory
with nar-sha256 hash 16a209173f87735020b29d84f497d44204cbcf86a451066342c51ff47996c8f7
at 'swh:1:dir:d3d1330dfc409be4624a01d384868fea0427c4c3'
swh:1:dir:d3d1330dfc409be4624a01d384868fea0427c4c3/
swh:1:dir:d3d1330dfc409be4624a01d384868fea0427c4c3/.appveyor.yml
...
```

Example of *tarball* retrieval

```
Trying to use Disarchive to assemble /gnu/store/zwmrwlb2l53xbllxcw3axad54kyqcplp-Python-3.12.2.tar.xz...
Retrieving Disarchive spec
from https://disarchive.guix.gnu.org/sha256/be28112dac813d2053545c14bf13a16401a21877f1a69eb6ea5d84c4a0f3d
Assembling the directory Python-3.12.2
Downloading /gnu/store/zwmrwlb2l53xbllxcw3axad54kyqcplp-Python-3.12.2.tar.xz from Software Heritage...
SWH vault: requested bundle cooking, waiting for completion...
SWH vault: Processing...
swh:1:dir:72d77318a8c52ddfc004251fb7297799135704e6/
swh:1:dir:72d77318a8c52ddfc004251fb7297799135704e6/Python-3.12.2/pyconfig.h.in
...
Checking Python-3.12.2 digest... ok
Assembling the tarball Python-3.12.2.tar
Checking Python-3.12.2.tar digest... ok
Assembling the XZ file Python-3.12.2.tar.xz
Checking Python-3.12.2.tar.xz digest... ok
Copying result to /gnu/store/zwmrwlb2l53xbllxcw3axad54kyqcplp-Python-3.12.2.tar.xz
successfully built /gnu/store/nx97h7yr21l04nn60mqlf1yzfyxj06jh-Python-3.12.2.tar.xz.drv
source is at 'Python-3.12.2'
applying '/gnu/store/cdla0h7pcnckxlk3aflik3zsmbsfxzfp-python-3-deterministic-build-info.patch'...
applying '/gnu/store/ns40bs4bs19syckgh7v37rbxax0wfq01-python-3.12-fix-tests.patch'...
```

Example of *bugs* (read Missing in Coverage)

```
Trying to download from Software Heritage...
SWH: found directory
with nar-sha256 hash c98bd6991721d60b9a79600428bfbe8db0aaac3d383cd2df803ed7867c7cb63b
at 'swh:1:dir:218d95849f10fc0691d7dfa80999ce5061e654ef'
swh:1:dir:218d95849f10fc0691d7dfa80999ce5061e654ef/
...
swh:1:dir:218d95849f10fc0691d7dfa80999ce5061e654ef/wisp.py
r:sha256 hash mismatch for /gnu/store/7pcac04x82wyhknyfkdwhk3j958n2r75-guile-wisp-1.0.7-checkout:
  expected hash: 0fxngiy8dmryh3gx4g1q7nnamc4dpszjh130g6d0pmi12ycxd2y9
  actual hash:   0z7y487nnmw22xry82bb75shwp50gacm4kbwn01vhhli2bchpx37
hash mismatch for store item '/gnu/store/7pcac04x82wyhknyfkdwhk3j958n2r75-guile-wisp-1.0.7-checkout'
build of /gnu/store/8lcrd6n6m18hzh9dszm8c1xhjyfd54d9-guile-wisp-1.0.7-checkout.drv failed
View build log at '/var/log/guix/drvs/8l/crd6n6m18hzh9dszm8c1xhjyfd54d9-guile-wisp-1.0.7-checkout.drv.gz'
guix build: error: build of '/gnu/store/8lcrd6n6m18hzh9dszm8c1xhjyfd54d9-guile-wisp-1.0.7-checkout.drv' f
```

Bug report #5093 (SWH)
Patch #71631 (Guix)

how to redeploy later and elsewhere what has been deployed today and here?

Traceability and transparency

being collectively able to study bug-to-bug

Guix should manage everything about the **environment**

```
guix time-machine -C state.scm -- cmd -m list-software.scm
```

if it is specified

« how to build » `channels.scm` (*state*)

« what to build » `manifest.scm` (*software list*)

how to redeploy later and elsewhere what has been deployed today and here?

Traceability and transparency

being collectively able to study bug-to-bug

Guix should manage everything about the **environment**

```
guix time-machine -C state.scm -- cmd -m list-software.scm
```

if it is specified

« how to build » `channels.scm` (*state*)

« what to build » `manifest.scm` (*software list*)

What is required in addition to these 2 files?

Preservation of what? (1/3)

```
guix time-machine -C channels.scm -- shell -m manifest.scm
```

- ★ each channel used by `channels.scm` (= Git repository defining packages)
- ★ code source used by `manifest.scm` (= URI pointing to upstream)

```
(define python                          ;package definition
  (package
    (name "python")
    (version "3.9.9")
    (source ... )                       ;package source
    (build-system gnu-build-system)
    (arguments ... )
    (inputs (list ...))))
```

Preservation of what? (2/3)

example of source

```
(source
 (origin
   (method url-fetch)
   (uri (string-append "https://www.python.org/ftp/python/"
                       version "/Python-" version ".tar.xz"))
   (patches (search-patches ...))
   (sha256
    (base32
     "09vd7g71i11iz5ydqghwc8kaxr0vgji94hhwwnj77h3kll28r0h6"))))
```

Preservation of what? (3/3)

- Git repository (channel)
- `source`
  - archive *tarballs* (compressed) `url-fetch`
  - Git repository `git-fetch`
  - Subversion repository `svn-fetch`
  - Mercurial repository `hg-fetch`
  - CVS repository `cvs-fetch`

Preservation of what? (3/3)

- Git repository (channel)
- `source`
  - archive *tarballs* (compressed) `url-fetch`
  - Git repository `git-fetch`
  - Subversion repository `svn-fetch`
  - Mercurial repository `hg-fetch`
  - CVS repository `cvs-fetch`

```
$ guix repl -- sources.scm | sort | uniq -c | sort -nr
  13432 url-fetch
   6691 git-fetch
    391 svn-fetch
     43 other
     31 hg-fetch
      3 cvs-fetch
```

Why preserving?

Because **online services sometimes stop**

- Google Code (link) early 2016
- Alioth (Debian) in 2018 replaced by Salsa
- Gna! in 2017 after 13 years
- Gitourious in 2015 (the second most popular service for hosting Git repository in 2011)
- etc.

Why preserving?

Because **online services sometimes stop**

- Google Code (link) early 2016
- Alioth (Debian) in 2018 replaced by Salsa
- Gna! in 2017 after 13 years
- Gitourious in 2015 (the second most popular service for hosting Git repository in 2011)
- etc.
- `gforge.inria.fr` for example Guix issue #42162 (link)

  ```
  Believe it or not, gforge.inria.fr was finally phased out on
  Sept. 30th.  And believe it or not, despite all the work and all the
  chat :-), we lost the source tarball of Scotch 6.1.1 for a short period
  of time (I found a copy and uploaded it to berlin a couple of hours
  ago).
  ```

How to preserve?

Forge $\neq$ Archive

collaborative software platform for developing
L'objectif d'une forge est de permettre à plusieurs développeurs de **participer ensemble au développement** d'un ou plusieurs logiciels, le plus souvent à travers le réseau Internet.

https://fr.wikipedia.org/wiki/Forge_(informatique)

(no English wikipedia entry)
L'archivage est un ensemble d'actions qui a pour but de garantir l'accessibilité sur le long terme d'informations (dossiers, documents, données) que l'on doit ou souhaite conserver pour des raisons juridiques

https://fr.wikipedia.org/wiki/Archivage

Software Heritage « *are building the universal software archive* » (link)

Online service sometimes stop...

Why would it be different for Software Heritage?

No guarantee but...

Software Heritage is an open, non-profit initiative unveiled in 2016 by Inria. It is supported by a broad panel of institutional and industry partners, in collaboration with UNESCO.

The long term goal is to collect all publicly available software in source code form together with its development history, replicate it massively to ensure its preservation, and share it with everyone who needs it.

- Strong support by national and international institutes
- With the mission to specifically archive all the open source code

(SWH demo?)

Preservation with Software Heritage

https://www.softwareheritage.org/

collect and **preserve** software in source code form in the very **long term**
(not a forge!)

Guix is able:

- save source code from Guix package definition and the Guix package definition itself
- use Software Heritage archive as fallback if upstream source disappears

Questions:

- How to cite a software? Reference to source code only? Dependencies? Build options?
- **Intrinsic** identifier *(depends only on the object; as checksum)*
  vs **Extrinsic** identifier
  (depends on a register to keep the correspondence between identifier and object; as label version)

Fallback in action

```
$ guix time-machine -C channels.scm -- shell -m manifest.scm
Updating channel 'guix' from Git repository at 'https://git.savannah.gnu.org/gi
Updating channel 'example' from Git repository at 'https://whatever-here.org/do
SWH: found revision 67c9f2143aa6f545419ae913b4ae02af4cd3effc with directory at
SWH vault: requested bundle cooking, waiting for completion...
swh:1:rev:67c9f2143aa6f545419ae913b4ae02af4cd3effc.git/
[...]
fatal: could not read Username for 'https://github.com': No such device or addr
Trying content-addressed mirror at berlin.guix.gnu.org...
Trying to download from Software Heritage...
SWH: found revision e1eefd033b8a2c4c81babc6fde08ebb116c6abb8 with directory at'
[...]
```

https://simon.tournier.info/posts/2021-10-25-software-heritage.html

Redo the past

Being able to redeploy from now the same computational environment as 3 years ago

It requires:

- Exact same source code
- Rebuild on compatible hardware
- Deterministic rebuild

hard engineering tasks