

These are the slides of a [talk](#) given at the [1st MaRDI workshop on Scientific Computing](#)

1st MaRDI Workshop on Scientific Computing

Reproducibility infrastructure of the Julia language

Jürgen Fuhrmann
 WIAS Berlin

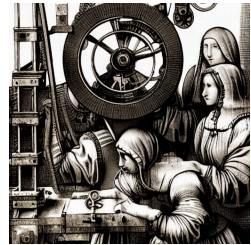


The Two Language Problem

... is at least a **three language problem**: you need to consider the **build system**, and at present we are talking about **CMake**...

- **Each project needs its own guru** to maintain the build system and to help to compile the code on new machines or to maintain docker containers
- Python APIs are easy to explain to general users, efficient algorithms are implemented in C/C++. **Python project codebases are intransparent for many of their users**
- **Containers are the new binaries**

We are in an **exponential boundary layer hitting a wall** regarding the complexity of build systems



dreamstudio.ai CC01.0

Julia

- Syntax comparable to matlab, python/numpy
- Just-ahead-of-time compilation to native code \Rightarrow high potential for performance without need to vectorize or to call a computational kernel in another language
- Performant multi-dimensional arrays
- Comprehensive linear algebra
- Parallelization: SIMD, multithreading, distributed
- Interoperability with C, C++, python, R...
- Use of modern knowledge in language design
- Open source (MIT License)

<https://julialang.org>

Open source

Julia is an open source project with [GitHub](#).

Packages

Packages extend Julia's core functionality. Each package is a git repository with standardized structure.

- **Package Registries** provide the infrastructure for finding package repositories via package names
- Default **General Registry**: \approx 8500 open source packages
- [Pkg.jl](#) - the built-in **Package Manager** is part of the Julia standard library
- **Composability** of packages due to generic Julia source code (like C++ header only libs)

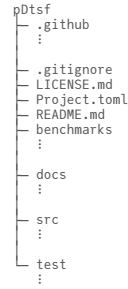
xkcd.com

```
INSTALL.SH
#!/bin/bash
pip install "$1" &
easy_install "$1" &
brew install "$1" &
npm install "$1" &
yum install "$1" &
dnf install "$1" &
docker run "$1" &
pkg install "$1" &
apt-get install "$1" &
sudo apt-get install "$1" &
steamoscmd +app.update "$1" validate &
git clone https://github.com/"$1"/"$1" &
cd "$1"/..;/configure;make;make install &
curl "$1" | bash &
```

Anatomy of a Julia package

Julia packages have a standardized structure

- Locally, each package is stored in a directory named e.g. MyPack for package MyPack.jl.
- Structure of a package directory:
 - MyPack/src: subdirectory for package source code
 - MyPack/src/MyPack.jl: code defining a module MyPack
 - Further Julia sources included by MyPack.jl
 - MyPack/test: code for unit testing
 - MyPack/docs: markdown sources + code for documentation
 - LICENSE: (open source) license
 - Project.toml: Metadata



Package metadata

Contents of Project.toml for the ForwardDiff.jl package

```

name = "ForwardDiff"
uuid = "f6369f11-7733-5829-9624-2563aa707210"
version = "0.10.32"

[deps]
CommonSubexpressions = "bbf7d656-a473-5ed7-a52c-81e309532950"
DiffResults = "163ba53b-c6d8-5494-b064-1a3d43ac40c5"
DiffRules = "b552c78f-8df3-52c6-915a-0e097449b14b"
LinearAlgebra = "37e2e46d-f89d-539d-b4ee-838fcccc9c8e"
LogExpFunctions = "2ab3a3ac-af41-5b50-aa03-7779005ae688"
NaNMath = "77ba4419-2d1f-58cd-9bb1-8f7ee604a2e3"
Preferences = "21216c6a-2e73-6563-6e65-726566657250"
Printf = "de0858da-6303-5e67-8744-51eddeeeb8d7"
Random = "9a3f8284-a2c9-5f02-9a11-845980a1fd5c"
SpecialFunctions = "276daf66-3868-5448-9aa4-cd146d93841b"
StaticArrays = "90137ffa-7385-5640-81b9-e52037218182"

[compat]
Calculus = "0.2, 0.3, 0.4, 0.5"
CommonSubexpressions = "0.3"
DiffResults = "0.0.1, 0.0.2, 0.0.3, 0.0.4, 1.0.1"
DiffRules = "1.4.0"
DiffTests = "0.0.1, 0.1"
LogExpFunctions = "0.3"
NaNMath = "0.2.2, 0.3, 1"
Preferences = "1"
SpecialFunctions = "0.8, 0.9, 0.10, 1.0, 2"
StaticArrays = "0.8.3, 0.9, 0.10, 0.11, 0.12, 1.0"
julia = "1"

[extras]
Calculus = "49dc2e85-a5d0-5ad3-a950-438e2897f1b9"
DiffTests = "de460e47-3fe3-5279-bb4a-814414816d5d"
InteractiveUtils = "b77e0a4c-d291-57a0-90e8-8db25a27a240"
SparseArrays = "2f01184e-e22b-5df5-ae63-d93eabab69eaf"
Test = "8dfed614-e22c-5e08-85e1-65c5234f0b40"

[targets]
test = ["Calculus", "DiffTests", "SparseArrays", "Test", "InteractiveUtils"]
  
```

Package metadata

Contents of Project.toml

- Package name
- UUID to identify package, name is secondary
 - ⇒ manage different packages with the same name
- Version according to Semantic Versioning
- [deps] section: list of package dependencies with UUIDs
- [compat] section: version compatibility bounds for dependencies and julia
- Further info: author, additional packages for testing ...



dreamstudio.ai CC01.0

Adding a package

```
julia> Pkg.add("MyPkg")
```

1. Package name and UUID are looked up in a **registry**
2. Package git repo URL read from registry (nowadays packages are cached and served from a package server by default)
3. Calculation of version compatibility for package and dependencies
4. Code of package and dependencies downloaded to ~/.julia/packages/
5. Package and dependencies recorded in current active **environment**

After adding a package, using MyPkg allows to use the package content in a Julia session or project source.



dreamstudio.ai CC01.0

Registries

A **registry** is a directory collecting metadata of packages for look-up

- Default: <https://github.com/JuliaRegistries/General>
 - Like blockchain: no deletions, continued forever
 - Packages must be open source
 - Automated heuristic decision process for new packages to be registered
- Local copy kept up-to-date for each Julia installation
- Multiple (e.g. institutional) registries are possible



dreamstudio.ai CC0.0

Environments

Environment: directory with `Project.toml` and `Manifest.toml`

- `Project.toml`: name + UUIDs of all packages added
- `Manifest.toml`: name + UUID + version + git-hash of package *and all of its dependencies and their dependencies*
- Each project can have its own environment
 - `$ julia`: activate default environment for julia version, e.g. `~/julia/environments/v1.7`
 - `$ julia --project=@xyz`: activate environment in `~/julia/environments/xyz`
 - `$ julia --project=dir` and `julia> Pkg.activate("dir")` activate environment in directory `dir`



dreamstudio.ai CC0.0

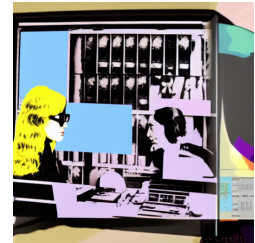
Further features & details

- Consistent package updates
- Package garbage collection
- Access to older revisions and git branches
- Standardized test environment for Julia packages. Julia core developers can test new julia versions with the registered packages to find out regressions
- **Artifacts:** `Artifacts.toml` records BLOBS available at given URL + content hash to be installed along with a project/package without the need to have them in git
- **Binary (jll) packages:** pre-built, cross compiled libraries for all relevant architectures managed as Artifacts allow to use libraries implemented in other languages in a simple an reliable way
 - See the Yggdrasil repository for the build scripts for all registered jlls

Reproducible projects

Transferring `Project.toml` and `Manifest.toml` allows to reproduce the exact package composition of a project

- **Alice**, working on Linux, creates a project using Julia and **a number of Julia packages**. She develops the code in a directory which is activated as a Julia environment. She sets up a git repository containing source code, documentation and both `Project.toml` and `Manifest.toml` files.
- **Bob**, working on windows with the same Julia version, checks out the code from the repo. A call to `Pkg.instantiate()` **installs all packages** in the exact combination of versions as Alice had them on her computer.



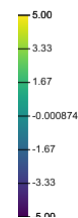
dreamstudio.ai CC0.0

Reproducible Notebooks: Pluto.jl

Browser based notebooks implemented in Julia and Javascript

- **Easy installation:** installed as a single Julia package on Linux, MacOS, Windows
- **Reactive:** cell results are automatically recalculated
- **Version controllable:** no computational results in the notebook
- **Reproducible:** notebooks contain their own environment
- Efficient interaction with HTML+Javascript
- Created by Fons van der Plas & his friends

PlutoVista.jl+vtk.js+webgl: 132651 nodes



```
f(x,y,z)=sin(x)*cos(y)*z;
```

More Pluto.jl Benefits

- **Clara** teaches a Julia based course in scientific computing. She prepares the **course material as Pluto notebooks**. After installation with simple instructions, **students run them** on their computers. The package environment automatically installs all packages necessary for a notebook. HTML and PDF previews available as well.
- **Students** prepare their **exam projects as Pluto notebooks**. Clara can receive their work and run it on her computer.
- MIT (Computational Thinking), TU Berlin (Scientific Computing) ...

Download Julia and install it according to the procedure on you particular operating system. Invoke Julia and issue the following commands:

```
using Pkg
Pkg.add("Pluto")
using Pluto
Pluto.run()
```

A menu will show up in the browser which allows to start the notebooks downloaded from the course homepage.

Further infrastructure

- [DrWatson.jl](#) manages code and computational results in a Julia project repository
 - Automatic generation of data file names from simulation parameters
- [Documenter.jl](#): documentation
 - from docstrings in package sources
- Visual Studio Code integration
- [Jupyter notebook support](#)
- Integration with [quarto](#) for reproducible publications



Some Issues

- Package loading and using latency due to JIT precompilation aka "Time to first plot"
 - Currently, the Julia community undertakes dedicated successful efforts towards fixing this problem
- Missing formal interface descriptions
 - Julia alternative to C++20 concepts ? Traits ?
 - Bottom up design process, fear to lose opportunities due to too rigid formalizations
- Resources for keeping infrastructure running
 - Many volunteers are involved at central points
 - Competitiveness depends on package contributions
 - Server infrastructure costs



Conclusions

Julia provides as well a **fresh approach to reproducibility**, learning from the experiences of `conda`, `npm` etc.

- Package management is part of the standard Julia workflow, available without further installation
- Transparent package and project source code without the need to know two languages or handling of build systems
- Introductions to Julia at an early stage should explain working with environments etc.
- Can we contribute to Julia and its infrastructure from the NFDI context ? (Same question for other open source ecosystems ...)

SIAM REVIEW
Vol. 59, No. 1, pp. 65–98

© 2017 Society for Industrial and Applied Mathematics

Julia: A Fresh Approach to Numerical Computing*

Jeff Bezanson[†]
Alan Edelman[‡]
Stefan Karpinski[§]
Viral B. Shah[†]



