

Functional Programming: Real World Performance, Nix and Warp Server

Wong Ding Feng

August 4, 2019

Contents

1	Who am I? Introduction to myself	2
1.1	My interests	2
1.2	For whom is this talk for?	2
2	The big problem, package manager wars!	2
2.1	slant.co's opinions	3
2.2	Some modern day package management systems	3
2.3	What about sub ecosystems?	4
2.4	How to make a package manager?	4
2.5	How to make a package manager?	4
2.5.1	real senario	4
2.6	Problems with modern package management	4
2.7	TODO Why imperative is bad? What is so imperative about installing packages?	5
2.8	Are you familiar with DEPENDENCY HELL ?	5
2.9	All types of "DEPENDENCY HELL"	5
2.10	Not Atomic 01	6
2.11	Not Atomic 02	6
2.12	Whats bad about imperative summary?	6
3	What it should/could/would have been?	7
3.1	GUIX vs Nix	7
3.2	Introducing Nix Package Management	7
3.3	Main mechanism	7
3.4	What you get for free with this mechanism?	7
3.4.1	no sudo , where is my sudo ?	8
3.4.2	easy revert, rollback	8
3.4.3	Select specific version	8
3.4.4	Installing and running 2 version of same software	8
3.4.5	Same development environment and runtime environment	8
3.5	Going all the way, NixOS	8
3.5.1	NixOS	9
3.5.2	docker	9
3.5.3	easy cd/dvd	9
3.5.4	easy vm	9
4	How does nix actually work?	9
4.1	Nix expressions	9
4.2	Language features	10
4.3	The main point	10
4.4	Example: Xmonad	11
4.5	Example: Xmonad	12
4.6	Main mechanism	13

5	Nix as infrastructure (imagination)	13
5.1	Main componenets	13
5.2	Caching build farm or cachix	14
6	references	14

1 Who am I? Introduction to myself

- Follow me on github! <https://github.com/TomatoCream>
- Linux user for 5 years now
 - Ubuntu
 - Proxmox
 - ArchLinux
 - Centos (server management)

1.1 My interests

- AI, ML
- Functional programming and abstraction (what the hell is so good about this?)
- philosophy
 - occam's razor


1.2 For whom is this talk for?

- Linux users! Sorry windows users
 - But not really (departs away from a unix way of doing things)
- Show you what functional programming can do?
 - purity?
 - referential transparency?
- State management
- DevOps
- Images, Docker, VM, Clusters
- I will give you a feel of Nix not the nitty gritty details

2 The big problem, package manager wars!

- What is the best package manager?
- Has anyone ever used some sort of package management system?

2.1 slant.co's opinions






 [Log in](#) • [Sign up](#) [ADD QUESTION](#)

DEVELOPMENT LINUX [FOLLOW](#) [...](#)

What are the best Linux package managers?

21 OPTIONS CONSIDERED 548 USER RECS. Jul 27, 2019 LAST UPDATED

THE BEST 2 OF 15 OPTIONS [i WHY?](#)

BEST LINUX PACKAGE MANAGERS		PRICE	PLATFORMS	DEFAULT REPOSITORY
94	 Nix	-	-	-
94	 Pacman	-	-	-
84	 Portage	-	-	-
81	 DPKG family (apt, aptit...	-	-	-
77	 DNF	-	-	-

[SEE FULL LIST](#)

[I RECOMMEND...](#) [v](#)

94 **EXCELLENT** // Nix [...](#)

2.2 Some modern day package management systems

Package manager	Distributions
apt, apt-get	Debian, Ubuntu
rpm, yum	Redhat, Centos
pacman	ArchLinux
brew	MacOS

2.3 What about sub ecosystems?

Package manager	???
pip, virtualenv, pipenv	Python2,3(???)
npm, yarn	Nodejs
cabal, stack, hackage	Haskell :)
go?	go?
brew	MacOS
use-package, vim, fish, zsh	...

2.4 How to make a package manager?

- What are the basic parts that we need?

2.5 How to make a package manager?

build dependencies	What do I need to build the program?
runtime dependencies	What <code>.so</code> shared objects do I need?
configurations	What in <code>/etc/...</code> config files

- essentially think of it as a graph, whenever we upgrade or install a package, we are mutating a node on this graph to point to something else.

2.5.1 real senario

```
pkgname=pacman
pkgver=5.1.0
_pkgver=1.0.0
pkgrel=2
pkgdesc="A library-based package manager with dependency support"
arch=('i686' 'x86_64')
url="http://www.archlinux.org/pacman/"
license=('GPL')
groups=('base' 'base-devel')
depends=('bash>=4.2.042-2' 'glibc>=2.17-2' 'libarchive>=3.1.2' 'curl>=7.39.0'
        'gpgme' 'archlinux-keyring' 'manjaro-keyring' 'pacman-mirrors>=4.1.0')
checkdepends=('python2' 'fakechroot')
makedepends=('asciidoc' 'pacman>=5.1')
optdepends=('haveged: for pacman-init.service')
provides=('pacman-contrib' 'pacman-init')
conflicts=('pacman-contrib' 'pacman-init')
replaces=('pacman-contrib' 'pacman-init')
backup=(etc/pacman.conf etc/makepkg.conf)
install=pacman.install
options=('strip' 'debug')
```

2.6 Problems with modern package management

https://wiki.debian.org/DontBreakDebian#Don.27t_make_a_FrankenDebian

Contents

1. Advice For New Users On Not Breaking Their Debian System

1. Don't make a FrankenDebian
2. Don't use GPU manufacturer install scripts
3. Don't suffer from Shiny New Stuff Syndrome
4. 'make install' can conflict with packages
5. Don't blindly follow bad advice
6. Read The Fantastic Manuals
7. Don't blindly remove software
8. Read package descriptions before installing
9. Take notes
10. Some safer ways to install software not available in Debian Stable
 1. Backported packages
 2. Building from source
 3. Using chroot, containers, and virtual machines
 1. Flatpak
 2. Snap
11. Get the most out of peer support resources

2. See Also

2.7 TODO Why imperative is bad? What is so imperative about installing packages?

referential transparency

2.8 Are you familiar with DEPENDENCY HELL?

- https://www.reddit.com/r/ProgrammerHumor/comments/75txp4/nodejs_dependency_hell_visualized_for_the_first/?utm_source=share&utm_medium=web2x
- <https://github.com/vector-im/riot-web/network/dependencies>

2.9 All types of "DEPENDENCY HELL"

https://miro.medium.com/max/984/0*7ezJ0tYUkI5zyqWU.png

- { DLL, dependency, npm, cabal } hell, different names for the same demon
- conflicting dependency
 - shared components like library links `cuda.7.so` vs `cuda.6.so`
- multiple version side by side and roll backs
- possible solutions
 - set of stable packages like Debian or haskell stack snapshots

2.10 Not Atomic 01

- kill upgrades half way
 - packages left in a semi updated state
 - sometimes need to manually remove lock files

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
dpkg	29329	root	3uW	REG	8,7	0	262367	/var/lib/dpkg/lock

2.11 Not Atomic 02

- can be fixed but kinda troublesome.

You can give a try to fix your problem using Recovery Mode if none of the method works for you. If you are using wired connection then no problem, for wireless connection I'm not sure whether it will work or not, because I never tried to enable network in Recovery Mode when using wireless connection. Although you can give it a try.

- When your system starts chose **Recovery Mode** (2nd option in grub menu).
- From the Menu just go to **Grub** option, it will give a message like **Updating grub will mount your system in read/write mode**. Just chose **yes** to mount your system in read/write mode. It will update your grub and will exit from **Grub** menu.
- chose **network** option it may enable your network.
- Then chose **dpkg** menu from the list, chose **yes** for all.
- Finally chose **root** option and login. Execute following commands one after another:

```
# apt-get autoremove
# apt-get autoclean
# apt-get update
# apt-get -f install
# apt-get dist-upgrade
# apt-get upgrade
```

Then reboot your system and check whether your issues are fixed or not. Run this command to reboot:

```
# reboot
```

Reply if something goes wrong.s

2.12 Whats bad about imperative summary?


- No referential transparency
 - cannot point to older versions of the same thing
- Dependency hell
 - conflicting dependencies
- Not atomic upgrades
 - unknown state if break half way

These problems are really similar to the problems with imperative languages! like **JAVA** and people have already made solutions for them like how **Haskell** does. We could learn a thing or two from them.

3 What it should/could/would have been?

- Imagine now that we implemented all the things of a functional programming language to create a functional package management system?
- What can we do with this?


3.1 GUIX vs Nix



OverviewDownload

- ★ **Liberating.** Guix is an advanced distribution of the GNU operating system developed by the GNU Project—which respects the freedom of computer users.
- ★ **Dependable.** Guix supports transactional upgrades and roll-backs, unprivileged package management, and more. When used as a standalone distribution, Guix supports declarative system configuration for transparent and reproducible operating systems.
- ★ **Hackable.** It provides Guile Scheme APIs, including high-level embedded domain-specific languages (EDSLs) to define packages and whole-system configurations.

DOWNLOAD v1.0.1CONTRIBUTE



Nix

The Purely Functional Package Manager

Nix is a powerful package manager for Linux and other Unix systems that makes package management reliable and reproducible. It provides atomic upgrades and rollbacks, side-by-side installation of multiple versions of a package, multi-user package management and easy setup of build environments. [Read more...](#)

Get Nix

3.2 Introducing Nix Package Management

- solves all of the problems above
 - No referential transparency
 - * cannot point to older versions of the same thing
 - Dependency hell
 - Not atomic upgrades
 - * unknown state if break half way

3.3 Main mechanism

- referential transparency
 - install everything in path `/nix/store/{hash}-name`
 - via symlinking

3.4 What you get for free with this mechanism?

- no sudo
- easy revert and roll back
- select specific version
- 2 different version can run at the same time

- same **development** environment as the **runtime** environment!

- nix-shell

3.4.1 no sudo, where is my sudo?

- linux was developed as a **time sharing** system
- many users were expected to share a single computer.
- thus to manage conflicts, a **super user**, **root** was required to install and manage packages

```
nix-env -iA nixos.figlet
```

3.4.2 easy revert, rollback

```
figlet "I am here!"
```

```
nix-env --rollback
```

```
figlet "are you still here?"
```

3.4.3 Select specific version

```
cd ~/projects/nix-config/
```

```
git checkout ??
```

```
nix-env -f ~/projects/nix-config/ -iA screenfetch
```

screenfetch 2016 vs current

3.4.4 Installing and running 2 version of same software

```
stack --version
```

```
su
```

```
stack --version
```

3.4.5 Same development environment and runtime environment

- I am not an electrical engineer or something but I program my own keyboard. So I need some sort of firmware flasher. like **dfuprogrammer** I dont need it on my system.

```
cd ~/projects/qmk_firmware/
```

```
make
```

```
dfuprogrammer
```

```
nix-shell
```

```
make
```

```
dfuprogrammer
```

3.5 Going all the way, NixOS

- whole system management via Nix and thus NixOS
 - Version controlled operating system
 - show OS reboot
 - I wanted to show my generations so had been delaying removing my older generations

```
df -h /
```

```
nix-collect-garbage --delete-older-than 10 --dry-run
```


3.5.1 NixOS

- show file:///home/df/nix-config/configuration.nix
- python package management file:///home/df/nix-config/configuration.nix
- gnupg agent file:///home/df/nix-config/configuration.nix
- ports file:///home/df/nix-config/configuration.nix
 - I think it helps me get a state of all the ports in one place
- users and security all in one place file:///home/df/nix-config/configuration.nix
 - authorisedkeys
- postgresql can be packaged in shell.nix file:///home/df/nix-config/configuration.nix
 - separate project called nixos-shell <https://github.com/chrisfarms/nixos-shell>
- filesystems file:///etc/nixos/hardware-configuration.nix

3.5.2 docker

<https://nixos.wiki/wiki/Docker>

```
virtualisation.docker.enable = true;  
users.users.<myuser>.extraGroups = [ "docker" ];
```

```
nix-build '<nixpkgs>' -A dockerTools.examples.redis  
docker load < result
```

<https://github.com/NixOS/nixpkgs/blob/master/pkgs/build-support/docker/examples.nix>

3.5.3 easy cd/dvd

```
cd ~/projects/nixpkgs  
nix-build -A config.system.build.isoImage -I nixos-config=modules/installer/cd-dvd/installation-cd-minimal
```

3.5.4 easy vm

```
cd ./nixops  
nixops create -d simple02 network.nix  
nixops deploy -d simple02
```

```
deployment.targetEnv = "ec2";  
deployment.region = "eu-west-1";
```

4 How does nix actually work?

4.1 Nix expressions

- functional expressions, not general purpose please do not program things with it
- comes with its own BNF grammar

expressions

e ::= x	identifier
nat str	literal
[e*]	list
rec [?] {b*}	(optionally recursive) attribute set
let b* in e	local declarations
e.x	attribute selection
x : e	plain λ -abstraction
{fs [?] } : e	λ -abstraction pattern-matching an attribute set
e e	function application
if e then e else e	conditional
with e ₁ ; e ₂	add attributes from set e ₁ to lexical scope of e ₂
(e)	grouping

bindings

b ::= ap = e;	allows concise nested attribute sets, e.g. x.y.z = true
inherit x*;	copy value of attribute x from lexical scope

attribute path

ap ::= x.ap | x

formals

fs ::= x, fs | x | ... "... means ignore unmatched attributes

string literals

str ::= "(char \${e})*"	strings; interpolation using \${e}
'''(char \${e})*'''	strings with common indentation removed
uri	Uniform Resource Identifiers (Berners-Lee <i>et al.</i> , 1998)

Fig. 1. Informal partial syntax overview of the Nix expression language

4.2 Language features

- Nix expressions
 - dynamically typed
 - lazy
 - pure

4.3 The main point

- Nix expressions are here to describe a graph of build actions called **derivations**
 - build script
 - set of environment variables
 - set of dependencies

4.4 Example: Xmonad

```
{ stdenv, fetchurl, ghc, X11, xmessage }: ❶

stdenv.mkDerivation ❷ (rec {
  name = "xmonad-0.5";

  src = fetchurl {
    url = "http://hackage.haskell.org/.../${name}.tar.gz";
    sha256 = "1i74az7w7nbirw6n6lcm44vf05hjq1yyhnsssc779yh0n00lbk6g";
  };

  buildInputs = [ ghc X11 ]; ❸

  configurePhase = '' ❹
    substituteInPlace XMonad/Core.hs --replace \
      "xmessage" "${xmessage}/bin/xmessage" ❺
    ghc --make Setup.lhs
    ./Setup configure --prefix="$out" ❻
  '';

  buildPhase = ''
    ./Setup build
  '';

  installPhase = ''
    ./Setup copy
    ./Setup register --gen-script
  '';

  meta = { ❼
    description = "A tiling window manager for X";
  };
})
```

Fig. 2. xmonad.nix: Nix expression for xmonad

```
rec {  
  xmonad = import ../xmonad.nix { 8  
    inherit stdenv fetchurl ghc X11 xmessage;  
  };  
  
  xmessage = import ../xmessage.nix { ... };  
  
  ghc = ghc68;  
  
  ghc68 = import ../development/compilers/ghc-6.8 {  
    inherit fetchurl stdenv readline perl gmp ncurses m4;  
    ghc = ghcboot;  
  };  
  
  ghcboot = ...;  
  stdenv = ...;  
  ...  
}
```

Fig. 4. all-packages.nix: Function calls to instantiate packages

4.6 Main mechanism

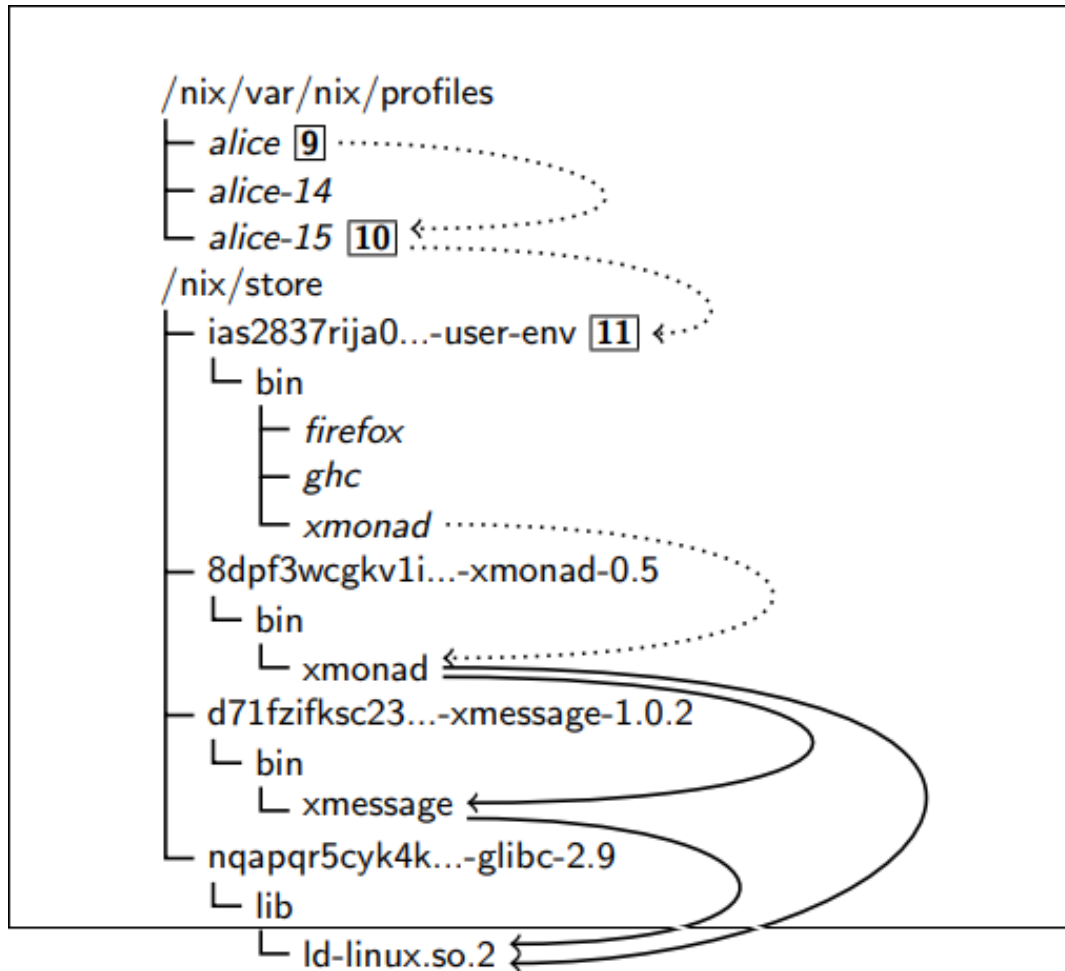


Fig. 5. The Nix store, containing xmonad and some of its dependencies, and profiles


5 Nix as infrastructure (imagination)

- how might one use nix in JPMC's infrastructure?

5.1 Main componenets

- Hydra caching
- Dependency management
- Ease of use
 - nix-shell
- Security



5.2 Caching build farm or cachix

 Hydra ▾

Hydra

About

Hydra is a Nix-based continuous build system, released under the terms of the [GNU GPLv3](#) or (at your option) any later version. It continuously checks out sources of software projects from version



Build Nix packages once and share them for good.

6 references

- [HTML] Nix: A Safe and Policy-Free System for Software Deployment.
 - E Dolstra, M De Jonge, E Visser - [usenix.org](https://nixos.org/~eelco/pubs/nspfssd-lisa2004-final.pdf)
 - <https://nixos.org/~eelco/pubs/nspfssd-lisa2004-final.pdf>
- [PDF] A Purely Functional Linux Distribution - NixOS
 - E Dolstra
 - <https://nixos.org/~eelco/pubs/nixos-jfp-final.pdf>
- Hydra - NixOS
 - <https://nixos.org/~eelco/pubs/hydra-scp-submitted.pdf>