GNU Guix and Open science, a crush?

Simon Tournier

simon@tournier.info

February, 4th, 2023

https://hpc.guix.info

day job: Research Eng.

Université Paris Cité
2013: *Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma* (Nature Genetics) (link) → Conclusion A

2011: SF3B1 mutations are associated with alternative splicing in uveal melanoma (Cancer Discovery) (link) → Conclusion B

2022: 35 Master students independently reanalyse → Conclusion C

Conclusion A \≠ Conclusion B \≠ Conclusion C

Ah, maybe it's an unique case?

FOSDEM 2023 GNU Guix and Open science, a crush? 1 / 26
2013: Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma

(Nature Genetics) (link)

→ Conclusion A

2013: SF3B1 mutations are associated with alternative splicing in uveal melanoma

(Cancer Discovery) (link)

→ Conclusion B

with the same data

Conclusion A \(\neq\) Conclusion B \(\neq\) Conclusion C

Ah, maybe it's an unique case?
2013: *Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma*  
(Nature Genetics) (link)

→ **Conclusion A**

2013: *SF3B1 mutations are associated with alternative splicing in uveal melanoma*  
(Cancer Discovery) (link)

→ **Conclusion B**

2022: 35 Master students independently *reanalyse*  
→ **Conclusion C**

**Conclusion A** \( \neq \) **Conclusion B** \( \neq \) **Conclusion C**

Ah, maybe it's an unique case?

FOSDEM 2023

GNU Guix and Open science, a crush?
2013: Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma
(Nature Genetics) (link)
→ Conclusion A

2013: SF3B1 mutations are associated with alternative splicing in uveal melanoma
(Cancer Discovery) (link) with the same data
→ Conclusion B

2022: 35 Master students independently reanalyse
→ Conclusion C with the same data

Conclusion A ≠ Conclusion B ≠ Conclusion C

Ah, maybe it’s an unique case?
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?

Open Science helps

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

security and “open science” are two sides of the same coin
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

- 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

- 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

is 15 billion parameters declarative and minimalist computing?
**1905:** Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen

- 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

- 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 declarative and minimalist computing?*
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?
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 + \textit{Numerical processing}

Science in the digital age:

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

How to \textit{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?

that’s Guix!

“Open science”, a tautology?
“Scientific” knowledge is based on

- being able to read the result
- being able to study all the details and adapt them for your own research
- being able to share with peers
- being able to share your own results adapted from the others
“Scientific” knowledge is based on

- being able to read the result
- being able to study all the details and adapt them for your own research
- being able to share with peers
- being able to share your own results adapted from the others

Wait, it looks like the *free software* definition

*free software and scientific research are two sides of the same coin*
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?)

▶ 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: (universal?)

▶ 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”)
Computational environment

Alice says: “using this data, you need this C file and GCC@11.2.0 to run my analysis”

- What is source code?
- What are the tools required for building?
- What are the tools required at run time?
- And recursively for each tool...
Alice says: “using this data, you need this C file and GCC@11.2.0 to run my analysis”

- 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.
Computational environment

Alice says: “using this data, you need this C file and GCC@11.2.0 to run my analysis”

- 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?

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
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

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

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...

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

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

Easy to try
Guix: computational environment manager on steroids

- A package manager (as APT, Yum, etc.)
- Transactional and declarative
- Which produces shareable packs
- Which produces isolated virtual machines
- Used to build a whole Linux distribution
- Used also as a Scheme library

how Guix is helping me

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

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

Easy to try
Guix, really yet another package manager?

- Package install/remove without any special privilege
- Declarative management (declarative = configuration file)
- Transactional (= no « broken » state)
- Binary substitutes (= fetch pre-compiled components)

- Isolated environment on-the-fly (guix shell --container)
- Factory for images (guix pack -f docker)

The profiles allow to install several versions.

(profile ≈ “environment à la virtualenv”)
Interesting features, but why is the computational environment controlled \((\text{reproducible})\)?

We need to talk about version!

Example: Alice and Blake are collaborating
When Alice says “GCC at version 11.2.0”

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?

```bash
$ 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

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
- 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
Collaboration in action

Guix is helping me (1/2)

<table>
<thead>
<tr>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>describes her environment:</td>
</tr>
<tr>
<td>▶ the list of the tools using the file <code>manifest.scm</code>, spawns her environment e.g.,</td>
</tr>
<tr>
<td>guix shell -m manifest.scm</td>
</tr>
</tbody>
</table>
Alice describes her environment:

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

- the revision (Guix itself and potentially all the other channels)
  
  ```sh
  guix describe -f channels > state-alice.scm
  ```
Guix is helping me (1/2)

Collaboration in action

**collaborate = share one computational environment**

Alice

describes her environment:

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

- the revision (Guix itself and potentially all the other channels)
  
  ```bash
  guix describe -f channels > state-alice.scm
  ```

Blake spawns the same computational environment from these two files:

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

Guix is helping me (1/2)

collaborate = share one computational environment \Rightarrow share one specific graph

Alice

describes her environment:

- the list of the tools using the file manifest.scm, spawns her environment e.g.,
  \texttt{guix shell -m manifest.scm}
- the revision (Guix itself and potentially all the other channels)
  \texttt{guix describe -f channels > state-alice.scm}
Collaboration in action

Guix is helping me (1/2)

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.,
  ```shell
guix shell -m manifest.scm
  ```
- the revision (Guix itself and potentially all the other channels)
  ```shell
guix describe -f channels > state-alice.scm
  ```
- then shares these two files: `state-alice.scm` and `manifest.scm`
Collaboration in action

Guix is helping me (1/2)

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
Collaboration in action

Guix is helping me (1/2)

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.,
  ```shell
  guix shell -m manifest.scm
  ```
- the revision (Guix itself and potentially all the other channels)
  ```shell
  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

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

Carole can also reproduce the same environment as Blake (and thus Alice)
Reproducible = jump to different states

guix time-machine

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

(= source code long term archive)

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 as fallback if 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)
Wait, my collaborators do not run Guix

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?

A pack is a 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`
- RPM package `.rpm`
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 manifest.scm
```

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

```bash
guix pack --format=docker -m manifest.scm
```

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

Blake does not run (yet?) Guix

```bash
$ docker run -ti project-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.
Building a *pack* for sharing  

guix pack -f docker

- Alice builds a *pack* using the format Docker

```bash
guix pack --format=docker -m manifest.scm
```

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

- Blake does not run (yet?) Guix

```bash
$ docker run -ti project-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`
- RPM package `.rpm`

- relocatable binaries
- **without** Dockerfile
- using squashfs
- **without** debian/rule (experimental)
- **without** RPM machinery (patch#61255)

Flexible to contexts

*the key point is the full control* of binaries going inside the container
Summary
Guix: computational environment manager on steroids

Guix is helping me

- 3 commands + 2 files for sharing computational environments
  - guix shell -m manifest.scm
  - guix time-machine -C channels.scm -- subcommand
  - guix describe -f channels > channels.scm
- "packing factory" for sharing computational environments on infrastructure without Guix

Guix precisely controls the complete implicit graph of configurations

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*
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

Guix

The ReScience Journal
Thanks:

- Guix community. You are awesome!
- Ludovic Courtès
- Mathieu Othacehe

Questions?

guix-science@gnu.org

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

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

These slides are archived.
(Software Heritage id swh:1:dir:2494ed717a50f1b9ee33f0c8881017d7cd8e93fd)
Appendix

More about

- Declarative approach
- What a package looks like
- Package transformation
- What the file capturing the state looks like
- Extended environment, isolated
Declarative approach (1/3)

guix shell python python-numpy --export-manifest

declarative = configuration file

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

```
(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 :-} \(\text{link}\)

\[
\begin{align*}
&\text{(define } \text{python } "\text{python}"	ext{)} \\
&\text{(specifications->manifest} \\
&\quad \text{(append} \\
&\quad \quad \text{(list } \text{python}) \\
&\quad \quad \text{(map } (\text{lambda } (\text{pkg}) \\
&\quad \quad \quad \text{(string-append } \text{python } "-" \text{ pkg})) \\
&\quad \quad \text{(list} \\
&\quad \quad \quad "\text{matplotlib}" \\
&\quad \quad \quad "\text{numpy}" \\
&\quad \quad \quad "\text{scipy}"))
\end{align*}
\]

Guix DSL, \textit{variables}, Scheme et chaîne de caractères.
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")))

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

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

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

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

*also available using manifest file*
(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

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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>--pure</td>
<td>clear environment variable definitions (from the parent environment)</td>
<td></td>
</tr>
<tr>
<td>--container</td>
<td>spawn isolated container (from the rest of the system)</td>
<td></td>
</tr>
<tr>
<td>--development</td>
<td>include dependencies of the package</td>
<td></td>
</tr>
</tbody>
</table>

---

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