NixOS: A Purely Functional Linux Distribution

ICFP 2008, Victoria, Canada

Eelco Dolstra[1] Andres Löh[2]

[1]Delft University of Technology, EWI,
Department of Software Technology

[2]Utrecht University,
Department of Information and Computing Sciences

September 24, 2008

Universiteit Utrecht

Overview

- Operating systems are installed and managed using tools that have an *imperative model*
- This causes lots of problems: upgrading is unreliable, rollbacks are hard, etc.
- This paper shows that it is possible to implement a system with a *purely functional model*
- Implemented in a Linux distribution called *NixOS*

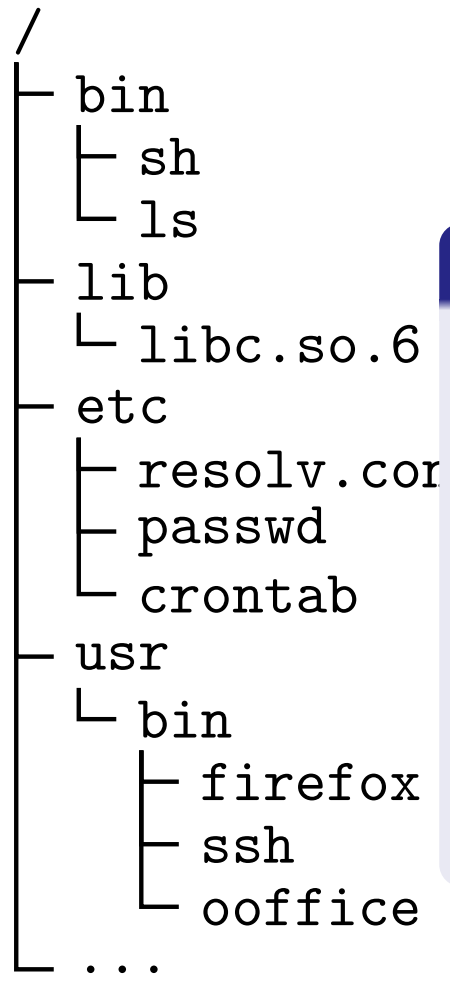
Introduction

Your operating system and applications are managed in an **imperative way**.

```
/
├─ bin
│  ├─ sh
│  └─ ls
├─ lib
│  └─ libc.so.6
├─ etc
│  ├─ resolv.conf
│  ├─ passwd
│  └─ crontab
├─ usr
│  └─ bin
│     ├─ firefox
│     ├─ ssh
│     └─ ooffice
└─ ...
```

Introduction

Your operating system and applications are managed in an **imperative way**.



Examples

- Package managers (e.g. RPM) perform *destructive updates* to the filesystem while doing upgrades.
- Packages have post-install scripts that scribble all over `/etc`.
- Sysadmins make manual changes to config files in `/etc`.

Why is statefulness bad?

- No *predictability* (determinism)
  - If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - This is why upgrading is riskier than a full re-install
- Configuration actions clobber the previous configuration
  - No multiple versions
  - No atomicity
  - No rollbacks
  - Hard to safely test a configuration
- No *traceability*
  - Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - Hard to reproduce a configuration

Why is statefulness bad?

- No *predictability* (determinism)
  - If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - This is why upgrading is riskier than a full re-install
- Configuration actions clobber the previous configuration
  - No multiple versions
  - No atomicity
  - No rollbacks
  - Hard to safely test a configuration
- No *traceability*
  - Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - Hard to reproduce a configuration

Why is statefulness bad?

- No *predictability* (determinism)
  - If an action depends on an ill-defined initial state, then the result is probably ill-defined
  - This is why upgrading is riskier than a full re-install
- Configuration actions clobber the previous configuration
  - No multiple versions
  - No atomicity
  - No rollbacks
  - Hard to safely test a configuration
- No *traceability*
  - Configuration is the result of a sequence of (sometimes manual) imperative actions over time
  - Hard to reproduce a configuration

There is a better way!

- Solution: **borrow from purely functional languages**.
- We're going to...
  - Build packages from pure functions
  - Store packages as immutable values in the filesystem
  - Extend this to the other static bits of the system

There is a better way!

- Solution: **borrow from purely functional languages**.
- We're going to...
  - Build packages from pure functions
  - Store packages as immutable values in the filesystem
  - Extend this to the other static bits of the system

Old stuff: Nix

There is a better way!

- Solution: **borrow from purely functional languages**.
- We're going to...
  - Build packages from pure functions
  - Store packages as immutable values in the filesystem
  - Extend this to the other static bits of the system

Old stuff: Nix

New stuff: NixOS

Nix: Purely functional package management

Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they're never overwritten by newer versions.
  - Unused packages are garbage collected automatically.

Nix: Purely functional package management

Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they're never overwritten by newer versions.
  - Unused packages are garbage collected automatically.

Nix: Purely functional package management

Nix is a purely functional package manager.

- Lazy, higher-order, dynamically typed, purely functional language to describe how to build and compose packages.
- Build results only depend on declared inputs.
- Packages are stored as a kind of purely functional data structure:
  - They never change after they have been built.
  - So in particular they're never overwritten by newer versions.
  - Unused packages are garbage collected automatically.

Nix store

Central idea: store all packages in isolation from each other:

`/nix/store/axrzx0rh0ivw...-firefox-2.0.0.3`

Paths contain a 160-bit **cryptographic hash** of **all** inputs used to build the package:

- Sources
- Libraries
- Compilers
- Build scripts
- ...

```
/nix/store
├── l9w6773m1msy...-openssh-4.6p1
│   ├── bin
│   │   └── ssh
│   └── sbin
│       └── sshd
├── smkabrbibqv7...-openssl-0.9.8e
│   └── lib
│       └── libssl.so.0.9.8
├── c6jbqm2mc0a7...-zlib-1.2.3
│   └── lib
│       └── libz.so.1.2.3
└── im276akmsrhv...-glibc-2.5
    └── lib
        └── libc.so.6
```

Nix expressions

Nix expressions describe how to build packages.

```
{stdenv, fetchurl, openssl, zlib}:

stdenv.mkDerivation {
  name = "openssh-4.6p1";
  src = fetchurl {
    url = http://.../openssh-4.6p1.tar.gz;
    sha256 = "0fpjlr3bfind0y94bk442x2p...";
  };
  buildCommand = ''
    tar xjf $src
    ./configure --prefix=$out --with-openssl=${openssl}
    make; make install
  '';
}
```

Nix expressions

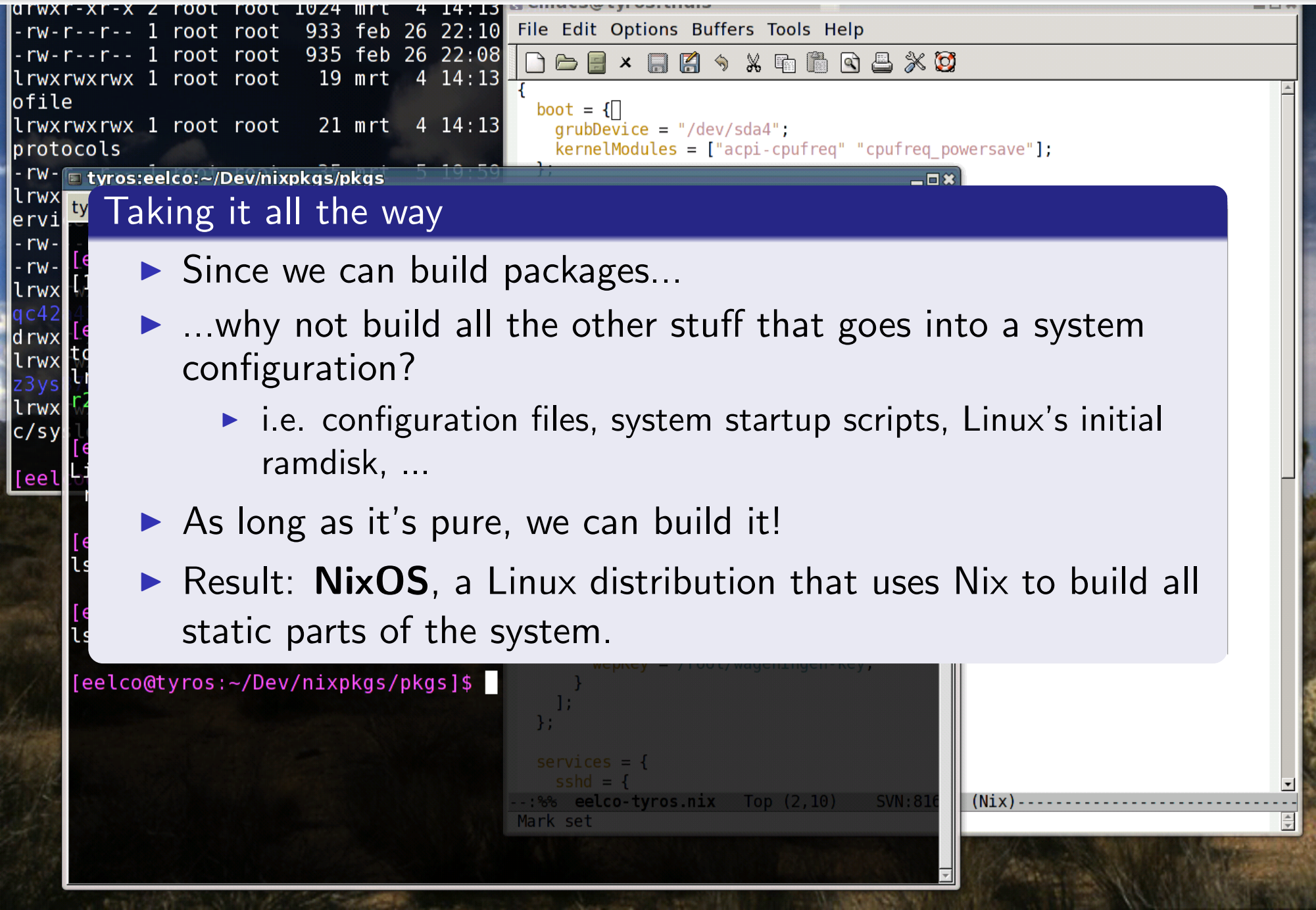
Nix expressions describe how to build packages.

```
{stdenv, fetchurl, openssl, zlib}:

stdenv.mkDerivation {
  name = "openss
  src = fetchurl
    url = http:/
    sha256 = "0f
  };
  buildCommand =
    tar xjf $src
    ./configure
    make; make i
  '';
}
```

Evaluating the result of this function will produce an OpenSSH package in the Nix store.

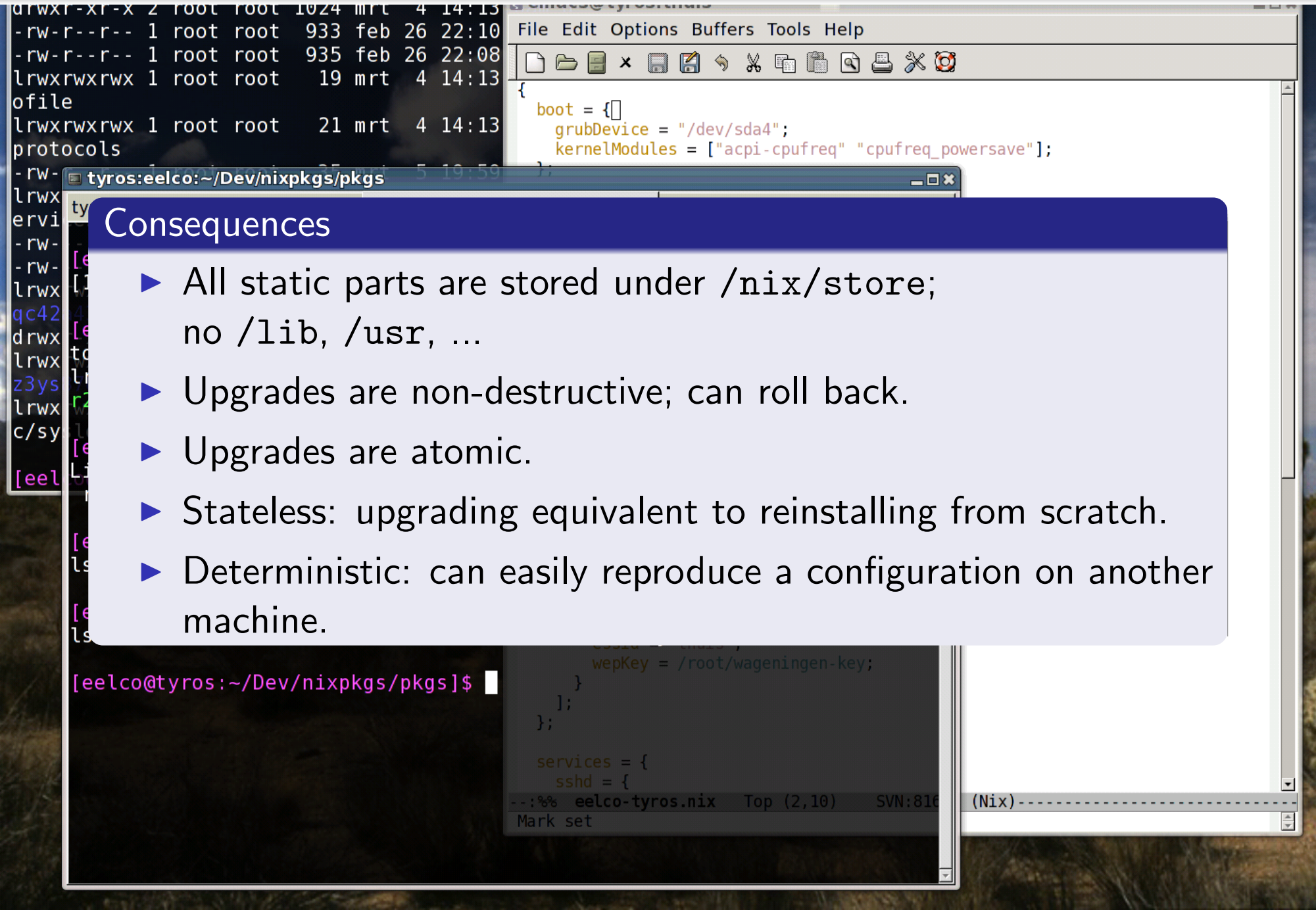
```
/nix/store
├─ 19w6773m1msy...-openssh-4.6p1
│   ├─ bin
│   │   └─ ssh
│   └─ sbin
│       └─ sshd
└─ ...
```



Taking it all the way

- Since we can build packages...
- ...why not build all the other stuff that goes into a system configuration?
  - i.e. configuration files, system startup scripts, Linux's initial ramdisk, ...
- As long as it's pure, we can build it!
- Result: **NixOS**, a Linux distribution that uses Nix to build all static parts of the system.

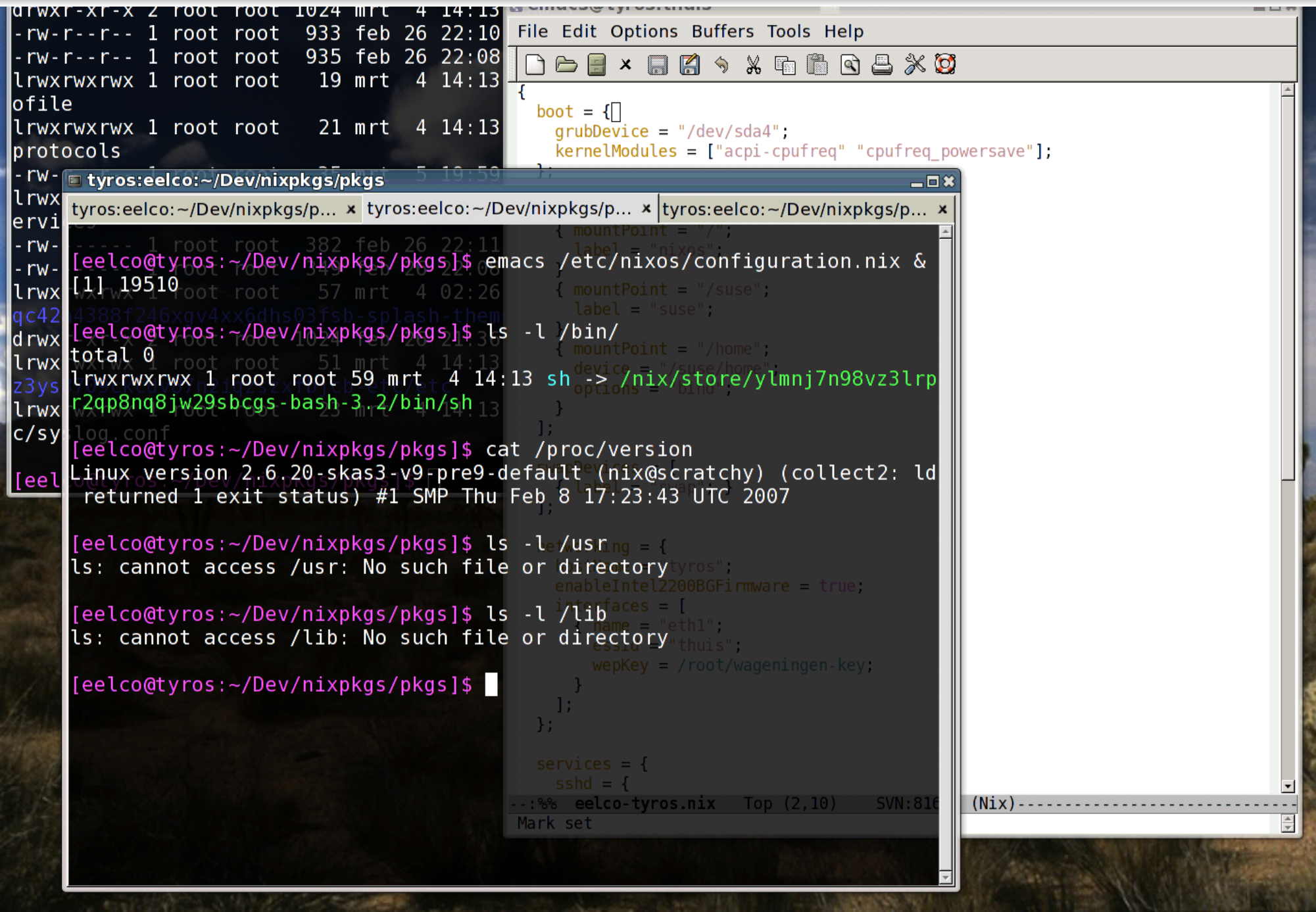
NixOS



Consequences

- All static parts are stored under `/nix/store`; no `/lib`, `/usr`, ...
- Upgrades are non-destructive; can roll back.
- Upgrades are atomic.
- Stateless: upgrading equivalent to reinstalling from scratch.
- Deterministic: can easily reproduce a configuration on another machine.

NixOS


File Edit Options Buffers Tools Help
boot = {
grubDevice = "/dev/sda4";
kernelModules = ["acpi-cpufreq" "cpufreq_powersave"];
tyros:eelco:~/Dev/nixpkgs/pkgs
tyros:eelco:~/Dev/nixpkgs/p...
tyros:eelco:~/Dev/nixpkgs/p...
tyros:eelco:~/Dev/nixpkgs/p...
[eelco@tyros:~/Dev/nixpkgs/pkgs]$ emacs /etc/nixos/configuration.nix &
[1] 19510
[eelco@tyros:~/Dev/nixpkgs/pkgs]$ ls -l /bin/
total 0
lrwxrwxrwx 1 root root 59 mrt 4 14:13 sh -> /nix/store/ylmnj7n98vz3lrpr2qp8nq8jw29sbcgs-bash-3.2/bin/sh
[eelco@tyros:~/Dev/nixpkgs/pkgs]$ cat /proc/version
Linux version 2.6.20-skas3-v9-pre9-default (nix@scratchy) (collect2: ld returned 1 exit status) #1 SMP Thu Feb 8 17:23:43 UTC 2007
[eelco@tyros:~/Dev/nixpkgs/pkgs]$ ls -l /usr
ls: cannot access /usr: No such file or directory
[eelco@tyros:~/Dev/nixpkgs/pkgs]$ ls -l /lib
ls: cannot access /lib: No such file or directory
[eelco@tyros:~/Dev/nixpkgs/pkgs]$
--:%% eelco-tyros.nix Top (2,10) SVN:816
Mark set
(Nix)

Example

Nix expression for ssh_config

```
{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.services.sshd.forwardX11 then ''
    ForwardX11 yes
    XAuthLocation ${pkgs.xorg.xauth}/bin/xauth
  '' else ''
    ForwardX11 no
  ''}
''
```

Example

Nix expression for ssh_config

```
{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.services.sshd.forwardX11 then ''
    ForwardX11 yes
    XAuthLocation ${pkgs.xorg.xauth}/bin/xauth
  '' else ''
    For
  ''}
''
```

Laziness in action!

Example

Nix expression for ssh_config

```
{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.serv
    ForwardX11 yes
    XAuthLocation
  '' else ''
    ForwardX11 no
  ''}
''
```

Nix store

```
/nix/store
├─ 33lcnh62yll3...-ssh_config
└─ kyv6n69a40q6...-xauth-1.0.2
   └─ bin
      └─ xauth
```

Example

Nix expression for ssh_config

```
{config, pkgs}:

pkgs.writeText "ssh_config" ''
  SendEnv LANG LC_ALL ...
  ${if config.serv
    ForwardX11 yes
    XAuthLocation
  '' else ''
    ForwardX11 no
```

Nix store

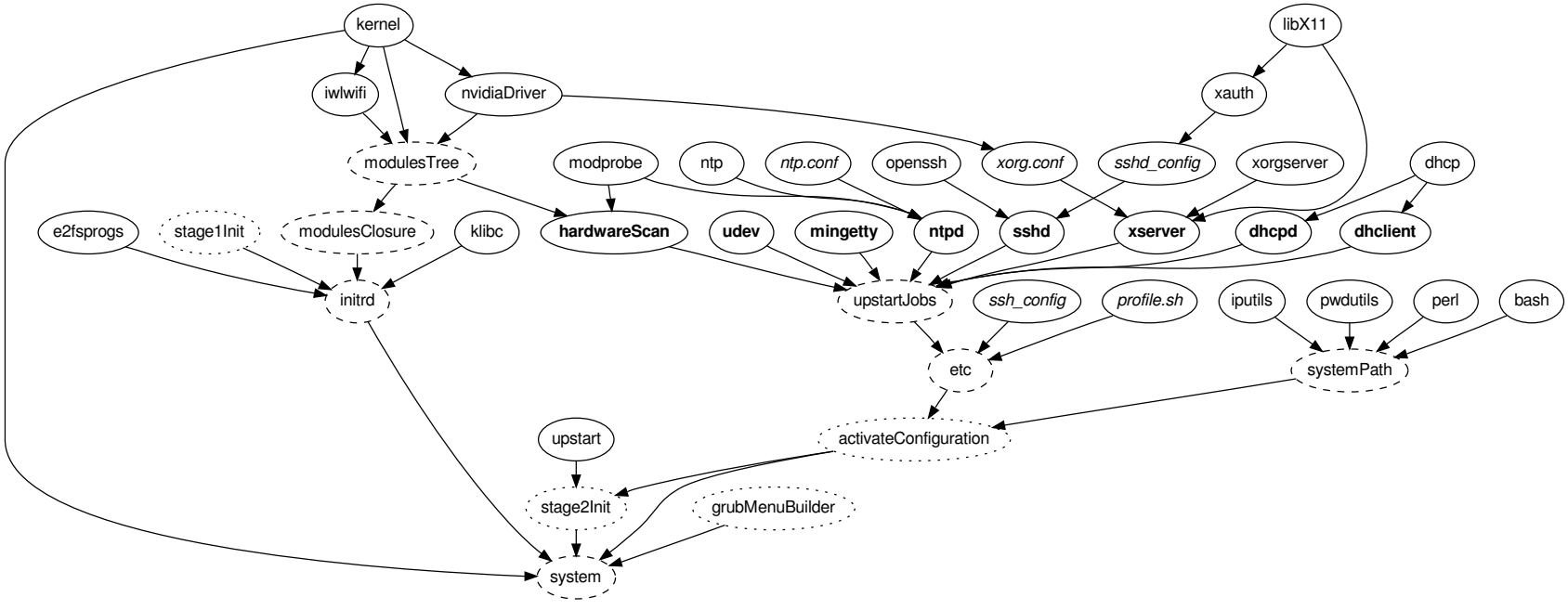
```
/nix/store
├ 33lcnh62yll3...-ssh_config
└ kyv6n69a40q6...-xauth-1.0.2
  └ bin
```

Generated file: 33lcnh62yll3...-sshd_config

```
SendEnv LANG LC_ALL ...
ForwardX11 yes
XAuthLocation /nix/store/kyv6n69a40q6...-xauth-1.0.2/bin/xauth
```

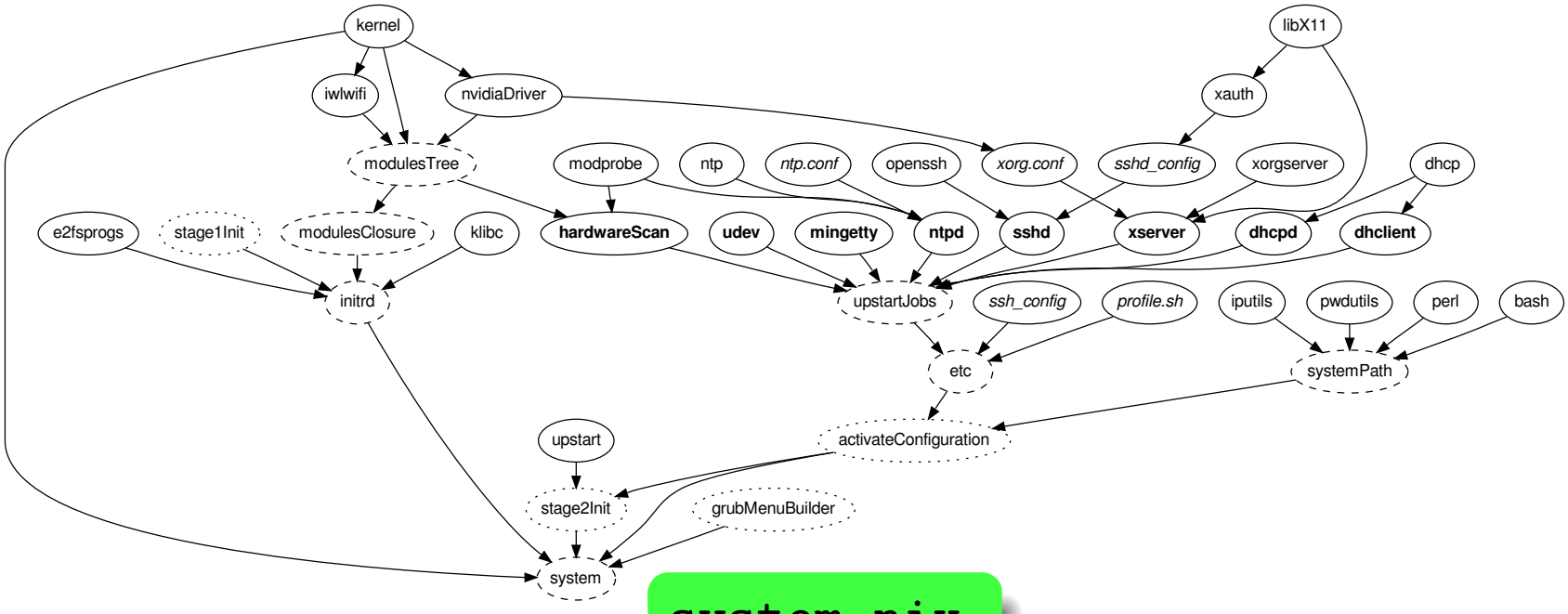
NixOS build time dependency graph

Nix expressions to build each part of the system: system packages, applications, their dependencies, kernel modules, initrd, configuration files, Upstart jobs, boot scripts, ...



NixOS build time dependency graph

Nix expressions to build each part of the system: system packages, applications, their dependencies, kernel modules, initrd, configuration files, Upstart jobs, boot scripts, ...



The system configuration file

/etc/nixos/configuration.nix

```
{
  boot = { grubDevice = "/dev/sda"; };
  fileSystems = [
    { mountPoint = "/";
      device = "/dev/sda1";
    }
  ];
  swapDevices = [ { device = "/dev/sdb1"; } ];
  services = {
    sshd = {
      enable = true;
      forwardX11 = true;
    };
  };
}
```

The system configuration file

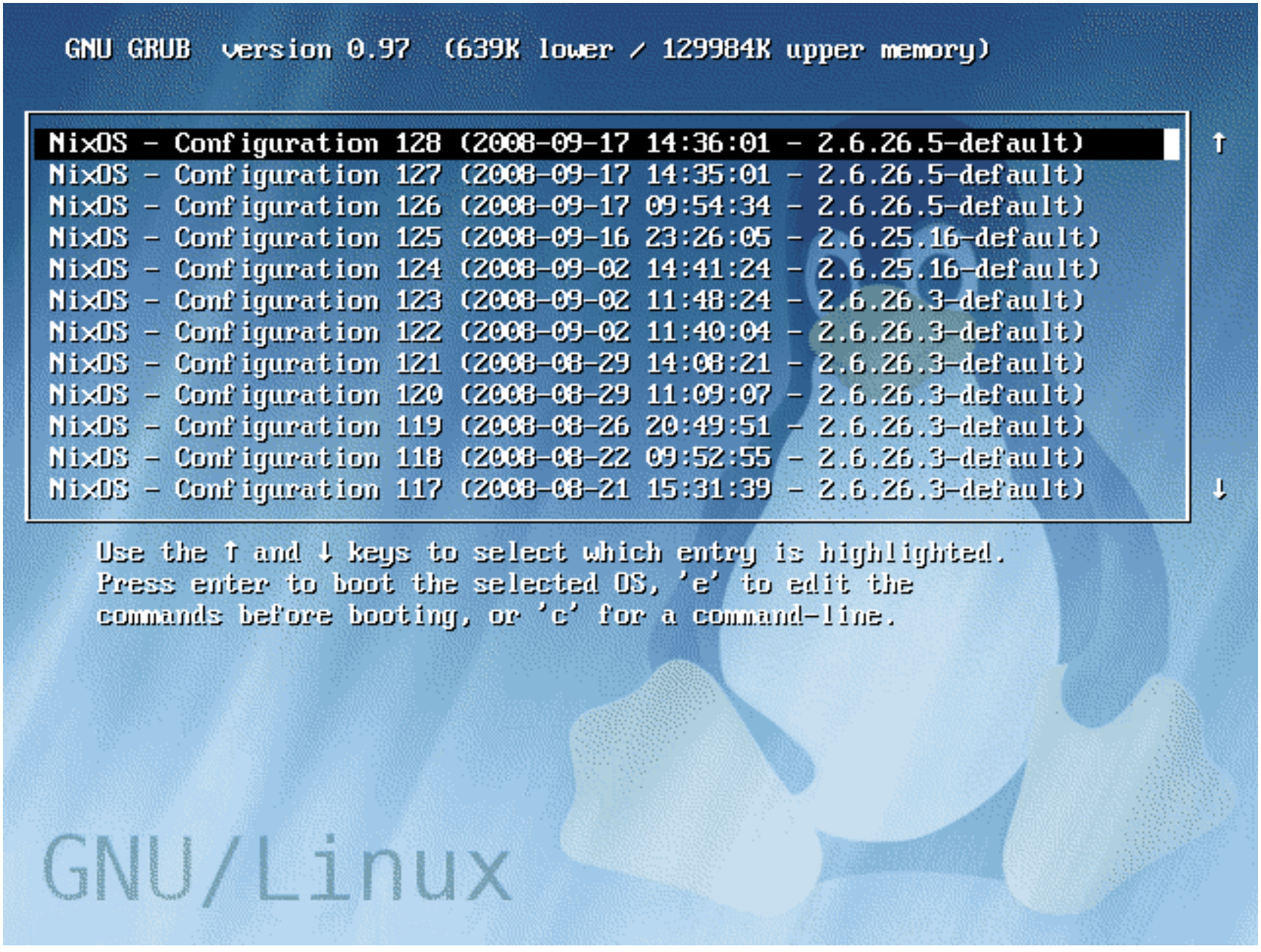
/etc/nixos/configuration.nix

```
{
  boot = { grubDevice = "/dev/sda"; };
  fileSystems = [
    { mountPoint = "/";
      device = "/dev/sda1";
    }
  ];
  swapDevices = [
  services = {
    sshd = {
      enable = tru
      forwardX11 =
    };
  };
}
```

End-user perspective

- Edit `configuration.nix`.
- Run `nixos-rebuild`.
- This builds `system.nix` and runs its activation script.
- Non-destructive; various rollback mechanisms.

NixOS — Grub boot menu



Evaluation: How pure are we?

Code

- No `/bin`, `/sbin`, `/lib`, `/usr`.
- Only exception: `/bin/sh`
- Only minor changes needed to a few packages.

```
[eelco@dutibo:~]$ ls -l /bin/
total 1
lrwxrwxrwx 1 root root 63 2008-09-24 02:09 sh -> /nix/store/a46s354jbdmj

[eelco@dutibo:~]$ ls -l /usr
ls: cannot access /usr: No such file or directory

[eelco@dutibo:~]$ ls -l /lib
ls: cannot access /lib: No such file or directory
```

Evaluation: How pure are we?

Configuration data

- Almost all of `/etc` resides in the Nix store
- E.g. `sshd_config`, Upstart job specifies full store path.
- But some configuration files are cross-cutting (`/etc/nsswitch.conf`, `/etc/services`), so we symlink them in `/etc`.

```
-rw-r--r-- 1 root root  276 2008-09-24 02:09 group
-rw-r--r-- 1 root root  267 2008-09-24 02:09 group.old
lrwxrwxrwx 1 root root   17 2008-09-24 02:09 hosts -> /etc/static/hosts
lrwxrwxrwx 1 root root   19 2008-09-24 02:09 inputrc -> /etc/static/inputrc
lrwxrwxrwx 1 root root   17 2008-09-24 02:09 issue -> /etc/static/issue
lrwxrwxrwx 1 root root   22 2008-09-24 02:09 login.defs -> /etc/static/logi
drwxr-xr-x 3 root root 1024 2008-01-07 14:52 lvm
-rw-r--r-- 1 root root  280 2008-09-24 19:33 mtab
lrwxrwxrwx 1 root root   20 2008-09-24 02:09 nix.conf -> /etc/static/nix.co
```

Evaluation: How pure are we?

Mutable state

- E.g. `/var/log`.
- Handled normally; not part of the purely functional model.
- Nasty: hybrid configuration+state: `/etc/passwd`; handled by the imperative activation script.

```
[eelco@dutibo:~]$ ls -l /var/
total 11
drwxr-xr-x 6 root root 1024 2008-09-24 17:10 cache
drwx--x--- 3 root root 1024 2008-01-07 14:52 cron
drwxr-xr-x 4 root root 1024 2008-06-06 10:20 db
dr-xr-xr-x 2 root root 1024 2008-01-07 14:52 empty
drwxr-xr-x 7 root root 1024 2008-06-06 10:48 lib
drwx------ 3 root root 1024 2008-01-07 14:52 lock
drwxr-xr-x 5 root root 1024 2008-09-23 03:15 log
drwxr-xr-x 8 root root 1024 2008-09-24 02:09 run
```

Evaluation: How pure are we?

Build actions

- Can't *guarantee* that build actions are pure.
- E.g. output could use
  - System time
  - The network
  - Files outside the Nix store
- Experiment: built 485 packages on two machines.
- Result: out of 165 927 files, 5059 differed.
- Almost all (possibly all) are due to timestamps or the hostname.

Conclusion

- Applied lazy, purely functional paradigm to a very different domain — system configuration management.
  - Purity and laziness are incredibly useful here.
- NixOS shows that a purely functional system configuration model is feasible and practical.
- Advantages: reproducibility, predictable upgrading, rollbacks, multi-user package management, ...

More information / download

- `http://nixos.org/`
- ISO images for x86, x86_64 are available.