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)  

Many causes... one solution?

Open Science helps

\[
\begin{align*}
\text{reproducibility} &= \text{verification} \\
\text{replicability} &= \text{validation}
\end{align*}
\]
Open Science

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

Science in 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

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

Science in the digital age:

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

How to glue it all?

“Open science”, a tautology?
Open Science

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

Science in 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 it all?

“Open science”, a tautology?
Open Science

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

Science in 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 it all?

today’s topic

“Open science”, a tautology?
Redo (reproduce or replicate) a result?

<table>
<thead>
<tr>
<th>audit</th>
<th>opaque</th>
<th>depend?</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>paper + data + analysis</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>protocol + instrument + materials</td>
<td></td>
</tr>
<tr>
<td>analysis</td>
<td>script + data + environment</td>
<td></td>
</tr>
</tbody>
</table>

- **audit** is the “tractable” part
- **opaque** is generally the hard part
Redo (reproduce or replicate) a result?

<table>
<thead>
<tr>
<th>audit</th>
<th>opaque</th>
<th>depend?</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>paper + data + analysis</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>protocol + instrument + materials</td>
<td></td>
</tr>
<tr>
<td>analysis</td>
<td>script + data + environment</td>
<td></td>
</tr>
</tbody>
</table>

- **audit** is the “tractable” part
- **opaque** is generally the hard part
- how to eliminate **depend?** from the equations
Redo (reproduce or replicate) a result?

<table>
<thead>
<tr>
<th>audit</th>
<th>opaque</th>
<th>depend?</th>
</tr>
</thead>
<tbody>
<tr>
<td>result ←− paper + data + analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data ←− protocol + instrument + materials</td>
<td></td>
<td></td>
</tr>
<tr>
<td>★ analysis ←− script + data + environment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **audit** is the “tractable” part
- **opaque** is generally the hard part
- how to eliminate **depend?** from the equations

★ *our issue*
Redo (reproduce or replicate) a result?

<table>
<thead>
<tr>
<th>audit</th>
<th>opaque</th>
<th>depend?</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>←← paper</td>
<td>+ data</td>
</tr>
<tr>
<td>data</td>
<td>←← protocol</td>
<td>+ instrument</td>
</tr>
<tr>
<td>★ analysis</td>
<td>←← script</td>
<td>+ data</td>
</tr>
</tbody>
</table>

- **audit** is the “tractable” part
- **opaque** is generally the hard part
- how to eliminate **depend**? from the equations

**our issue**

(“computer” ≈ instrument
  computationnal env. ↔
  “computation” ≈ measurement
  experimental setup)
Redo (reproduce or replicate) a result?

<table>
<thead>
<tr>
<th>audit</th>
<th>opaque</th>
<th>depend?</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>←− paper</td>
<td>+ data</td>
</tr>
<tr>
<td>data</td>
<td>←− protocol</td>
<td>+ instrument</td>
</tr>
<tr>
<td>★ analysis</td>
<td>←− script</td>
<td>+ data</td>
</tr>
</tbody>
</table>

- **audit** is the “tractable” part
- **opaque** is generally the hard part
- **how to eliminate** depend? **from the equations**...

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

**our issue**

( “computer” ≈ instrument and “computation” ≈ measurement
 computationnal env. ↔ 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:

▶ Independent 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:

▶ Independent 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 redo later and elsewhere what has been done here and today?

(implicitly using a “computer”)
Questions about a computational environment

- What is source code?
- What are the tools required for building?
- What are the tools required at run time?
- And recursively for each tool...
Questions about a computational environment

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

Answering these questions enables control over sources of variations.
Questions about a computational environment

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

Answering these questions enables control over sources of variations.

How to capture the answer to these questions?

Usually: package manager (Conda, APT, Brew, ...); Modulefiles; container; etc.
Solution(s)

1. package manager: APT (Debian/Ubuntu), YUM (RedHat), etc.
2. environment manager: Conda, Pip, Modulefiles, etc.
3. container: Docker, Singularity

APT, Yum: Hard to have several versions or rollback?

Pip/Conda: Transparency?

Modulefiles: How are they maintained? (who uses them on their laptop?)

Docker: Dockerfile based sur APT, YUM, etc.

```
RUN apt-get update && apt-get install
```

FOSDEM 2023
Toward reproducible research using Guix
Solution(s)

1. package manager: APT (Debian/Ubuntu), YUM (RedHat), etc.
2. environment manager: Conda, Pip, Modulefiles, etc.
3. container: Docker, Singularity

Guix = #1 + #2 + #3

APT, Yum  Hard to have several versions or rollback?
Pip/Conda  Transparency?
           who knows what’s inside PyTorch with pip install torch? (link)
Modulefiles  How are they maintained? (who uses them on their laptop?)
Docker  Dockerfile based sur APT, YUM, etc.

RUN apt-get update && apt-get install
Guix: computational environment manager on steroids

a package manager
transactional and declarative
which produces shareable packs
which produces isolated virtual machines
used to build a whole Linux distribution
...and also a Scheme library...

(as APT, Yum, etc.)
(rollback, concurrent versions)
(Docker or Singularity container)
(à la Ansible or Packer)
(better than other? :-)
(extensibility!)

Guix runs on top of a Linux distribution, or standalone.

Easy to try
Guix: computational environment manager on steroids

a package manager
transactional and declarative
which produces shareable packs
which produces isolated virtual machines
used to build a whole Linux distribution
... and also a Scheme library...

(as APT, Yum, etc.)
(rollback, concurrent versions)
(Docker or Singularity container)
(à la Ansible or Packer)
(better than other? :-)
(extensibility!)

Guix runs on top of a Linux distribution, or standalone.

Easy to try

25 minutes...
... is a quick summary calling for your own experimentation (maybe?)

(this talk is an small apéritif)
Guix: computational environment manager on steroids

A package manager (as APT, Yum, etc.)

Transactionally and declaratively

Which produces shareable packs

Which produces isolated virtual machines

Used to build a whole Linux distribution

...and also a Scheme library...

how Guix can help open research?

25 minutes...

...is a quick summary calling for your own experimentation (maybe?)

(this talk is an small apéritif)

Guix runs on top of a Linux distribution, or standalone.

Easy to try
Install on **foreign distro**

Guix runs on **any recent Linux distribution**

Superuser privileges \((root)\) is only required for installing.

```bash
$ cd /tmp
$ wget https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh
$ chmod +x guix-install.sh
$ sudo ./guix-install.sh
```

(More some minor adjustments, see the manual)

Getting started:

```bash
$ guix help
```
Let talk about

- Deployment of scientific software using Guix
- Reproducible from one machine to the other? About time?

1. Introduction

2. Package management
   - Basics

3. Reproducing a computational environment

4. Summary
Guix, yet another package manager!

(Julia is one example, idem for any other)

```guix
guix search high-performance dynamic language # 1.
guix show julia # 2.
guix install julia # 3.
guix install julia-pyplot julia-dataframes # 4.
guix remove julia-pyplot # 5.
guix install julia-csv julia-zygote # 6.
```

Alias of `guix package`, e.g. `guix package --install`

**Transactional**

```guix
guix package -r julia-pyplot -i julia-csv julia-zygote # 5. & 6.
guix package --roll-back # 4. -> 3.
```
Guix, really yet another package manager?

- Command line interface as many other package managers
- Package install/remove without any special privilege
- Transactional
  - (= no « broken » state)
- Binary *substitutes*
  - (= fetch pre-compiled components)
- Declarative management
  - (declarative = configuration file)
  - (guix shell --container)
  - (guix pack -f docker)
  - (guix system image)
- Isolated environment *on-the-fly*
- Factory for *images*

The *profiles* allow to install several versions.

*profile* ≈ “environment à la virtualenv”
Interesting features, but what makes it reproducible?

We need to talk about versions!

Example: Alice and Blake are collaborating
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 is my version of Guix?

$ 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
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

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
- build-time tools
- dependencies

and potentially some ad-hoc modifications (patch) compilers, build automation, configuration flags etc. other packages (→recursive ⇝ graph)

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

“GCC at version 11.2.0” = one fixed 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
Alice describes her environment:

- the list of the tools using the file `manifest.scm`

spawns her environment e.g.,

```
guix shell -m manifest.scm
```
Alice describes her environment:

- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels): 

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

spawns her environment e.g.,

```
guix shell -m manifest.scm
```
Collaboration in action

2 command lines + 2 files

package manager ↔ state

collaborate = share one computational environment

Alice describes her environment:

- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels):
  
  ```
  guix describe -f channels > state-alice.scm
  ```

spawns her environment e.g.,

```guix shell -m manifest.scm```
Collaboration in action

2 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:
- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels):

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

spawns her environment e.g.,

```
    guix shell -m manifest.scm
```
Collaboration in action

2 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:
- the list of the tools using the file manifest.scm
- the revision (Guix itself and potentially all the other channels):
  guix describe -f channels > state-alice.scm

spawns her environment e.g.,
  guix shell -m manifest.scm

then shares these two files: state-alice.scm and manifest.scm.
Collaboration in action

2 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:

▶ the list of the tools using the file `manifest.scm`
▶ the revision (Guix itself and potentially all the other channels):

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

spawns her environment e.g., then shares these two files: `state-alice.scm` and `manifest.scm`.

Carole can also reproduce the same environment as Alice and Blake.
Collaboration in action

3 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:

- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels): `guix describe -f channels > state-alice.scm`

spawns her environment e.g., then shares these two files: `state-alice.scm` and `manifest.scm`.

Blake spawns the same computational environment from these two files.
**Collaboration in action**

3 command lines + 2 files

---

package manager ↔ state ↔ graph manager

**collaborate = share one computational environment ⇒ share one specific graph**

Alice describes her environment:

- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels):
  ```bash
guix describe -f channels > state-alice.scm
  ```

spawns her environment e.g.,

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

Blake spawns the same computational environment from these two files

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

Carole can also reproduce the same environment as Alice and Blake.
Collaboration in action

3 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:

- the list of the tools using the file `manifest.scm`
- the revision (Guix itself and potentially all the other channels):
  
  ```
  guix describe -f channels > state-alice.scm
  ```

  spawns her environment e.g., 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
  ```

Carole can also reproduce the same environment as Alice and Blake.

FOSDEM 2023
Toward reproducible research using Guix
16 / 27
Collaboration in action

3 command lines + 2 files

package manager ↔ state ↔ graph manager

collaborate = share one computational environment ⇒ share one specific graph

Alice describes her environment:

▶ the list of the tools using the file manifest.scm
▶ the revision (Guix itself and potentially all the other channels):
  guix describe -f channels > state-alice.scm

spawns her environment e.g., 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

Carole can also reproduce the same environment as Alice and Blake.
Requirements for being reproducible with the passing of time using Guix:

- Preservation of the **all** source code (≈ 75% archived (link) in Software Heritage (link))
- *Backward* compatibility of the Linux kernel
- Compatibility of *hardware* (to some extent)

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.
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)
▶ **Extrinsic** identifier  
  (depends on a register to keep the correspondence between identifier and object; as label version)
Summary
Guix: computational environment manager on steroids

- **a declarative package manager**
  - `guix package` (\(-m manifest\))
- temporarily extended
  - `guix shell` (\(--container\))
- controlling exactly the *state*
  - `guix time-machine` (\(-C channels\))

+ `guix describe`

**Guix precisely controls the complete implicit graph of configurations**

```
guix time-machine -C channels.scm -- command options manifest.scm
```

**manifest.scm is reproducible** at the exact same `channels.scm`

*Reproducible from one machine to another with the passing of time*
Resources

GuixHPC = Guix for scientific research

https://hpc.guix.info

▶ Toward practical transparent verifiable and long-term reproducible research using Guix (Nature Scientific Data, 2022)(link)

▶ Reproductibilité des environnements logiciels avec GNU Guix (mini-tuto 1h JRES, 2022)(link in French)

https://10years.guix.gnu.org/

FOSDEM 2014 (1), 15 (2), 16 (3 4 5 6), 17 (7 8 9 10 11 12 13 14), 18 (15), 19 (16 17 18 19), 20 (20 21 22 23 24 25), 21 (26 27 28 29), 22 (30 31)
### Running in production

<table>
<thead>
<tr>
<th>Organization</th>
<th>Location</th>
<th>Nodes</th>
<th>Cores/Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid’5000</td>
<td>(France)</td>
<td>828-nodes</td>
<td>(12,000+ cores, 31 clusters)</td>
</tr>
<tr>
<td>GliCID (CCIPL)</td>
<td>(France)</td>
<td>392-nodes</td>
<td>(7500+ cores)</td>
</tr>
<tr>
<td>PlaFrIM Inria</td>
<td>(France)</td>
<td>120-nodes</td>
<td>(3000+ cores)</td>
</tr>
<tr>
<td>GriCAD</td>
<td>(France)</td>
<td>72-nodes</td>
<td>(1000+ cores)</td>
</tr>
<tr>
<td>Max Delbrück Center</td>
<td>(Allemagne)</td>
<td>250-nodes</td>
<td>+ workstations</td>
</tr>
<tr>
<td>UMC</td>
<td>(Pays-Bas)</td>
<td>68-nodes</td>
<td>(1000+ cores)</td>
</tr>
<tr>
<td>UTHSC Pangenome</td>
<td>(USA)</td>
<td>11-nodes</td>
<td>(264 cores)</td>
</tr>
</tbody>
</table>

- more all laptops and desktops

#### Examples:

- Azithromycin promotes relapse by disrupting immune and metabolic networks after allogeneic stem cell transplantation ([Blood, 2022](link))

- SARS-CoV-2 infection dynamics revealed by wastewater sequencing analysis and deconvolution ([Science of The Total Environment, 2022](link))
Finalizing

the message you should get back to home
How to redo later and elsewhere what has been done here and today?

Open Science

Traceability and transparency

being able, collectively, to study bug-to-bug

Guix should manage everything about the computational environment

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

if it is specified

“how to build”

“what to build”

channels.scm

manifest.scm
The vision

Software Heritage

The ReScience Journal

Guix
Thanks:

- Guix community. You are awesome!
- Ludovic Courtès
- Ricardo Wurmus

Questions?

guix-science@gnu.org

#guix and #guix-hpc on Libera Chat IRC network
dedicated Mattermost (chat) (link)

Café Guix

https://hpc.guix.info/events/2022/café-guix/

These slides are archived.
(Software Heritage id swh:1:dir:1a1206421f77f1b89068eba9c07733f40584461a)
Appendix

More about

- We need more than label version
- Declarative approach
- What a package looks like
- Package transformation
- What the file capturing the state looks like
- Extended environment, isolated
- What is the issue about container and how guix pack helps
Bessel function $J_0$ in the 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
Why? In spite of everything being available ("open").
Determining whether the difference is significant or not is left to experts of each scientific domain.
Bessel function $J_0$ in the 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

Determining whether the difference is significant or not is left to experts of each scientific domain.
Bessel function $J_0$ in the 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

Why? In spite of everything being available ("open").

Determining whether the difference is significant or not is left to experts of each scientific domain.
Alice and Blake both run “GCC at version 11.2.0”
In concrete terms (2/2)

Alice and Blake both run “GCC at version 11.2.0”

still different*

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>alice@laptop$</td>
<td>5.643440E−08</td>
<td></td>
</tr>
<tr>
<td>blake@desktop$</td>
<td>5.963430E−08</td>
<td></td>
</tr>
</tbody>
</table>

*Not an issue with floating-point computations
In concrete terms (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 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 computational environments**

* Not an issue with floating-point computations
** C language is an example, similar issues occur in Python, R, Perl, etc.
In concrete terms (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-built-in && ./a.out
5.963430E-08
```

(due to constant folding**)

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

**We need more than a version number.**

* Not an issue with floating-point computations

** C language is an example, similar issues occur in Python, R, Perl, etc.
Declarative approach (1/3)

```
-guix shell python python-numpy --export-manifest
```

**declarative = configuration file**

The file `my-tools.scm` could contain this declaration:

```scheme
(specifications->manifest
 (list
   "python"
   "python-numpy"))
```

guix package --manifest=my-tools.scm

equivalent to

guix install python python-numpy
Declarative approach (2/3)

Version? We will see later

Language? Domain-Specific Language (DSL) based on Scheme (link)

(= « Lisp functional language » (link))

Declarative vs Imperative (links) (and not passive Data vs active Program)

Declarative programming = functional (OCaml) or dataflow (Lustre) or logic (Prolog) programming
Declarative approach (2/3)

Version? We will see later

Language? Domain-Specific Language (DSL) based on Scheme (link)

(= Lisp functional language » (link))

▶ (Yes (when (= Lisp parentheses) (baroque)))

▶ But continuum:

1. configuration (manifest)
2. package definition (or services)
3. extension
4. the core of Guix is Scheme too

Declarative vs Imperative (links) (and not passive Data vs active Program)

Declarative programming = functional (OCaml) or dataflow (Lustre) or logic (Prolog) programming
Declarative approach (2/3)

Version? We will see later

Language? Domain-Specific Language (DSL) based on Scheme (link)

(= « Lisp functional language » (link))

▶ (Yes (when (= Lisp parentheses) (baroque)))

▶ But continuum:
  1 configuration (manifest)
  2 package definition (or services)
  3 extension
  4 the core of Guix is Scheme too

Guix is flexible for most needs

Declarative vs Imperative (links) (and not passive Data vs active Program)
Declarative programming = functional (OCaml) or dataflow (Lustre) or logic (Prolog) programming
Declarative approach: example of transformation (3/3)

Rube Goldberg machine :-) (link)

(define python "python")

(specifications->manifest
 (append
  (list python)
  (map (lambda (pkg)
         (string-append python "-" pkg))
       (list
        "matplotlib"
        "numpy"
        "scipy"))))

Guix DSL, variables, Scheme et chaîne de caractères.
Declarative approach: example of transformation (3/3)

Rube Goldberg machine :-)

(define python "python")

(specifications->manifest
 (append
  (list python)
  (map (lambda (pkg)
        (string-append python "-" pkg)))
  (list
   "matplotlib"
   "numpy"
   "scipy")))

(specifications->manifest
 (list
  "python"
  "python-matplotlib"
  "python-numpy"
  "python-scipy"))

Guix DSL, variables, Scheme et chaîne de caractères.
Recipe for defining a package

(define python
  (package
    (name "python")
    (version "3.9.9")
    (source ... ) ; points to URL source code
    (build-system gnu-build-system) ; ./configure & make
    (arguments ... ) ; configure flags, etc.
    (inputs (list bzip2 expat gdbm libffi sqlite openssl readline zlib tcl tk))))

Note the terminology (inputs, arguments) as in mathematical function definition

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

What are the root of the graph? Part of the broad bootstrapping (link) problem
Package transformation (1/2)

How to build the package `python` with the compiler GCC@7?

- package = recipe for configuring, building and installing a software
  
  ```
  ./configure && make && make install
  ```

  The recipe defines:
  
  ▶ code source and potentially some ad-hoc modifications (patch)
  
  ▶ build-time tools (compilers, build automation, etc., e.g. gcc, cmake)
  
  ▶ dependencies (= other packages)

  package transformation allows to rewrite them
How to build the package python with the compiler GCC@7?

package = recipe for configuring, building and installing a software

(./configure && make && make install)

The recipe defines:

- code source and potentially some *ad-hoc* modifications (patch)
- build-time tools (compilers, build automation, etc., e.g. gcc, cmake)
- dependencies (= other packages)
How to build the package python with the compiler GCC@7?

package = recipe for configuring, building and installing a software

(./configure && make && make install)

The recipe defines:

- **code source** and potentially some *ad-hoc* modifications (patch)
- **build-time tools** (compilers, build automation, etc., e.g. gcc, cmake)
- **dependencies** (= other packages)

package transformation allows to rewrite them
guix package --help-transformations

--with-source  use SOURCE when building the corresponding package
--with-branch  build PACKAGE from the latest commit of BRANCH
--with-commit   build PACKAGE from COMMIT
--with-git-url build PACKAGE from the repository at URL
--with-patch    add FILE to the list of patches of PACKAGE
--with-latest   use the latest upstream release of PACKAGE
--with-c-toolchain  build PACKAGE and its dependents with TOOLCHAIN
--with-debug-info  build PACKAGE and preserve its debug info
--without-tests build PACKAGE without running its tests
--with-input replace dependency PACKAGE by REPLACEMENT
--with- graft  graft REPLACEMENT on packages that refer to PACKAGE

also available using manifest file
Example state-alice.scm

guix describe -f channels-sans-intro

(list (channel
  (name 'guix)
  (url "https://git.savannah.gnu.org/git/guix.git")
  (branch "master")
  (commit "00ff6f7c399670a76efffb91276dea2633cc130c"))
(channel
  (name 'guix-cran)
  (url "https://github.com/guix-science/guix-cran")
  (branch "master")
  (commit "ab70c9b745a0d60a40ab1ce08024e1ebca8f61b9"))
(channel
  (name 'my-team)
  (url "https://my-forge.my-institute.xyz/my-custom-channel")
  (branch "main")
  (commit "ab70c9b745a0d60a40ab1ce08024e1ebca8f61b9"))}
Alice would like to quickly jump to a productive environment
Blake prefers IPython as interpreter
guix shell -m project-tools.scm # Alice

guix shell -m project-tools.scm python-ipython -- ipython3 # Blake
```
guix shell -m project-tools.scm  # Alice

guix shell -m project-tools.scm python-ipython -- ipython3  # Blake
```

- **--pure**: clear environment variable definitions (from the parent environment)
- **--container**: spawn isolated container (from the rest of the system)
guix shell -m project-tools.scm  # Alice

guix shell -m project-tools.scm python-ipython -- ipython3  # Blake

▶ --pure : clear environment variable definitions (from the parent environment)
▶ --container : spawn isolated container (from the rest of the system)

guix shell -m project-tools.scm python-ipython  # 1.
guix shell -m project-tools.scm python-ipython --pure  # 2.
guix shell -m project-tools.scm python-ipython --container  # 3.
guix shell -m project-tools.scm  # Alice

guix shell -m project-tools.scm python-ipython -- ipython3  # Blake

- **--pure**: clear environment variable definitions (from the parent environment)
- **--container**: spawn isolated container (from the rest of the system)
- **--development**: include dependencies of the package

---

guix shell -m project-tools.scm python-ipython  # 1.
guix shell -m project-tools.scm python-ipython --pure  # 2.
guix shell -m project-tools.scm python-ipython --container  # 3.

**Bonus**: guix shell emacs git git:send-email --development guix
Wait, now we would like to build and share isolated containers.

How to create a container?

Example: Alice wants to share a Docker image
How to capture an environment for sharing? with a container, right?

Container = smoothie :-) 

- How to build the container? Dockerfile?
- How the binaries included inside the container are they built?
How to capture an environment for sharing? with a container, right?

Container = smoothie :-) 

- How to build the container? Dockerfile?
- How the binaries included inside the container are they built?

FROM amd64/debian:stretch
RUN apt-get update && apt-get install git make curl gcc g++ ...
RUN curl -L -O https://... && ... && make -j 4 && ...
RUN git clone https://... && ... && make ... /usr/local/lib/libopenblas.a ...

(seen for nightly automation; maybe used in production?)

Considering one Dockerfile at time t, how to rebuild the image at time t’?
What is a pack?

pack = collection of packages stored in one archive format

What is the aim of a pack?

- Alice provides « everything » to Blake,
- Blake does not have Guix but will run the exact same environment.
What is a *pack*?

*pack* = collection of packages stored in one archive format

What is the aim of a *pack*?

- Alice provides « everything » to Blake,
- Blake does not have Guix but will run the exact same environment.

What does it mean an archive format?

- tar (*tarballs*)
- Docker
- Singularity
- Debian binary package `.deb`
What does it mean « everything »?

Blake needs *transitive closure* (= all the dependencies)

```
$ guix size python-numpy --sort=closure
store item          total  self
python-numpy-1.20.3  301.5  23.6  7.8%
...                  ...
python-3.9.9         155.3  63.7  21.1%
openblas-0.3.18      152.8  40.0  13.3%
...                  ...
total: 301.5 MiB
```

guix pack builds this archive containing « everything »
Alice builds a *pack* using the format Docker

```
guix pack --format=docker -m project-tools.scm
```

then shares this Docker container (using some *registry* or else).
Alice builds a pack using the format Docker

```
guix pack --format=docker -m project-tools.scm
```

then shares this Docker container (using some registry or else).

Blake does not run (yet?) Guix

```
$ docker run -ti projet-alice python3
Python 3.9.9 (main, Jan 1 1970, 00:00:01)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

and is running the exact same computational environment as Alice.
Alice builds a *pack* using the format Docker:

```
$ guix pack --format=docker -m project-tools.scm
```

then shares this Docker container (using some *registry* or else).

Blake does not run (yet?) Guix:

```
$ docker run -ti projet-alice python3
Python 3.9.9 (main, Jan 1 1970, 00:00:01)
 [GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

and is running the exact same computational environment as Alice.

---

**How to rebuild the exact same Docker *pack* using Guix on 2 machines at 2 different moments** (link)
Summary, guix pack is

_Agnostic concerning the « container » format_

- tar (*tarballs*)
- Docker
- Singularity
- Debian binary package .deb

- relocatable binaries
- **without** Dockerfile
- using squashfs
- **without** debian/rule (experimental)

_Flexible to contexts_

_the key point is the full control of binaries going inside the container_