Reproducible Research at the Cloud Era
Overview, Hands-on and Open Challenges

Sébastien Varrette, PhD
Parallel Computing and Optimization Group (PCOG),
University of Luxembourg (UL), Luxembourg

Before the tutorial starts: Visit
https://goo.gl/19mCsM

for preliminary setup instructions!
Summary

1 Introduction and Motivating Examples

2 Reproducible Research
   Easy-to \{read\|take\|share\} Docs
   Sharing Code and Data
   Mastering your [reproducible] environment

3 Conclusion
About me

- **2003 – 2007**: PhD between INP Grenoble & UL → *Security in Large Scale Distributed Systems: Authentication and Result Checking*

- **2007 – now**: Research Associate at UL → Part of the PCOG Team led by Prof. P. Bouvry → Manager of the UL High Performance Computing Facility
  ✓ ≃ 197 TFlops (2017), 5.844 PB, 4 sysadmins

**Research Interests: Distributed Computing Platforms**

- Security (crash/cheating faults, obfuscation) in DGVCS
- Performance of HPC/cloud platforms
  → Energy Efficiency, Performance, Cost…
Disclaimer: Acknowledgements

A large part of these slides were courtesy borrowed, with permission, from:

- Lucas Nussbaum (INRIA, Univ. Lorraine)
- Arnaud Legrand (INRIA, Univ. Grenoble)
- Valentin Plugaru (Univ. of Luxembourg)
- and many others...

In particular, to know more about Reproducible Research:

- Webinars on Reproducible Research [https://github.com/alegrand/RR_webinars](https://github.com/alegrand/RR_webinars)
- Reproducible build [https://reproducible-builds.org/](https://reproducible-builds.org/)
  - initiative of various free software projects
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 – 10:00</td>
<td>Reproducible Research in Computer Science</td>
</tr>
<tr>
<td>10:00 – 10:30</td>
<td>Hands-On: Build these slides using Vagrant</td>
</tr>
<tr>
<td>10:30 – 11:00</td>
<td><strong>Coffee Break</strong></td>
</tr>
<tr>
<td>11:00 – 11:30</td>
<td>Hands-On: Reproducible Software Environment with Easybuild</td>
</tr>
<tr>
<td></td>
<td>Hands-On: Docker</td>
</tr>
<tr>
<td></td>
<td>Reproducible Results</td>
</tr>
<tr>
<td>12:15 –</td>
<td><strong>Lunch</strong></td>
</tr>
</tbody>
</table>
Tutorial Pre-Requisites / Setup

- Create (if need) accounts for the **cloud services** we will use:
  - Github, Vagrant Cloud and Docker Hub
- Install **mandatory software**, *i.e.* (apart from Git):
  - Virtual Box
  - Vagrant
  - Docker
- Check installed software and download the boxes we will use:

  ```bash
  $> git clone https://github.com/Falkor/RR-tutorials.git
  $> cd RR-tutorials
  $> make setup
  $> vagrant up && docker pull ubuntu:14.04  # might take some time...
  ```
Introduction and Motivating Examples

Summary

1. Introduction and Motivating Examples

2. Reproducible Research
   - Easy-to \{read|take|share\} Docs
   - Sharing Code and Data
   - Mastering your [reproducible] environment

3. Conclusion
Two classical approaches for validation:

- **Formal**: equations, proofs, etc.
- **Experimental**, on a scientific instrument

Often a mix of both:

- In Physics
- In Computer Science

Quite a lot of **formal** work in Computer Science

But also quite a lot of experimental validation

- Distributed computing, networking
  - testbeds: IoT-LAB, Grid'5000…
- Language/image processing ↦ evaluations using large corpuses
Two classical approaches for validation:

- **Formal**: equations, proofs, etc.
- **Experimental**, on a scientific instrument

Often a mix of both:

- In Physics
- In Computer Science

Quite a lot of **formal** work in Computer Science

But also quite a lot of experimental validation

- Distributed computing, networking
  - testbeds: IoT-LAB, Grid'5000...
- Language/image processing ~ evaluations using large corpuses

How good are we at performing experiments?
(Poor) State of Experimentation in CS

- **1994**: survey of 400 papers\(^1\)
  - among published CS articles in ACM journals
  - 40\%-50\% of those requiring an experimental validation had none

- **1998**: survey of 612 papers\(^2\)
  - too many papers have no experimental validation at all
  - too many papers use an informal (assertion) form of validation
  - 2009 update: situation is improving\(^3\)

---


\(^3\) Marvin V. Zelkowitz. “An update to experimental models for validating computer technology”. In: *J. Syst. Softw.* 82.3 (Mar. 2009), pages 373–376.
Introduction and Motivating Examples

(Poor) State of Experimentation in CS

- Most papers do not use even basic statistical tools
  - Papers published at the Europar conference

<table>
<thead>
<tr>
<th>Year</th>
<th>#Papers</th>
<th>With error bars</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>89</td>
<td>5</td>
<td>5.6%</td>
</tr>
<tr>
<td>2008</td>
<td>89</td>
<td>3</td>
<td>3.4%</td>
</tr>
<tr>
<td>2009</td>
<td>86</td>
<td>2</td>
<td>2.4%</td>
</tr>
<tr>
<td>2010</td>
<td>90</td>
<td>6</td>
<td>6.7%</td>
</tr>
<tr>
<td>2011</td>
<td>81</td>
<td>7</td>
<td>8.6%</td>
</tr>
<tr>
<td>2007-2001</td>
<td>435</td>
<td>23</td>
<td>5.3%</td>
</tr>
</tbody>
</table>

---

4 Study carried out by E. Jeannot.

Sebastien Varrette (University of Luxembourg)
Introduction and Motivating Examples

(Poor) State of Experimentation in CS

2007: Survey of simulators used in P2P research\(^5\)

\[ \rightarrow \] 287 papers surveyed on P2P networking subject
\[ \rightarrow \] 141 of these papers reports the use of a simulator

- 30% use a custom tool
- 50% don’t report the used tool!

Introduction and Motivating Examples

(Poor) State of Experimentation in CS

- **2015**: 601 papers from ACM conferences and journals analysed\(^6\)
  - **Obj.**: attempt to locate any source code that backed up the published results; *if found, try to build the code.*
  - **EM\(^\text{no}\)** *(146 papers!)*: code cannot be provided!
  - **Original study**: 80% of non reproducible work

Introduction and Motivating Examples

And in Other Sciences?

- **Biology**: Increase in **retracted papers**\(^7\),
  - **Fraud** (data fabrication or falsification)
  - **Error** (plagiarism, scientific mistake, ethical problems)
    - see also Reproducibility: A tragedy of errors\(^8\)
    - cf. Duke University scandal with scientific misconduct on lung cancer
  - **High number of failing clinical trials**
    - Do We Really Know What Makes Us Healthy?, 2007
    - Lies, Damned Lies, and Medical Science, 2010

- **Psychology**:
  - unreplicable study about extrasensory perception (ESP)

- **Machine Learning**: Trouble at the lab, The Economist, 2013
  
  *According to some estimates, three-quarters of published scientific papers in the field of machine learning are bunk because of this “overfitting”.* Sandy Pentlan, MIT

---


**And in Other Sciences?**

- **Medicine:** Study shows lower fertility for mices exposed to transgenic maize (AFSSA report\(^9\))
  - Several calculation errors have been identified
  - led to a false statistical analysis & interpretation

\(^9\) Opinion of the French Food Safety Agency (Afssa) on the study by Velimirov et al.
And in Other Sciences?

- **Medicine**: Study shows lower fertility for mices exposed to transgenic maize (AFSSA report\(^9\))
  - Several calculation errors have been identified
  - led to a false statistical analysis & interpretation

- **Physics**: CERN / OPERA Experiment (2011)
  - faster-than-light neutrinos
  - ✓ People started gossiping about relativity violation . . .
  - ➪ caused by timing system failure in 2012

\(^9\) Opinion of the French Food Safety Agency (Afssa) on the study by Velimirov et al.
Introduction and Motivating Examples

And in Other Sciences?

- **Medicine**: Study shows lower fertility for mices exposed to transgenic maize (AFSSA report\(^9\))
  - Several calculation errors have been identified
  - led to a false statistical analysis & interpretation

- **Physics**: CERN / OPERA Experiment (2011)
  - faster-than-light neutrinos
    - People started gossiping about relativity violation...
    - caused by timing system failure in 2012

- 🙁: Not everything is perfect
- 😊: But some errors are properly identified
  - Stronger experimental culture in other (older?) sciences?
  - Long history of costly experiments, scandals, ...

---

\(^9\) Opinion of the French Food Safety Agency (Afssa) on the study by Velimirov et al.
Introduction and Motivating Examples

What About You (as Reviewer)?

“This may be an interesting contribution but...”

- This average value must hide something
- As usual, there is no confidence interval,
  
  I wonder about the variability and whether the difference is significant or not
Introduction and Motivating Examples

What About You (as Reviewer)?

“This may be an interesting contribution but…”

- This **average value** must hide something
- As usual, there is no **confidence interval**,  
  ➔ I wonder about the variability and whether the difference is **significant** or not

- Why is this graph in **logscale**? How would it look like otherwise?
- That can’t be true, I’m sure they removed some points
- The authors decided to show only a subset of the data.  
  ➔ I wonder what the rest looks like
Introduction and Motivating Examples

What About You (as Reviewer)?

“This may be an interesting contribution but...”

- This **average value** must hide something
- As usual, there is no **confidence interval**,
  \[ \rightarrow I\ wonder\ about\ the\ variability\ and\ whether\ the\ difference\ is\ **significant**\ or\ not \]

- Why is this graph in **logscale**? How would it looks like otherwise?
- That can’t be true, I’m sure they **removed some points**
- The authors decided to show only a subset of the data.
  \[ \rightarrow I\ wonder\ what\ the\ rest\ looks\ like \]

- There is no label/legend/... What is the **meaning of this graph**?
  \[ \rightarrow If\ only\ I\ could\ access\ the\ generation\ script \]
What About You (as Author)?

- I thought I used the same parameters...  
  but I’m getting different results!
Introduction and Motivating Examples

What About You (as Author)?

- I thought I used the *same parameters*...  
  → but I’m *getting different results*!

- The new student wants to compare with my *last year’s method*
- My advisor asked me whether I took care of setting this or this...  
  → but *I can’t remember*
- The damned fourth reviewer asked for a major revision...  
  → he wants me to *change figure 3 😞*
Introduction and Motivating Examples

What About You (as Author)?

- I thought I used the same parameters... 
  but I’m getting different results!

- The new student wants to compare with my last year’ method
- My advisor asked me whether I took care of setting this or this... 
  but I can’t remember
- The damned fourth reviewer asked for a major revision...
  he wants me to change figure 3 😞

Which code / data set did I use to generate this figure?
- It worked yesterday!
- 6 months later: just why did I do that?
Introduction and Motivating Examples

Why is it Hard to Reproduce? (any Scientific Work)

- **Human error:**
  - Experimenter bias
  - Programming errors or data manipulation mistakes
  - Poorly selected statistical test

- There is just no real incentive in doing so:
  - Legal barriers, copyright
  - Competition issue
  - Publication bias
  - Rewards for positive/novel results, not for consolidating results

Many ongoing discussions in US researchware, bibliometry, ...

only the idea matters, not the gory details...
Introduction and Motivating Examples

Why is it Hard to Reproduce? (any Scientific Work)

- **Human error:**
  - Experimenter bias
  - Programming errors or data manipulation mistakes
  - Poorly selected statistical test

- There is just no real incentive in doing so:
  - Legal barriers, copyright
  - Competition issue
  - Publication bias
  - Rewards for positive/novel results, not for consolidating results

- **Technical difficulty:**
  - Hardware and software evolve too quickly. It’s not worth it
  - No resources for storing so much data/information
  - Lack of easy-to-use tools
Summary

1. Introduction and Motivating Examples

2. Reproducible Research
   - Easy-to \{read|take|share\} Docs
   - Sharing Code and Data
   - Mastering your [reproducible] environment

3. Conclusion
Reproducible Research

Reproducible Research Movement

- Originated mainly in **Computational Sciences**
  - Computational biology, data-intensive physics, etc.
- Explores methods and tools to enhance experimental practices
  - Enable others to reproduce and build upon one’s work

- **Nothing New**
  - Fundamental basis of the scientific method
  - K. Poppler, 1934: *non-reproducible single occurrences are of no significance to science*
Terminology varies\textsuperscript{10}

\begin{itemize}
  \item \textbf{Replicability} $\sim$ same result
  \item \textbf{Reproducibility} $\sim$ same scientific conclusions
\end{itemize}

\begin{tikzpicture}[scale=0.8]
  \node at (0,0) {	extbf{Replicability}}; \node at (12,0) {	extbf{Reproducibility}};
  \draw[->] (0,0) -- (6,0) node[midway,above] {Reproduction of the original results using the same tools};
  \draw[->] (6,0) -- (12,0) node[midway,above] {Reproduction using different software, but with access to the original code};
  \draw[->] (0,-6) -- (6,-6) node[midway,above] {Completely independent reproduction based only on text description, without access to the original code};
  \draw[->] (6,-6) -- (12,-6) node[midway,above] {by the original author on the same machine};
  \draw[->] (6,-6) -- (12,-6) node[midway,above] {by someone in the same lab/using a different machine};
  \draw[->] (6,-6) -- (12,-6) node[midway,above] {by someone in a different lab};
\end{tikzpicture}

Reproducibility (Wikipedia)

- the ability of an entire experiment or study to be reproduced, either by the researcher or by someone else working independently.
- One of the main principles of the scientific method.

For an experiment involving software, reproducibility means:
- open access to the scientific article describing it
- open data sets used in the experiment
- source code of all the components
- environment of execution
- stable references between all this
Reproducible Research

The Research Pipeline

Author

Scientific Question

Protocol
(Design of Experiments)

Nature/System/...

Reader

Published Article

Courtesy of A. Legrand, inspired by Roger D. Peng’s lecture on reproducible research, May 2014

Sebastien Varrette (University of Luxembourg)

Reproducible Research at the Cloud Era
Reproducible Research

The Research Pipeline

Author

Measured Data → Analytic Data → Computational Results → Published Article

Figures

Tables

Numerical Summaries

Text

Nature/System/...

Protocol
(Design of Experiments)

Scientific Question

Reader

Courtesy of A. Legrand, inspired by Roger D. Peng’s lecture on reproducible research, May 2014
Reproducible Research

The Research Pipeline

Author

Measured Data → Analytic Data → Computational Results

Experiment Code

Analytic Code

Presentation Code

Published Article

Figures

Tables

Numerical Summaries

Text

Nature/System/...

Scientific Question

Protocol
(Design of Experiments)

Processing Code

Analysis Code

Experiment Code
(workload injector, VM recipes, ...)

Reader

Courtesy of A. Legrand, inspired by Roger D. Peng’s lecture on reproducible research, May 2014
Reproducible Research

The Research Pipeline

Try to keep track of the whole chain
= Provenance tracking

Courtesy of A. Legrand, inspired by Roger D. Peng’s lecture on reproducible research, May 2014
Analysis is generally not very domain-specific

Courtesy of A. Legrand, inspired by Roger D. Peng’s lecture on reproducible research, May 2014
The Distributed/Cloud Computing point-of-view:

- Experiments remains the HARD part and is very domain-specific
  - Rely on large, distributed, hybrid, prototype hardware/software
  - Measure execution times (makespans, traces, ...)
  - Many parameters, very costly and hard to reproduce

Figure 1. Experimental Diagram

Figure 2. Experimental Mess
Environment Management

Controlling/Providing your Environment

- An environment is a set of tools and materials that permits a complete reproducibility of part/whole experiment process.
Controlling/Providing your Environment

- An environment is a set of tools and materials that permits a complete reproducibility of part/whole experiment process.

Q1: How to describe/provide the software environment used?

“I used OpenFOAM with OpenMPI on Debian”

- Obvious solution: Virtual Machines
  - Easy way to [automatically] test recipes
  - Yet provides only the final result, not the logic behind
Accurate, organized and easy-to\{read\|take\|share\} Docs

→ Markdown, mkdocs, org-mode, Read the Docs...
Accurate, organized and easy-to-read Docs
   ➔ Markdown, mkdocs, org-mode, Read the Docs...

Sharing Code and Data
   ➔ git, Github, Bitbucket, Gitlab...
Accurate, organized and easy-to-read Docs

- Markdown, mkdocs, org-mode, Read the Docs...

Sharing Code and Data

- git, Github, Bitbucket, Gitlab...

Mastering your environment clean and automated by:

- Using common building tools (make, cmake etc.)
- Using a constrained environment
  - Sandboxed Ruby/Python, Vagrant, Docker
- Automate its building through cross-platform recipes
- Automatically test your recipes for Environment configuration
Accurate, organized and easy-to-read Docs

- Markdown, mkdocs, org-mode, Read the Docs...

Sharing Code and Data

- git, Github, Bitbucket, Gitlab...

Mastering your environment clean and automated by:

- Using common building tools make, cmake etc.
- Using a constrained environment
  - Sandbox Ruby/Python, Vagrant, Docker
- Automate building through cross-platform recipes
- Automatically test your recipes for Environment configuration

All covered in this tutorial!
Summary

1. Introduction and Motivating Examples

2. Reproducible Research
   Easy-to {read|take|share} Docs
   Sharing Code and Data
   Mastering your [reproducible] environment

3. Conclusion
Reproducible research assumes accurate and organized Docs

You need to document your:

- **Hypotheses**: keep track of your ideas/line of thoughts
- **Experiments**: details on how and why an experiment was run
  - including failed or ambiguous attempts.
- **Initial analysis or interpretation** of these experiments
  - was the outcome conform to the expectation or not?
  - does it (in)validate the hypothesis?
- **Organization**: keep track of things to do/ x/test/improve

**Structure**:

- General information about the document
- **commonly used commands** and how to set up experiments
- Experiment results
  - by date (tags)
  - by experiment campaigns (date/time)
Recommandation

- Plain-text with **Markdown** syntax
  - Easy to **track over Git** (text files, **not** Word/RFT etc.)
  - Easy to **export** to any format using **pandoc** / **multimarkdown**
  - Supports online/offline Wikis / Blogging platforms

- **Focus on writing**, viewers for all platform
  - Mac OS: MOU, Marked 2
  - Linux: Remarkable, Retext
  - Windows: MarkdownPad, Remarkable

- Git Based **Markdown** Blogging
  - Octopress, Jekyll
Git-based Markdown Wiki

- Permits to work offline
  - Gollum, as embedded in GitLab
    - run gollum (from root directory)
  - http://localhost:4567

Recommandation: MkDocs

- Better for **Hierarchical structure** of the docs
  - fully configured by mkdocs.yml and files in docs/
  - local [interpreted] site: mkdocs serve (from root directory)
    - http://localhost:8000
- compliant with **Read the Docs**
  - trigger automatic doc rebuild upon [git] push
  - cf http://rr-tutorials.readthedocs.io/ 😊
Mkdocs Workflow

```bash
$ mkdocs new  # initialize 'mkdocs.yml' and docs/ directory
```
Mkdocs Workflow

$> \texttt{mkdocs new} \quad \# \text{initialize 'mkdocs.yml' and docs/ directory}

\begin{verbatim}
# mkdocs.yml -- MkDocs configuration, all *.md files relative to docs/
site_name: My Environment Documentation

pages:
- Home: 'index.md'
- Tools:
  - SSH: 'tools/ssh.md'
  - Git: 'tools/git.md'
- Configuration:
  - CA Certificates: 'config/certificates/README.md'

theme: readthedocs
\end{verbatim}
Mkdocs Workflow

$> mkdocs new  # initialize 'mkdocs.yml' and docs/ directory

# mkdocs.yml -- MkDocs configuration, all *.md files relative to docs/
site_name: My Environment Documentation
pages:
- Home: 'index.md'
- Tools:
  - SSH: 'tools/ssh.md'
  - Git: 'tools/git.md'
- Configuration:
  - CA Certificates: 'config/certificates/README.md'
theme: readthedocs

$> mkdocs serve  # Run LOCAL builtin server http://localhost:8000
Hands-On 1: Markdown & MkDocs

Your Turn!

http://rr-tutorials.readthedocs.io/en/latest/hands-on/docs/

- Easy-to-{Read | Take | Share} Docs with MkDocs
  - installation of MkDocs
  - initialization
  - Markdown basis
  - Local serve

http://www.mkdocs.org/#installation
mkdocs new .
mkdocs serve
Summary

1 Introduction and Motivating Examples

2 Reproducible Research
   Easy-to {read|take|share} Docs
   Sharing Code and Data
   Mastering your [reproducible] environment

3 Conclusion
What kinds of systems are available?

- **Good**: The cloud - Dropbox, Google Drive, Figshare...
- **Better**: Version Control systems (VCS)
  - SVN, Git and Mercurial
- **Best**: Version Control Systems on the Public/Private Cloud
  - GitHub, Bitbucket, Gitlab
What kinds of systems are available?

- **Good**: The cloud
  - Dropbox, Google Drive, Figshare...
- **Better** - Version Control systems (VCS)
  - SVN, Git and Mercurial
- **Best** - Version Control Systems on the Public/Private Cloud
  - GitHub, Bitbucket, Gitlab

**Which one?**

- Depends on the level of privacy you expect
  - ... but you probably already know these tools 😊
- Few handle GB files...
Centralized VCS – CVS, SVN
Centralized VCS – CVS, SVN

Computer A
- Checkout
  - File

Computer B
- Checkout
  - File

Central VCS Server
- Version Database
  - Version 3
  - Version 2
  - Version 1
Everybody has the full history of commits
R reproducible research

Tracking changes (most VCS)

Checkins over Time

C1

file A

file B

file C
Reproducible Research

Tracking changes (most VCS)

Checkins over Time

C1 → file A → Δ1 → file B → file C → Δ1 → C2
Reproducible Research

Tracking changes (most VCS)

Checkins over Time

C1 -> C2 -> C3

file A -> Δ1

file B

file C -> Δ1 -> Δ2
Reproducible Research

Tracking changes (most VCS)

Checkins over Time

- C1
- C2
- C3
- C4

File A

Δ1 -> Δ2

File B

Δ1

File C

Δ1 -> Δ2
Reproducible Research

Tracking changes (most VCS)

Checkins over Time

C1 → Δ1 → Δ2
C2 → Δ3
C3 → Δ1
C4 → Δ2
C5

file A

file B

file C
Tracking changes (most VCS)

Checkins over Time

delta storage

C1

Δ1

file A

Δ2

C2

Δ1

file B

Δ2

Δ1

C3

Δ1

file C

Δ2

Δ1

C4

Δ2

Δ3

C5

Δ2
Reproducible Research

Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

C1 → Δ1 → Δ2
file A

C2 → Δ1
file B

C3 → Δ2

C4 → Δ2

C5 → Δ3

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

Checkins over Time

C1
file A
Δ1
Δ2
C2
file B
Δ1
Δ2
Δ1
Δ2
Δ3
C3
file C
Δ1
Δ2
Δ1
Δ2
Δ3
C4
Δ1
Δ2
Δ1
Δ2
Δ3
C5

delta storage

snapshot (DAG) storage

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

delta storage

snapshot (DAG) storage

Checkins over Time

file A

Δ1

Δ2

file B

Δ1

Δ2

file C

Δ1

Δ2

Δ3

Checkins over Time

A

A1

A1

B

B

B

C

C1

C1

C2

C2

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

delta storage

snapshot (DAG) storage

Checkins over Time

C1 → C2 → C3 → C4 → C5

file A → Δ1 → Δ2

file B → Δ1 → Δ2

file C → Δ1 → Δ2

Checkins over Time

C1 → C2 → C3

A → A1

B → B

C → C1 → C2
Reproducible Research

Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

Checkins over Time

file A
Δ1
Δ2

file B
Δ1
Δ2

file C
Δ1
Δ2
Δ3

delta storage

snapshot (DAG) storage

file A
file B
file C

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

Checkins over Time

delta storage

file A

file B

file C

Checkins over Time

snapshot (DAG) storage

A

B

C

A1

B

C1

A1

B

C2

A2

B1

C2

A2

B2

C3

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Tracking changes (Git)

Checkins over Time

delta storage

file A

Δ1

Δ2

file B

Δ1

Δ2

file C

Δ1

Δ3

snapshot (DAG) storage

C1

A

A1

A2

B

B1

B2

C

C1

C2

C3

C

C2

C3

Sebastien Varrette (University of Luxembourg)
Reproducible Research

VCS Taxonomy

- delta storage
  - local
  - rcs
  - Mac OS File Versions
  - centralized
  - cvs
  - Subversion
  - distributed
  - mercurial
  - hg

- snapshot (DAG) storage
  - local
    - cp -r
    - rsync
    - duplication
  - centralized
  - bontmia
    - backupninja
    - duplication
  - distributed
    - git
    - bazaar
    - bzr
    - time machine
    - bitkeeper
Reproducible Research

Git at the heart of RR

http://git-scm.org
(Reference) web-based Git repository hosting service

Set up Git
1. Fork repository
3. Work together

Create Repository
2. Work together
4. Work together
So what makes Git so useful?

(almost) Everything is local

- everything is fast
- every clone is a backup
- you work **mainly offline**

Ultra Fast, Efficient & Robust

- Snapshots, not patches (deltas)
- **Cheap branching and merging**
  - Strong support for thousands of parallel branches
- Cryptographic integrity everywhere
Other Git features

- **Git doesn’t delete**
  - **Immutable** objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    - ✓ Recovery can be tricky though
Other Git features

- **Git doesn’t delete**
  - Immutable objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    - ✓ Recovery can be tricky though

**Git Tools / Extension**

- cf. Git submodules or subtrees
- Introducing git-flow
  - workflow with a strict branching model
  - offers the git commands to follow the workflow

  ```
  $> git flow init
  $> git flow feature { start, publish, finish } <name>
  $> git flow release { start, publish, finish } <version>
  ```
## Installation on Linux / Mac OS

```bash
$> apt-get install git-core git-flow  # On Debian-like systems
$> yum install git gitflow          # On CentOS-like systems
$> brew install git git-flow        # On Mac OS, using Homebrew
```
Installation on Linux / Mac OS

```bash
$> apt-get install git-core git-flow # On Debian-like systems
$> yum install git gitflow # On CentOS-like systems
$> brew install git git-flow # On Mac OS, using Homebrew
```

Installation on Windows

- Incl. Git Bash/GUI & Shell Integration
  - install **Git bash** + command prompt
  - select checkout windows / commit unix
Installation on Linux / Mac OS

$> apt-get install git-core git-flow
$> yum install git gitflow
$> brew install git git-flow

# On Debian-like systems
# On CentOS-like systems
# On Mac OS, using Homebrew

Installation on Windows

MsysGit

- Incl. Git Bash/GUI & Shell Integration
  - install Git bash + command prompt
  - select checkout windows / commit unix

Your turn!


- Ensure you have git installed
Reproducible Research

Git GUI (default) Gitk
Reproducible Research

Git GUI (Mac OS) GitX-dev

http://rowanj.github.io/gitx/

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Git GUI (Windows/Mac) SourceTree

http://www.sourcetreeapp.com/

1. Let it install a default git ignore file
2. make it load your SSH key created with Putty
Preliminary Configurations

- Global Git configuration are stored in `~/.gitconfig`
  - Ex: see my personal `.gitconfig`
- You **SHOULD** at least configure your name and email to commit
  - open a terminal (Git bash under windows) for the below commands

```
$> git config --global user.name "Firstname LastName"
$> git config --global user.email "Firstname.Lastname@uni.lu"
$> git config --global color.ui true  # Colors
$> git config --global core.editor vim  # Editor
```
Global Git configuration are stored in ~/.gitconfig

- Ex: see my personal .gitconfig

You SHOULD at least configure your name and email to commit
- open a terminal (Git bash under windows) for the below commands

```bash
$> git config --global user.name "Firstname LastName"
$> git config --global user.email "Firstname.Lastname@uni.lu"
$> git config --global color.ui true
$> git config --global core.editor vim
```

# Colors

# Editor

Your Turn!

- Then check the changes by: git config -l | grep user
You can also create git command aliases in `~/.gitconfig`.

```bash
[alias]
up = pull origin
pu = push origin
st = status
df = diff
ci = commit -s
co = checkout
br = branch
w = whatchanged --abbrev-commit
ls = ls-files
gr = log --graph --oneline --decorate
amend = commit --amend
```
Reproducible Research

Git Workflow

- **Local**
  - working directory
  - staging area
  - git directory (repository)

- **Remote**
  - remote repo

- **Commands**
  - `git add`
  - `git commit`
  - `git push`
  - `git fetch` / `git pull`
  - `git merge`
  - `git checkout`
$> \text{git [flow] init}

- Initializes a new git (flow) repository in the current directory
Creating a Repository

$> git [flow] init

- Initializes a new git (flow) repository in the current directory

Your Turn!

$> cd /tmp
$> mkdir firstproject
$> cd firstproject

$> git init
Initialized empty Git repository in /private/tmp/firstproject/.git/
Cloning a Repository

```bash
$> git clone [--recursive] <url> [<path>]
```

<table>
<thead>
<tr>
<th>Type</th>
<th>URL Format / Example</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>/path/to/project.git</td>
<td>n/a</td>
</tr>
<tr>
<td>SSH</td>
<td>git+ssh://user@server:port/project.git</td>
<td>22</td>
</tr>
<tr>
<td>Git</td>
<td>git://server/project.git</td>
<td>9418</td>
</tr>
<tr>
<td>HTTPS</td>
<td><a href="https://github.com/Falkor/falkorlib.git">https://github.com/Falkor/falkorlib.git</a></td>
<td>443</td>
</tr>
</tbody>
</table>
Reproducible Research

Cloning a Repository

$> $ git clone [--recursive] <url> [<path>]

Your Turn!

$> cd /tmp
$> git clone https://github.com/Falkor/RR-tutorials.git
Cloning into 'tutorials'...
remote: Counting objects: 1247, done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 1247 (delta 32), reused 0 (delta 0), pack-reused 1181
Receiving objects: 100% (1247/1247), 15.74 MiB | 3.08 MiB/s, done.
Resolving deltas: 100% (588/588), done.
Checking connectivity... done.
$> git clone --recursive \n    https://github.com/Falkor/RR-tutorials.git /tmp/tutorials2
$>$ git status [-s]  # -s: short / simplified output
Inspecting a Repository

$> git status [-s]  
# -s: short / simplified output

Your Turn!

$> cd /tmp/firstproject
$> git status
On branch master
Initial commit
nothing to commit

$> git status
On branch master
Initial commit
Untracked files:
  README
nothing added to commit but untracked files present
$> git status -s
?? README

# Create an empty file
$> touch README.md
Add / Tracking [new] file(s)

$> \text{git add \([-f\]} \ <\text{pattern}>$

- Adds changes to the index
  - Add a specific file: \text{git add README}
  - Add a set of files: \text{git add *.py}

- Beware that empty directory cannot be added \textbf{directly}
  - due to the internal file representation (\textbf{blobs})
  - \textbf{Tips}: add an hidden file .empty (or .gitignore)
### Add / Tracking [new] file(s)

- **Add changes to the index**
  - Add a specific file: `git add README`
  - Add a set of files: `git add *.py`

- **Beware that empty directory cannot be added directly**
  - Due to the internal file representation (*blobs*)
  - **Tips:** add an hidden file `.empty` (or `.gitignore`)

---

#### Your Turn!

- `$ > cd /tmp/firstproject$
- `$ > git status -s$
- `?? README`

- `$ > git add README$
- `$ > git status -s$
- `A README`
Committing your changes

$> \text{git commit [-s] [-m "msg"]}$

- Commit all changes: `git commit -a`
$> git commit [-s] [-m "msg"]

Commit all changes: git commit -a

Your Turn!

$> cd /tmp/firstproject
$> git commit -s -m "add README"  # OR git ci -m "add README"
[master (root-commit) ee60f53] add README
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 README
$> git status  # OR git st
On branch master
nothing to commit, working directory clean
Removing Files

$> $ git rm [-rf] [--cached] <file>

- `--cached`: remove from Staging area
- otherwise (default): from index and file system
Ignoring Files

you can create a `.gitignore` file listing patterns to ignore

- Blank lines or lines starting with `#` are ignored
- End pattern with slash (`/`) to specify a directory
- Negate pattern with exclamation point (`!`)

Collection of useful `.gitignore` templates

- `.DS_Store`
- `*~`
- `*.asv`
- `*.m~`
- `*.mex*`
- `tmp/*`
- `\LaTeX\ `.gitignore`
- Python `.gitignore`
- Ruby `.gitignore`
Reproducible Research

Moving Files

$> \text{git mv} \text{ <source>} \text{ <destination>}

# Equivalent of:
mv \text{ <source>} \text{ <destination>}
git rm \text{ <source>}
git add \text{ <destination>}

Sebastien Varrette (University of Luxembourg)
Reproducible Research

Moving Files

$>\text{git mv } \text{<source>} \text{ <destination>}$

# Equivalent of:
\text{mv} \text{ <source>} \text{ <destination>}
\text{git rm} \text{ <source>}
\text{git add} \text{ <destination>}

Your Turn!

$>\text{cd }/\text{tmp/firstproject}$
$>\text{git mv } \text{README } \text{README.md}$
$>\text{git status}$
On branch master
Changes to be committed:
renamed: \text{README} \rightarrow \text{README.md}$
$>\text{git commit -m } "\text{a first move}"$
Reproducible Research

Check the Commit History

```bash
$> git log [-p] [--stat] [--graph --oneline --decorate]
```

- `-p / --stat`: show the differences introduced in each commit
- You can also perform some date filtering
  ```bash
  $> git log --since=2.weeks
  ```
- Ncurses-based text-mode interface: `tig`
Reproducible Research

Check the Commit History

```bash
$> git log [-p] [--stat] [--graph --oneline --decorate]
```

- `-p / --stat`: show the differences introduced in each commit
- You can also perform some date filtering
  ```bash
  $> git log --since=2.weeks
  $> git log --since=2.weeks
  ```
- Ncurses-based text-mode interface: `tig`

Your Turn!

```bash
$> cd /tmp/firstproject
$> git log --oneline --graph --decorate # OR git gr
* f1f0c27 (HEAD -> master) a first move
* ee60f53 add README
$> git log -p -1 # only the last commit OR git show
$> tig
```
Show differences

$ git diff [--cached] [<ref>]

- Check un-staged changes: `git diff`
  - `--cached`: check staged changes
- Relative to a specific revision:
  - $ git diff 1776f5
  - $ git diff HEAD~
Undoing Things

$> \texttt{git\ commit\ --amend}\quad\#\ Change\ the\ last\ commit
Undoing Things

$> git commit --amend  
   # Change the last commit

$> git unstage <file>  
   # or git reset HEAD <file>
Undoing Things

$> \text{git commit \texttt{-amend}} \quad \# \text{Change the last commit}$

$> \text{git unstage <file>} \quad \# \text{or git reset HEAD <file>}$

$> \text{git checkout -- <file>} \quad \# \text{DANGER! Un-modify modified file}$

- Restore to the last committed/cloned version: \textbf{all changes are lost!}
Undoing Things

$> \text{git commit --amend} \quad \# \text{Change the last commit}

$> \text{git unstage <file>} \quad \# \text{or git reset HEAD <file>}

$> \text{git checkout -- <file>} \quad \# \text{DANGER! Un-modify modified file}

$> \text{git revert <commit>} \quad \# \text{revert a <commit>}

- Make a new commit that undoes all changes made in <commit>
Undoing Things

$> git commit --amend  # Change the last commit

$> git unstage <file>  # or git reset HEAD <file>

$> git checkout -- <file>  # DANGER! Un-modify modified file

$> git revert <commit>  # revert a <commit>

Your Turn!

$> cd /tmp/firstproject
$> git commit --amend
$> echo 'toto' >> README.md

$> cat README.md && git status
$> git checkout -- README
$> git status
Git Summary

Basic Workflow

Edit files
Stage the changes
Review your changes
Commit the changes

vim / emacs / subl ...
git add
git status
git commit
Git Summary

For cheaters: A Basicerer Workflow

Edit files
vim / emacs / subl...
Stage & commit the changes

git commit -a
Git Summary

For cheaters: A Basicerer Workflow

Edit files
vim / emacs / subl ...
Stage & commit the changes
git commit -a

Advises: Commit early, commit often!

- commits = save points
  - use descriptive commit messages
- Don't get out of sync with your collaborators
- Commit the sources, not the derived files

Not covered here (by lack of time)

- Branches, tags, remotes, submodules, subtrees, etc...
Summary

1. Introduction and Motivating Examples

2. Reproducible Research
   - Easy-to {read|take|share} Docs
   - Sharing Code and Data
   - Mastering your [reproducible] environment

3. Conclusion
Environment Management

RR assumes that you **Master your environment**

Keep it **clean and automated** by:

- Using common building tools: `make`, `cmake` etc.
- Using a constrained environment:
  - Sandboxed Ruby environment: `bundler`, `Gemfile`
  - Sandboxed Python: `pip freeze`, `pyenv`, `virtualenv`
  - VMs or Containers: `Vagrant`, `Docker`
- Automate its building through cross-platform recipes
- Automatically test your recipes for Environment configuration
Consider using RVM, rbenv and more importantly Bundler
  ↩ Bring the flexibility of Rakefile (Makefile + Ruby)
  ↩ Bundler: **reproducible** running environment across developpers
  ↩ easy configuration through Gemfile [.lock] + bundle command

- RVM: sandboxed environment per project (**alternative**: rbenv)
  ↩ easy configuration through .ruby-{version,gemset} files
Consider using **RVM**, **rbenv** and more importantly **Bundler**

- Bring the flexibility of Rakefile (Makefile + Ruby)
- **Bundler**: **reproducible** running environment **across** developpers
- easy configuration through Gemfile[.lock] + bundle command

- **RVM**: sandboxed environment per project (**alternative**: rbenv)
  - easy configuration through .ruby-{version,gemset} files

**Typical setup of a freshly cloned project:**

```bash
$> gem install bundler  # assuming it is not yet available
$> bundle            # clone ruby deps/env as defined in Gemfile*
$> rake -T           # To list the available tasks
```
Consider using RVM, rbenv and more importantly Bundler

- Bring the flexibility of Rakefile (Makefile + Ruby)
- Bundler: **reproducible** running environment **across** developers
- easy configuration through Gemfile[.lock] + bundle command

RVM: sandboxed environment per project (**alternative**: rbenv)

- easy configuration through .ruby-\{version,gemset\} files

**Typical setup of a freshly cloned project:**

```
$> gem install bundler  # assuming it is not yet available
$> bundle              # clone ruby deps/env as defined in Gemfile*
$> rake -T             # To list the available tasks
```

**Recommended Gems**

rake, bundler, falkorlib
Reproducible Research

Controlled Python Environment

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```bash
$> pip install <package>  # install <package>
```
Controlled Python Environment

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```
$> pip install <package>  # install <package>
```

```
$> pip install -U pip  # upgrade on Linux/Mac OS
```
Controlled Python Environment

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```bash
$> pip install <package>  # install <package>
```

```bash
$> pip install -U pip  # upgrade on Linux/Mac OS
```

- Dump python environment to a requirements file

```bash
$> pip freeze -l > requirements.txt  # as Ruby Gemfiles
```
Reproducible Research

Pyenv / VirtualEnv / Autoenv

- **pyenv**: ≃ RVM/rbenv for Python
- **virtualenv**: ≃ RVM Gemset
- **(optional) autoenv**
  - Directory-based shell environments
  - Easy config through `.env` file. **Ex:**

```bash
# (rootdir)/.env : autoenv configuration file
pyversion='head .python-version'
pvenv='head .python-virtualenv'

pyenv virtualenv --force --quiet ${pyversion} ${pvenv}-${pyversion}
# activate it
pyenv activate ${pvenv}-${pyversion}
```
Let’s see how to reproduce a simple yet practical example in a constrained and reproducible VM environment.

**Challenge 1: Reproduce the Build of these Slides**

Several tricky issues illustrating previous best practices

- grab the sources `git`
- use of a constrained environment `Vagrant`
- installing the prerequisite software environment `apt-get`

✓ [un]common mix here: `make`, `latex-beamer`, `biber`, `pandoc`...
✓ generally the **major challenge** in reproducing computations...
Let’s see how to reproduce a simple yet practical example in a constrained and reproducible VM environment.

**Challenge 1: Reproduce the Build of these Slides**

Several tricky issues illustrating previous best practices:

- grab the sources
  - git
- use of a constrained environment
  - Vagrant
- installing the prerequisite software environment
  - apt-get

- [un]common mix here: make, latex-beamer, biber, pandoc...
- generally the major challenge in reproducing computations...


**IF NOT YET DONE:**
Reproducible Research

Grab the [Code/Data] Source

- You should have now Git installed
- Get the RR-tutorials repository from Github

```bash
$> git clone https://github.com/Falkor/RR-tutorials.git
$> cd RR-tutorials
$> make setup  # OR git submodule init && git submodule update
```

- **Notable** elements within this cloned repository:
  - the \LaTeX{} slides sources \texttt{slides/2016/cloudcom2016/src/}
  - Documentation sources \texttt{mkdocs.yml} and \texttt{docs/}
  - Vagrant configuration for this project \texttt{Vagrantfile}
  - Bats unit tests \texttt{tests/}
  - Continuous Integration settings through Travis-CI \texttt{.travis.yml}
Use a Constrained Environment

Development environments made easy.

Create and configure lightweight, reproducible, and portable development environments.

http://vagrantup.com/
What is Vagrant?

Create and configure lightweight, reproducible, and portable development environments

- **Command line** tool
- Easy and Automatic per-project VM management
  - Supports many hypervisors: VirtualBox, VMWare...
  - Easy text-based configuration (Ruby syntax) Vagrantfile
- Supports **provisioning** through configuration management tools
  - Shell
  - Puppet
  - Salt...

Cross-platform: runs on Linux, Windows, MacOS
Mac OS X:

→ best done using Homebrew and Cask

```
$ > brew install caskroom/cask/brew-cask
$ > brew cask install virtualbox  # install virtualbox
$ > brew cask install vagrant
$ > brew cask install vagrant-manager # cf http://vagrantmanager.com/
```

Windows / Linux:

→ install Oracle Virtualbox and the Extension Pack

→ install Vagrant
Why use Vagrant?

- Create new VMs quickly and easily: only one command!
  - `vagrant up`

- Keep the number of VMs under control
  - All configuration in VagrantFile

- **Reproducibility**
  - Identical environment in development and production

- **Portability**
  - Avoid sharing 4 GB VM disks images
  - Vagrant Cloud to share your images

- **Collaboration made easy:**
  
  ```
  $> git clone ...
  $> vagrant up
  ```
Reproducible Research

Minimal default setup

```bash
$> vagrant init [-m] <user>/<name> # setup vagrant cloud image
```

- A Vagrantfile is configured for box `<user>/<name>`
  - Find existing box: Vagrant Cloud
  - You can have multiple (named) box within the same Vagrantfile
    - See ULHPC/puppet-sysadmins/Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = '<user>/<name>'
  config.ssh.insert_key = false
end
```

<table>
<thead>
<tr>
<th>Box name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu/trusty64</td>
<td>Ubuntu Server 14.04 LTS</td>
</tr>
<tr>
<td>debian/contrib-jessie64</td>
<td>Vanilla Debian 8 “Jessie”</td>
</tr>
<tr>
<td>centos/7</td>
<td>CentOS Linux 7 x86_64</td>
</tr>
<tr>
<td>svarrette/RR-tutorials</td>
<td>IEEE CloudCom 2016 Tuto</td>
</tr>
</tbody>
</table>
Pulling and Running a Vagrant Box

```bash
vagrant up  # boot the box(es) set in the Vagrantfile
```

- Base box is downloaded and stored locally `~/.vagrant.d/boxes/`
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: `/vagrant` = directory hosting Vagrantfile
$>\text{vagrant up} \quad \# \text{ boot the box(es) set in the Vagrantfile}

- Base box is downloaded and stored locally $\sim/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: /vagrant = directory hosting Vagrantfile

$>\text{vagrant status} \quad \# \text{ State of the vagrant box(es)}
Pulling and Running a Vagrant Box

```bash
$> vagrant up  # boot the box(es) set in the Vagrantfile
```

- Base box is downloaded and stored locally ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: /vagrant = directory hosting Vagrantfile

```bash
$> vagrant status  # State of the vagrant box(es)
```

```bash
$> vagrant ssh        # connect inside it, CTRL-D to exit
```
Stopping Vagrant Box

$>\text{vagrant \{ destroy | halt \}} \quad \# \text{ destroy / halt}

- Once you have finished your work within a \textit{running} box:
  - save the state for later with \texttt{vagrant halt}
  - reset changes / tests / errors with \texttt{vagrant destroy}
  - commit changes by generating a new version of the box
Your Turn!  


- **Steps [1-4]** to cover the following elements:
  - Basic Usage of Vagrant
  - Build these Slides

  ✓ find the prerequisite software environment  
  ✓ [un]common mix here: make, latex-beamer, biber, pandoc...

**Hints:**

- if a package is missing, find the appropriate one apt-cache search
- Ubuntu Package Search for a missing *.sty  
  ✓ Search the contents of packages for Distribution Trusty
Now you have *hopefully* a working **documented procedure**

→ it’s time to **bundle it** for provisioning the box upon boot

→ key for sustainable reproducible environment

Simple case: **inline** provisioning *i.e.* list commands to run
Now you have *hopefully* a working **documented procedure**

→ it’s time to **bundle it** for provisioning the box upon boot

→ key for sustainable reproducible environment

Simple case: **inline** provisioning *i.e.* list commands to run

```bash
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update --fix-missing
  sudo apt-get upgrade
  # Complete the below list of missing packages
  apt-get -yq --no-install-suggests --no-install-recommends install \
    git make latex-beamer biber latex-make [...] 
SHELL
```
Now you have *hopefully* a working **documented procedure**

→ it’s time to **bundle it** for provisioning the box upon boot

→ key for sustainable reproducible environment

**Simple case:** **inline** provisioning *i.e.* list commands to run

```shell
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update --fix-missing
  sudo apt-get upgrade
  # Complete the below list of missing packages
  apt-get -yq --no-install-suggests --no-install-recommends install \ 
    git make latex-beamer biber latex-make [...] 
SHELL

$> vagrant provision           # test your provisioning config
```
**Your Turn!**

Steps 5:

- adapt the Vagrantfile to embed your commands
- recall that relative paths are expanded relative to the location of the root Vagrantfile
- inline command are run as the `vagrant` user, **not** `root`

**IMPORTANT:**

- all your commands should run in a **non-interactive** way

```bash
apt-get install -y <package>  # Debian / Ubuntu
yum   install -y <package>    # CentOS/ Redhat
```
Reproducible Research

Vagrant Box Shell Provisioning

• Embed your inline commands in a **Shell/Python/Ruby** script
  
  → see sample script `vagrant/bootstrap.sample.sh`

```bash
config.vm.provision "shell", path: "<script>{sh|py|rb}""n
```

Your Turn!


• **Steps 6:** copy and adapt `vagrant/bootstrap.sample.sh`
  
  → adapt the Vagrantfile to provision the VM with your script
  
  → test a reproducible provisioning from scratch

```bash
$> vagrant destroy && vagrant up && vagrant ssh
$> make -C make -C /vagrant/slides/2016/cloudcom2016/src/
```

Sebastien Varrette  (University of Luxembourg)
At some moment, you probably want to diffuse your custom box!

- **Ex:** `svarrette/RR-tutorials` used for this tutorial
- use **Vagrant Cloud** as a global storage media
- `VBoxManage list runningvms` to get the real box name

```bash
$> vagrant package --base <real-box-name> --output <name>.box
```
At some moment, you probably want to diffuse your custom box!

- Ex: svarrette/RR-tutorials used for this tutorial
- use Vagrant Cloud as a global storage media
- VBoxManage list runningvms to get the real box name

```bash
$> vagrant package --base <real-box-name> --output <name>.box
```

**BEFORE packaging your box:**

- Use official insecure SSH key: `config.ssh.insert_key=false`
- Purge the VM to reduce its size: see `vagrant/purge.sh`
  - remove useless [big] packages: `aptitude purge [...]`
  - Empty logs/history etc.
  - Zero out the free space: `dd if=/dev/zero of=/EMPTY bs=1M`
- Up-to-date Virtualbox Guest additions: `vagrant vbguest`
Reproducible Research

Detailed Pre-Packaging Steps (1/2)

- Ensure you **DO NOT** reset the default (insecure) SSH key
  - default **expected** setting to SSH your box
  - before `vagrant up`, ensure replacement of SSH keys is **not done**

```
config.ssh.insert_key = false  # in Vagrantfile
```

- **Purge** the VM, in particular to **Zero out the free space**
  - see `vagrant/purge.sh`

```
# Remove APT cache
apt-get clean -y && apt-get autoremove -y && apt-get autoremove -y

# Remove bash history
unset HISTFILE
rm -f /root/.bash_history & rm -f /home/vagrant/.bash_history

# Zero out free space to aid VM compression
dd if=/dev/zero of=/EMPTY bs=1M
rm -f /EMPTY
```
Detailed Pre-Packaging Steps (2/2)

Ensure an **Up-to-date Virtualbox Guest additions**

- ensure optimized usage of the box
- simplified management with the `vbguest` plugin

---

# Install the 'vbguest' plugin

```bash
$> vagrant plugin install vagrant-vbguest
$> vagrant vbguest --status
GuestAdditions versions on your host (5.1.8) and guest (4.3.36) do not match.
```

# Upgrade the GuestAdditions

```bash
$> vagrant vbguest --do install --auto-reboot [--force]
```

---

If you want the **manual** way:

- copy `/Applications/VirtualBox.app/Contents/MacOS/VBoxGuestAdditions.iso`
- mount in **within** the VM
- execute `VBoxLinuxAdditions.run`
Reproducible Research

Vagrant Box Packaging

# Locate the internal name of the running VM and repackage it

```bash
$ VBoxManage list runningvms
"RR-tutorials_default_1481463725786_57301" {...}

$ vagrant package \
   --base vagrant-vms_default_1431034026308_70455 \
   --output <os>-<version>-<arch>.box
```
# Locate the internal name of the running VM and repackage it

```
$ VBoxManage list runningvms
"RR-tutorials_default_1481463725786_57301" {...}
```

```
$ vagrant package \
   --base vagrant-vms_default_1431034026308_70455 \
   --output <os>--<version>--<arch>.box
```

- Now you can upload the generated box on **Vagrant Cloud**.
  - select 'New version', enter the new version number
  - add a new box provider (**Virtualbox**)
  - upload the generated box
# Locate the internal name of the running VM and repackage it

```
$ VBoxManage list runningvms
"RR-tutorials_default_1481463725786_57301" {...}
```

```
$ vagrant package \
   --base vagrant-vms_default_1431034026308_70455 \
   --output <os>-<version>-<arch>.box
```

- Now you can upload the generated box on Vagrant Cloud.
  - select ‘New version’, enter the new version number
  - add a new box provider (Virtualbox)
  - upload the generated box

- Upon successful upload: **release** the uploaded box
  - by default it is unreleased
  - Now people using the `<user>/<name>` box will be notified of a pending update
Vagrant Box Packaging

Your Turn!  

**Steps 7-8**: Package your box and diffuse it on Vagrant Cloud

- Make preliminary checks
- Purge the VM
- Package it and Upload to Vagrant Cloud
You might rely on **Falkor/vagrant-vms**

- use it at your own risks
- based on `packer` and `veewee`

```bash
$> git clone https://github.com/Falkor/vagrant-vms.git
$> cd vagrant-vms
$> gem install bundler && bundle install
$> rake setup
```
You might rely on Falkor/vagrant-vms

→ use it at your own risks
→ based on packer and veewee

```
$> git clone https://github.com/Falkor/vagrant-vms.git
$> cd vagrant-vms
$> gem install bundler && bundle install
$> rake setup
```

# initiate a template for a given Operating System:
```
$> rake packer:{Debian,CentOS,openSUSE,scientificlinux,ubuntu}:init
# Build a Vagrant box
$> rake packer:{Debian,CentOS,openSUSE,scientificlinux,ubuntu}:build
# If things goes fine:
$> vagrant box add packer/<os>-<version>-<arch>/<os>-<version>-<arch>.box
```
Shell provisioning is a reasonable good basis but **not sufficient**

→ hard to be cross-platform

You quickly something more **consistent**

→ Puppet

→ Salt...
Shell provisioning is a reasonable good basis but **not sufficient**

- hard to be cross-platform

You quickly something more **consistent**

- Puppet
- Salt...

**Puppet: Reproducible/Cross-Platform IT Environment**

- Advanced configuration management and **IT Automation**
  - cross-platform w. Puppet’s Resource Abstraction Layer (**RAL**)  
  - Git-based workflow

- Embed environment management in **manifests** and **modules**
  - **nodes manifests**: nodes definitions
  - **modules**: (reusable) set of recipe to configure a given service

  ✓ Large Community Recipes / Modules  
  https://forge.puppet.com/
Puppet Operational modes

- **Masterless** - apply Puppet manifests directly on the target system.
  - No need of a complete client-server infrastructure.
  - Have to distribute manifests and modules to the managed nodes.

$>$ puppet apply --modulepath /modules/ /manifests/file.pp
Puppet Operational modes

**Masterless** - apply Puppet manifests directly on the target system.
- No need of a complete client-server infrastructure.
- Have to distribute manifests and modules to the managed nodes.

```bash
$> puppet apply --modulepath /modules/ /manifests/file.pp
```

**Master / Client** Setup
- server (running as puppet) listening on 8140 on the Puppet Master
- client (running as root) on each managed node.
  - Run as a service (default), via cron (with random delays), manually or via MCollective
- Client and Server have to share SSL certificates
  - certificates must be signed by the Master CA

```bash
$> puppet agent --test [--noop] [--environment <environment>]
```
Puppet DSL

- A Declarative Domain Specific Language (DSL)
  - defines **STATES** (and not procedures)
- Puppet code is written in **manifests** `<file>.pp`
  - declare resources that affect elements of the system
    - each resource has a type (package, service, file, user, exec ...)
    - each resource has a uniq title
  - resources are grouped in **classes**
- Classes and configuration files are organized in **modules**
- **Example** of resources types:

```plaintext
file { '/etc/motd':
  content => "Toto"
}

package { 'openssh':
  ensure => present,
}

service { 'httpd':
  ensure => running,
  enable => true,
}
```
Containers of different resources

→ Can have parameters since Puppet 2.6

```ruby
class mysql (
    $root_password = 'default_value',
    $port = '3306',
) {
    package { 'mysql-server':
        ensure => present,
    }
    service { 'mysql':
        ensure => running,
    }
    [...] 
}
```
To use a class previously defined, we **declare** it.

“Old style” class declaration, without parameters:

```erb
include mysql
```

“New style” (from Puppet 2.6) with explicit parameters:

```erb
class { 'mysql':
  root_password => 'my_value',
  port => '3307',
}
```

A class is **uniq** to a given node.
Similar to parametrized classes ...  
\[\text{\ldots but can be used multiple times (with different titles).}\]

```ruby
# Definition of a define
define apache::virtualhost (
    $ensure = present,
    $template = 'apache/virtualhost.conf.erb',
    [...] ) {
    file { "ApacheVirtualHost_${name}":
        ensure => $ensure,
        content => template("${template}")},

    }
}
# Declaration of a define:
apache::virtualhost { 'www.uni.lu':
    template => 'site/apache/www.uni.lu-erb'
}
```
Puppet Variables and Facts

- Can be defined in different places and by different actors:
  - by client nodes as facts
  - defined by users in Puppet code, on Hiera on in the ENC
  - built-in and be provided directly by Puppet
- Facts using facter:
  - runs on clients and collects facts that the server can use as variables

```bash
$> facter
architecture => x86_64
fqdn => toto.uni.lu
kernel => Linux
memorytotal => 16.00 GB
operatingsystem => Centos
operatingsystemrelease => 6.3
osfamily => RedHat
virtual => physical
[...]
```

- Can be used outside Puppet
- Good tool to **abstract** your environment
  - permits **reproducible** and cross-platform developments
Puppet User Variables

- **In Puppet manifests:**

  \[
  \begin{align*}
  \$role &= \text{'mail'} \\
  \$package &= \$::operatingsystem \ ? \ { \\
  \quad &\quad /(?i:Ubuntu|Debian|Mint)/ \Rightarrow \text{'apache2'}, \\
  \quad &\quad \text{default} \Rightarrow \text{'httpd'}, \\
  \quad &\quad }
  \end{align*}
  \]

- **In an External Node Classifier (ENC)**

  \[\Rightarrow \text{Common ENC: Puppet DashBoard, the Foreman, Puppet Enterprise.} \]

- **In an Hiera backend**

  \[
  \$syslog_server = \text{hiera(syslog_server)}
  \]
A node/system is identified by its certname
  - defaults to the node’s fqdn

```puppet
node 'web01' {
    include apache
}
```

```puppet
node /^www\d+$/ {
    include apache
}
```

Nodes classification can be done by External Node Classifier (ENC)
  - Puppet DashBoard, The Foreman and Puppet Enterprise

Nodes classification can be done also by Hiera
  - In `/etc/puppet/manifests/site.pp`

```
hiera_include('classes')
```
Operate in **masterless** mode

Embed your manifests and modules in your repository

→ grab community modules with `librarian-puppet, r10K`

```ruby
config.vm.provision :puppet do |puppet|
  puppet.hiera_config_path = 'hieradata/hiera.yaml'
  puppet.working_directory = '/vagrant'
  puppet.manifests_path = "manifests"
  puppet.module_path = "modules"
  puppet.manifest_file = "init.pp"
  puppet.options = ['-v', '--report', '--show_diff', '--pluginsync']
end
```
Vagrant Puppet Provisionning

- Operate in **masterless** mode
- Embed your manifests and modules in your repository

  → grab community modules with **librarian-puppet**, **r10K**

```ruby
config.vm.provision :puppet do |puppet|
  puppet.hiera_config_path = 'hieradata/hiera.yaml'
  puppet.working_directory = '/vagrant'
  puppet.manifests_path = "manifests"
  puppet.module_path = "modules"
  puppet.manifest_file = "init.pp"
  puppet.options = [ '-v', '--report', '--show_diff', '--pluginsync' ]
end
```

Your Turn!
Operate in **masterless** mode

- Embed your manifests and modules in your repository
  
  → grab community modules with `librarian-puppet, r10K`

```ruby
config.vm.provision :puppet do |puppet|
  puppet.hiera_config_path = 'hieradata/hiera.yaml'
  puppet.working_directory = '/vagrant'
  puppet.manifests_path = 'manifests'
  puppet.module_path = 'modules'
  puppet.manifest_file = 'init.pp'
  puppet.options = ['-v', '--report', '--show_diff', '--pluginsync']
end
```

**Your Turn! ... Or not 😊(no time)**
Software/Modules Management

Software Management Challenge

- Not so much standardization
  - every machine/app has a different software stack / installation procedure
  - Sites share unique hardware among teams with very different requirements
  - You want to experiment with many exotic architectures

Software Flavor vs. Dependency nightmare vs Performance

- **Ex**: 3 compilers + 3 MPI + n software
- Complex set of CLI options,
- One of the main limits for RR

Some Tools can help you!

- Easybuild  
  [http://easybuild.readthedocs.io/](http://easybuild.readthedocs.io/)
- Spack  
  [http://spack.readthedocs.io/](http://spack.readthedocs.io/)
- CDE
- Kameleon  
EasyBuild: open-source framework to (automatically) build scientific software

**Why?** "Could you please install this software on the cluster?"

Scientific software are often **painful** to build
- non-standard build tools / incomplete build procedure
- hardcoded parameters and/or poor/outdated documentation

EasyBuild helps to facilitate this task
- consistent software build and installation framework
- automatically generates LMod modulefiles
# pick an installation prefix to install EasyBuild to
export EASYBUILD_PREFIX=$HOME/.local/easybuild
# download script
curl -O goo.gl/RK3Gpf  # Get bootstrap_eb.py
# bootstrap EasyBuild
python bootstrap_eb.py $EASYBUILD_PREFIX
# update $MODULEPATH, and load the EasyBuild module
module use $EASYBUILD_PREFIX/modules/all
module load EasyBuild
# Load EasyBuild module
module load EasyBuild
# Check version
eb --version
# Look for HPL
eb -S HPL
# Check what needs to be built to compile HPL 2.1 with Intel compiler
HPL-2.1-intel-2016b.eb
# Check what needs to be built to compile HPL 2.1 with GCC/OpenMPI/...
eb HPL-2.1-foss-2016b.eb -Dr
# Build HPL and its dependencies
eb HPL-2.1-foss-2016b.eb -r
# See available HPL now
module avail HPL
# Amending an existing easyconfig
eb HPL-2.1-foss-2016b.eb --try-software-version=2.2
Kameleon: Reproducible SW

- Uses recipes (high-level description)
  - Similar to cfengine, Puppet, Chef in the sysadmin world
- Persistent cache to allow re-generation without external resources
  - Linux distribution mirror → self-contained archive
  - Supports LXC, Docker, VirtualBox, qemu, Kadeploy images, etc.

---

Open-source engine
Automates the deployment of any application
  ↦ lightweight, portable, self-sufficient container
  ↦ will run virtually anywhere
Tries to achieve deterministic builds by isolating your service
  ↦ build done from a snapshotted OS and running imperative steps on top of it

Dependency hell:
  ↦ Docker works with images that consume minimal disk space
  ↦ all images are versioned, archivable, and shareable

Dockerfiles: resolving imprecise documentation
Virtual machines

- app + binaries + libraries
- incl. an entire guest OS

Container

- app + binaries + libraries
- kernel shared
- run on any computer
Pulling and Running Images

```bash
$ docker pull <name>:<tag>
```

- Pull a **public** image such as ubuntu or centos
  - If a tag is not specified, use “latest”.
$>\text{docker pull } \langle\text{name}\rangle:\langle\text{tag}\rangle$

- Pull a **public** image such as ubuntu or centos
  - if a tag is not specified, use “latest”.

$>\text{docker run -it } \langle\text{name}\rangle$
Pulling and Running Images

$> \text{docker pull} \text{ <name>:<tag>}$

- Pull a **public** image such as ubuntu or centos
  - if a tag is not specified, use “latest”.

$> \text{docker run} -it \text{ <name>}$

$> \text{docker commit} \text{ <ID> <name>}$
Pulling and Running Images

$> \texttt{docker pull } \textless \text{name}\textgreater :\textless \text{tag}\textgreater$

- Pull a \textbf{public} image such as \texttt{ubuntu} or \texttt{centos}
  - if a tag is not specified, use “latest”.

$> \texttt{docker run -it } \textless \text{name}\textgreater$

$> \texttt{docker commit } \textless \text{ID}\textgreater \textless \text{name}\textgreater$

\textbf{Your Turn!}

1 Introduction and Motivating Examples

2 Reproducible Research
   Easy-to {read|take|share} Docs
   Sharing Code and Data
   Mastering your [reproducible] environment

3 Conclusion
Conclusion

The Research Pipeline

Experiments
- Measured Data
- Nature/System/...
- Protocol (Design of Experiments)

Analysis
- Processing Code
- Analysis Code
- Presentation Code
- Figures
- Tables
- Numerical Summaries
- Published Article
- Text

Analysis/experiment feedback loop

Author

Reader

Scientific Question

Experiment Code (workload injector, VM recipes, ...)

Computational Results

Analytic Data

Sebastien Varrette (University of Luxembourg)

Reproducible Research at the Cloud Era
Accurate, organized and easy-to{read|take|share} Docs

→ Markdown, mkdocs, org-mode, Read the Docs...
Conclusion

**RR: Trying to Bridge the Gap**

- Accurate, organized and easy-to-read Docs
  - Markdown, mkdocs, org-mode, Read the Docs...

- Sharing Code and Data
  - git, Github, Bitbucket, Gitlab...
Conclusion

RR: Trying to Bridge the Gap

- Accurate, organized and easy-to-read Docs
  - Markdown, mkdocs, org-mode, Read the Docs...

- Sharing Code and Data
  - git, Github, Bitbucket, Gitlab...

- Mastering your environment clean and automated by:
  - Using common building tools: make, cmake etc.
  - Using a constrained environment
    - Sandboxed Ruby/Python, Vagrant, Docker
  - Automate its building through cross-platform recipes
  - Automatically test your recipes for Environment configuration
Sharing Code and Data

Is this enough?

1. Use a workflow that documents both data and process
2. Use the machine readable CSV format
3. Provide raw data and meta data, not just statistical outputs
4. Never do data manipulation and statistical tests by hand
5. Use R, Python or another free software to read and process raw data
   ✓ ideally to produce complete reports with code, results and prose
Conclusion

Reproducibility axes

- Always keep track of:
  - your **methodology**
  - your **code**
  - your (input) **data**

- Can you later come back and:
  - reproduce your experiment
  - including its environment
  - ... and obtain the same results?
Conclusion

Reproducibility axes

- Always keep track of:
  - your methodology
  - your code
  - your (input) data

- Can you later come back and:
  - reproduce your experiment
  - including its environment
  - ... and obtain the same results?

- If not, then now is the best time to start:
  - documenting your processes
  - describing your environment (software and hardware!)
  - versioning and tagging your code and data
  - (... and keep backups of it all)
Conclusion

Reproducibility levels

Is your research\textsuperscript{12}:

- reviewable
  \(\rightarrow\) desc. of your methods can be independently assessed?
- replicable
  \(\rightarrow\) are the tools available to duplicate the results?
- confirmable
  \(\rightarrow\) can the main conclusions be attained independently of your tools?
- auditable
  \(\rightarrow\) do you have records such that your research can be later defended?
  \(\rightarrow\) … or differences between independent confirmations resolved?

- open or reproducible, such that
  \(\rightarrow\) the procedures can be fully audited \textit{and}
  \(\rightarrow\) the results can be replicated or independently reproduced \textit{and}
  \(\rightarrow\) the results can be extended or the method applied to new problems

\textsuperscript{12}ICERM Report 2013: "Reproducibility in Computational and Experimental Mathematics"
Open challenges

Sometimes you need to:

- Continue your computation **elsewhere**
  → another HPC node/cluster, supercomputer, cloud instance

- Continue your computation **in a different environment**
  → another software stack (*just* OS, *some* libraries / compiler flags)

- Use a **different version** of a commercial or community software
Conclusion

Open challenges

Sometimes you need to:

- Continue your computation **elsewhere**
  \[\rightarrow \text{another HPC node/cluster, supercomputer, cloud instance}\]
- Continue your computation **in a different environment**
  \[\rightarrow \text{another software stack (just OS, some libraries / compiler flags)}\]
- Use a **different version** of a commercial or community software

Are your results consistent?
Conclusion

Open challenges

Sometimes you need to:

- Continue your computation **elsewhere**
  - another HPC node/cluster, supercomputer, cloud instance
- Continue your computation **in a different environment**
  - another software stack (*just* OS, *some* libraries / compiler flags)
- Use a **different version** of a commercial or community software

**Are your results consistent?**

Be wary of:

- Comparing algorithms running on diverse hw. infrastructures
- Restarting calculation with the same code but on diff. sw. env.
- ... different (usually newer...) version of the code
Conclusion

Open challenges

Sometimes you need to:

- Continue your computation **elsewhere**
  - another HPC node/cluster, supercomputer, cloud instance
- Continue your computation **in a different environment**
  - another software stack (*just* OS, *some* libraries / compiler flags)
- Use a **different version** of a commercial or community software

Are your results consistent?

Be wary of:

- Comparing algorithms running on diverse hw. infrastructures
- Restarting calculation with the same code but on diff. sw. env.
- ... different (usually newer...) version of the code

Keep track of your environment changes!
Thank you for your attention...

Questions?

Sebastien Varrette

mail:  Sebastien.Varrette@uni.lu
Office E-007
Campus Kirchberg
6, rue Coudenhove-Kalergi
L-1359 Luxembourg

1 Introduction and Motivating Examples
2 Reproducible Research
   Easy-to {read|take|share} Docs
3 Conclusion

Sharing Code and Data
Mastering your [reproducible] environment