Automated Software Testing and Release with Nix Build Farms

Eelco Dolstra[1] Eelco Visser[2]

[1]Universiteit Utrecht, Faculty of Science,
Department of Information and Computing Sciences

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

March 23, 2007

Universiteit Utrecht

TU Delft

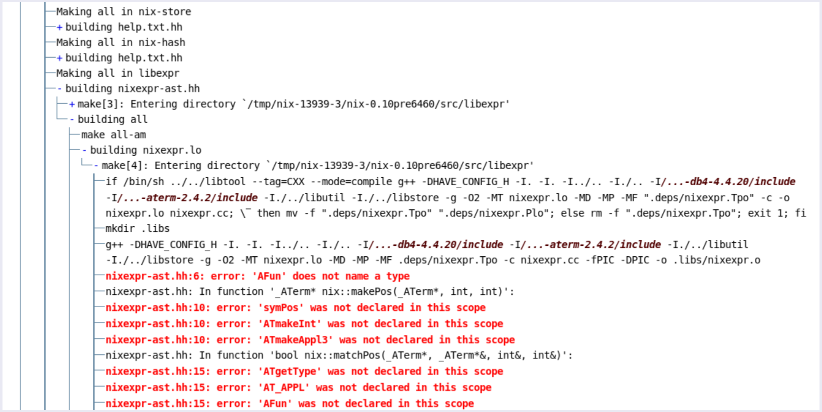
Technische Universiteit Delft

Build Farms

Build farm: a set of machines that continuously builds and tests software components from a version management system, producing status reports and/or releases.

Build farm goals

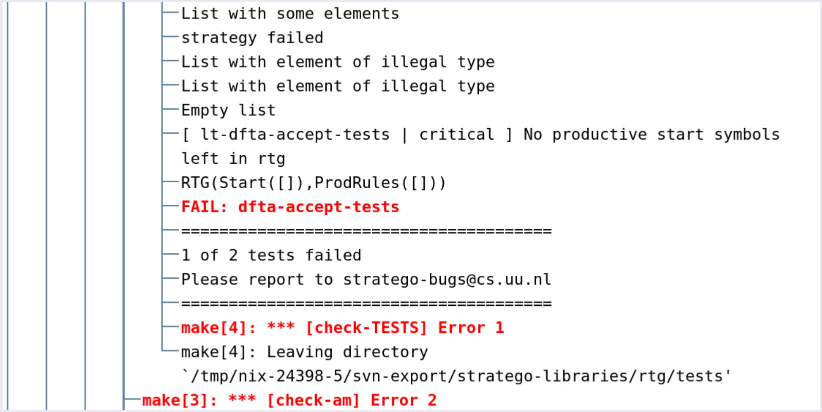
Goal: Building



```
├─Making all in nix-store
├─+ building help.txt.hh
├─Making all in nix-hash
├─+ building help.txt.hh
├─Making all in libexpr
├─- building nixexpr-ast.hh
│ ├─+ make[3]: Entering directory `/tmp/nix-13939-3/nix-0.10pre6460/src/libexpr'
│ └─- building all
│   ├─make all-am
│   ├─- building nixexpr.lo
│   │ └─- make[4]: Entering directory `/tmp/nix-13939-3/nix-0.10pre6460/src/libexpr'
│   │     ├─if /bin/sh ../../libtool --tag=CXX --mode=compile g++ -DHAVE_CONFIG_H -I. -I. -I../.. -I../.. -I/...-db4-4.4.20/include
│   │     │ -I/...-aterm-2.4.2/include -I../../libutil -I../../libstore -g -O2 -MT nixexpr.lo -MD -MP -MF ".deps/nixexpr.Tpo" -c -o
│   │     │ nixexpr.lo nixexpr.cc; \¯ then mv -f ".deps/nixexpr.Tpo" ".deps/nixexpr.Plo"; else rm -f ".deps/nixexpr.Tpo"; exit 1; fi
│   │     ├─mkdir .libs
│   │     ├─g++ -DHAVE_CONFIG_H -I. -I. -I../.. -I../.. -I/...-db4-4.4.20/include -I/...-aterm-2.4.2/include -I../../libutil
│   │     │ -I../../libstore -g -O2 -MT nixexpr.lo -MD -MP -MF .deps/nixexpr.Tpo -c nixexpr.cc -fPIC -DPIC -o .libs/nixexpr.o
│   │     ├─nixexpr-ast.hh:6: error: 'AFun' does not name a type
│   │     ├─nixexpr-ast.hh: In function '_ATerm* nix::makePos(_ATerm*, int, int)':
│   │     ├─nixexpr-ast.hh:10: error: 'symPos' was not declared in this scope
│   │     ├─nixexpr-ast.hh:10: error: 'ATmakeInt' was not declared in this scope
│   │     ├─nixexpr-ast.hh:10: error: 'ATmakeAppl3' was not declared in this scope
│   │     ├─nixexpr-ast.hh: In function 'bool nix::matchPos(_ATerm*, _ATerm*&, int&, int&)':
│   │     ├─nixexpr-ast.hh:15: error: 'ATgetType' was not declared in this scope
│   │     ├─nixexpr-ast.hh:15: error: 'AT_APPL' was not declared in this scope
│   │     ├─nixexpr-ast.hh:15: error: 'AFun' was not declared in this scope
```

Build farm goals

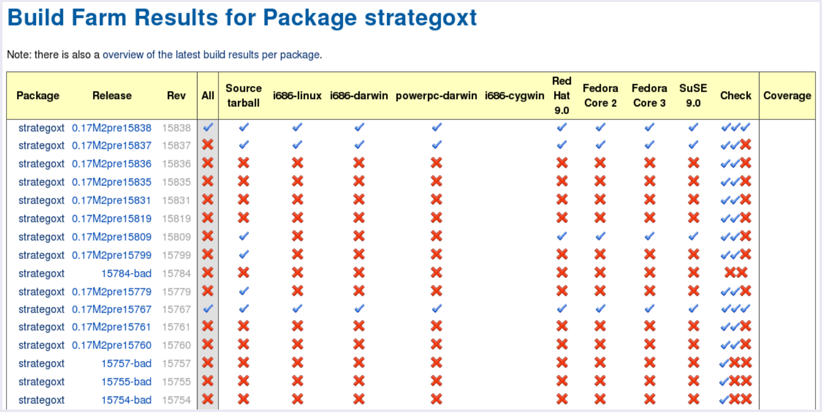
Goal: Running test suites



```
│   │   │   │   ├─List with some elements
│   │   │   │   ├─strategy failed
│   │   │   │   ├─List with element of illegal type
│   │   │   │   ├─List with element of illegal type
│   │   │   │   ├─Empty list
│   │   │   │   ├─[ lt-dfta-accept-tests | critical ] No productive start symbols
│   │   │   │   │ left in rtg
│   │   │   │   ├─RTG(Start([]),ProdRules([]))
│   │   │   │   ├─FAIL: dfta-accept-tests
│   │   │   │   ├─=======================================
│   │   │   │   ├─1 of 2 tests failed
│   │   │   │   ├─Please report to stratego-bugs@cs.uu.nl
│   │   │   │   ├─=======================================
│   │   │   │   ├─make[4]: *** [check-TESTS] Error 1
│   │   │   │   └─make[4]: Leaving directory
│   │   │   │     `/tmp/nix-24398-5/svn-export/stratego-libraries/rtg/tests'
│   │   │   ├─make[3]: *** [check-am] Error 2
```

Build farm goals

Goal: Portability testing



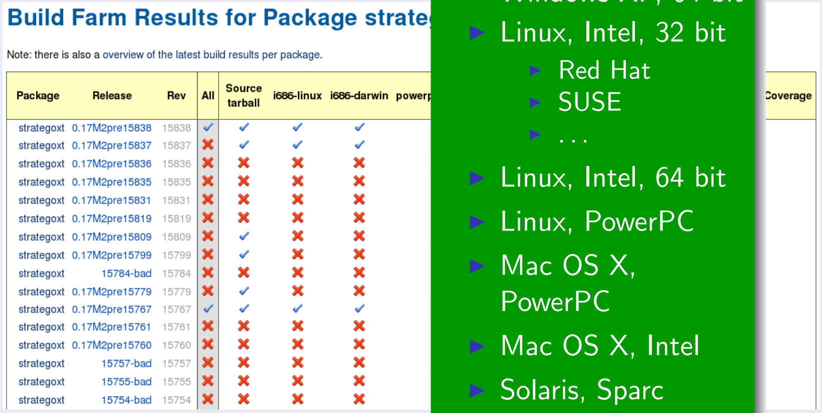
Build Farm Results for Package strategoxt

Note: there is also a overview of the latest build results per package.

Package	Release	Rev	All	Source tarball	i686-linux	i686-darwin	powerpc-darwin	i686-cygwin	Red Hat 9.0	Fedora Core 2	Fedora Core 3	SuSE 9.0	Check	Coverage
strategoxt	0.17M2pre15838	15838	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓✓✓	
strategoxt	0.17M2pre15837	15837	✗	✓	✓	✓	✓		✓	✓	✓	✓	✓✓✗	
strategoxt	0.17M2pre15836	15836	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15835	15835	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15831	15831	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15819	15819	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15809	15809	✗	✓	✗	✗	✗		✓	✓	✓	✓	✓✓✗	
strategoxt	0.17M2pre15799	15799	✗	✓	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	15784-bad	15784	✗	✗	✗	✗	✗		✗	✗	✗	✗	✗✗	
strategoxt	0.17M2pre15779	15779	✗	✓	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15767	15767	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓✓✓	
strategoxt	0.17M2pre15761	15761	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	0.17M2pre15760	15760	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✓✗	
strategoxt	15757-bad	15757	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✗✗	
strategoxt	15755-bad	15755	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✗✗	
strategoxt	15754-bad	15754	✗	✗	✗	✗	✗		✗	✗	✗	✗	✓✗✗	

Build farm goals

Goal: Portability testing



Build Farm Results for Package strate

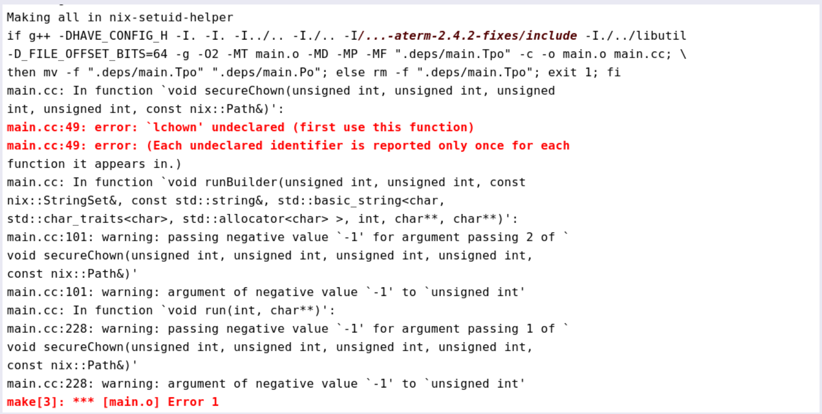
Note: there is also a overview of the latest build results per package.

Package	Release	Rev	All	Source tarball	i686-linux	i686-darwin	powerp	Coverage
strategoxt	0.17M2pre15838	15838	✓	✓	✓	✓		
strategoxt	0.17M2pre15837	15837	✗	✓	✓	✓		
strategoxt	0.17M2pre15836	15836	✗	✗	✗	✗		
strategoxt	0.17M2pre15835	15835	✗	✗	✗	✗		
strategoxt	0.17M2pre15831	15831	✗	✗	✗	✗		
strategoxt	0.17M2pre15819	15819	✗	✗	✗	✗		
strategoxt	0.17M2pre15809	15809	✗	✓	✗	✗		
strategoxt	0.17M2pre15799	15799	✗	✓	✗	✗		
strategoxt	15784-bad	15784	✗	✗	✗	✗		
strategoxt	0.17M2pre15779	15779	✗	✓	✗	✗		
strategoxt	0.17M2pre15767	15767	✓	✓	✓	✓		
strategoxt	0.17M2pre15761	15761	✗	✗	✗	✗		
strategoxt	0.17M2pre15760	15760	✗	✗	✗	✗		
strategoxt	15757-bad	15757	✗	✗	✗	✗		
strategoxt	15755-bad	15755	✗	✗	✗	✗		
strategoxt	15754-bad	15754	✗	✗	✗	✗		

- Windows XP, 32 bit
- Windows XP, 64 bit
- Linux, Intel, 32 bit
  - Red Hat
  - SUSE
  - ...
- Linux, Intel, 64 bit
- Linux, PowerPC
- Mac OS X, PowerPC
- Mac OS X, Intel
- Solaris, Sparc
- ...

Build farm goals

Goal: Portability testing

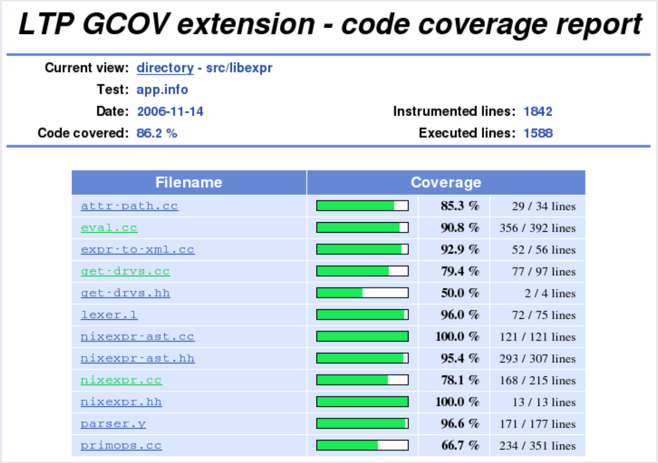


```
Making all in nix-setuid-helper
if g++ -DHAVE_CONFIG_H -I. -I. -I../.. -I./.. -I/...-aterm-2.4.2-fixes/include -I./../libutil
-D_FILE_OFFSET_BITS=64 -g -O2 -MT main.o -MD -MP -MF ".deps/main.Tpo" -c -o main.o main.cc; \
then mv -f ".deps/main.Tpo" ".deps/main.Po"; else rm -f ".deps/main.Tpo"; exit 1; fi
main.cc: In function `void secureChown(unsigned int, unsigned int, unsigned
int, unsigned int, const nix::Path&)':
main.cc:49: error: `lchown' undeclared (first use this function)
main.cc:49: error: (Each undeclared identifier is reported only once for each
function it appears in.)
main.cc: In function `void runBuilder(unsigned int, unsigned int, const
nix::StringSet&, const std::string&, std::basic_string<char,
std::char_traits<char>, std::allocator<char> >, int, char**, char**)':
main.cc:101: warning: passing negative value `-1' for argument passing 2 of `
void secureChown(unsigned int, unsigned int, unsigned int, unsigned int,
const nix::Path&)'
main.cc:101: warning: argument of negative value `-1' to `unsigned int'
main.cc: In function `void run(int, char**)':
main.cc:228: warning: passing negative value `-1' for argument passing 1 of `
void secureChown(unsigned int, unsigned int, unsigned int, unsigned int,
const nix::Path&)'
main.cc:228: warning: argument of negative value `-1' to `unsigned int'
make[3]: *** [main.o] Error 1
```

Build farm goals

Goal: Run analysis tools

Static analyses (e.g., Lint, FindBugs) or dynamic analyses (e.g., code coverage, Valgrind).



LTP GCOV extension - code coverage report

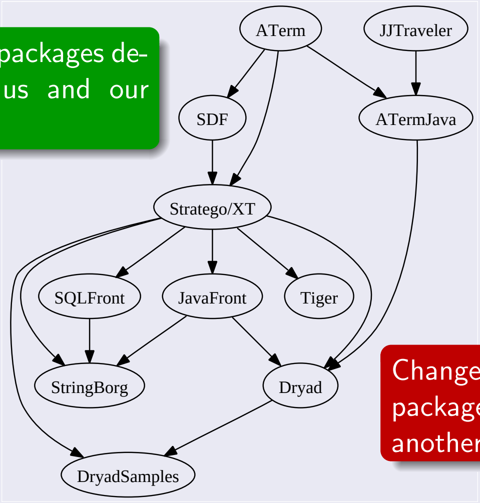
Current view:	directory - src/libexpr		
Test:	app.info		
Date:	2006-11-14	Instrumented lines:	1842
Code covered:	86.2 %	Executed lines:	1588

Filename	Coverage		
attr-path.cc		85.3 %	29 / 34 lines
eval.cc		90.8 %	356 / 392 lines
expr-to-xml.cc		92.9 %	52 / 56 lines
get-drvs.cc		79.4 %	77 / 97 lines
get-drvs.hh		50.0 %	2 / 4 lines
lexer.l		96.0 %	72 / 75 lines
nixexpr-ast.cc		100.0 %	121 / 121 lines
nixexpr-ast.hh		95.4 %	293 / 307 lines
nixexpr.cc		78.1 %	168 / 215 lines
nixexpr.hh		100.0 %	13 / 13 lines
parser.y		96.6 %	171 / 177 lines
primops.cc		66.7 %	234 / 351 lines

Build farm goals

Goal: Continuous integration

Some of the packages developed by us and our colleagues

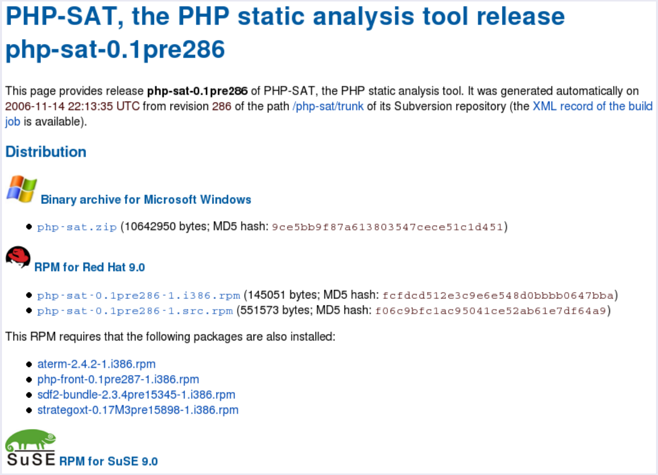


Changes in one package can break another

Build farm goals

Goal: Release management

If a build succeeds, the result can be made available as an installable package to users.



PHP-SAT, the PHP static analysis tool release php-sat-0.1pre286

This page provides release **php-sat-0.1pre286** of PHP-SAT, the PHP static analysis tool. It was generated automatically on 2006-11-14 22:13:35 UTC from revision 286 of the path /php-sat/trunk of its Subversion repository (the XML record of the build job is available).

Distribution

Binary archive for Microsoft Windows

- `php-sat.zip` (10642950 bytes; MD5 hash: `9ce5bb9f87a613803547cece51c1d451`)

RPM for Red Hat 9.0

- `php-sat-0.1pre286-1.i386.rpm` (145051 bytes; MD5 hash: `fcfdcd512e3c9e6e548d0bbbb0647bba`)
- `php-sat-0.1pre286-1.src.rpm` (551573 bytes; MD5 hash: `f06c9bfc1ac95041ce52ab61e7df64a9`)

This RPM requires that the following packages are also installed:

- aterm-2.4.2-1.i386.rpm
- php-front-0.1pre287-1.i386.rpm
- sdf2-bundle-2.3.4pre15345-1.i386.rpm
- strategoxt-0.17M3pre15898-1.i386.rpm

SuSE RPM for SuSE 9.0

Current build farm tools

Examples

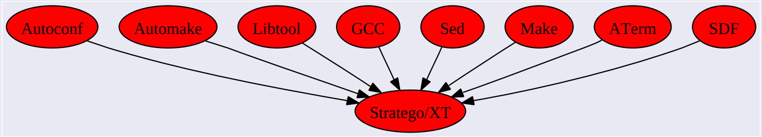
- Mozilla Tinderbox
- CruiseControl
- AntHill
- BuildBot
- SourceForge Compile Farm

Central Problem

How do we manage the build environment?

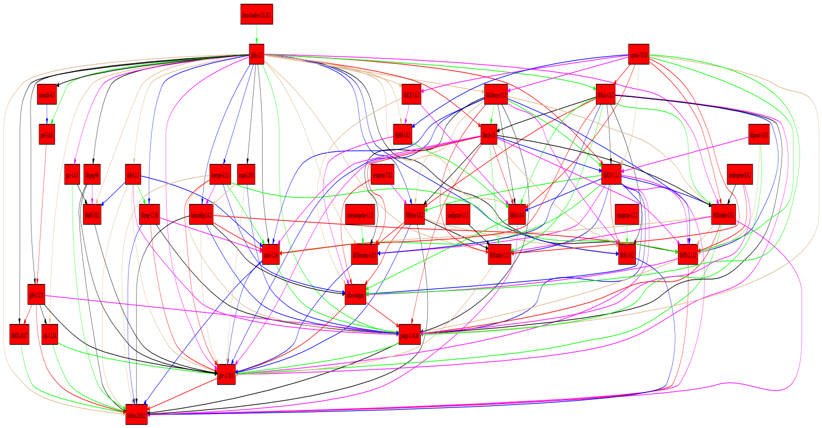
Problem: creating the build environment

- A package typically has a lot of build time dependencies that must be distributed to each build machine
- Dependencies of Stratego/XT that have caused problems in the past:



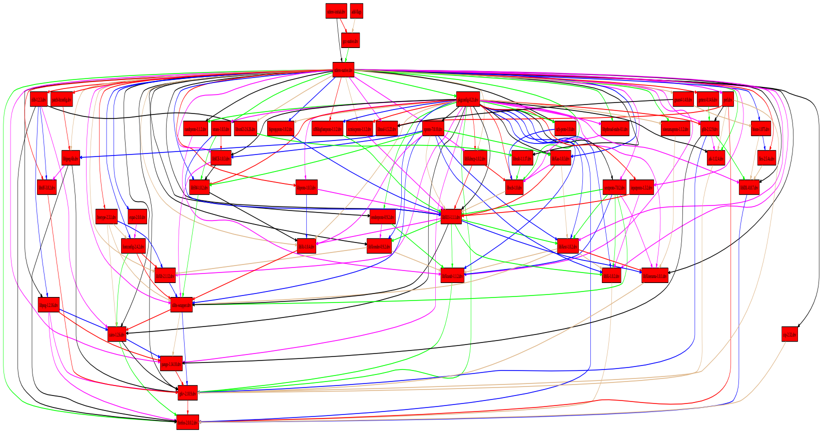
- N dependencies, M platforms
  $\Rightarrow \Theta(N \times M)$ effort to keep the build farm up to date
- And what if there are conflicting dependencies?

Runtime dependencies



Firefox

Build-time dependencies



Firefox

The Nix Deployment System

- Deployment system developed at Utrecht University: `http://nix.cs.uu.nl/`
- *Purely functional* package management: package builds only depend on declared inputs; never change after they have been built.
- Main features:
  - Enforce correct dependency specifications.
  - Support concurrent variants/versions.
  - Safe and automatic garbage collection of unused components.
  - Transparent source/binary deployment model.
  - Atomic upgrades/rollbacks.
  - Simple component language with variability support.
  - Mechanism, not policy; lots of different deployment policies can be defined using basic Nix mechanisms (e.g., channels).
  - Not just for software deployment but also service deployment.

The Nix Deployment System

- Deployment system developed at Utrecht University: `http://nix.cs.uu.nl/`
- *Purely functional* package management: package builds only depend on declared inputs; never change after they have been built.
- Main features:
  - Enforce correct dependency specifications.
  - Support concurrent variants/versions.
  - Safe and automatic garbage collection of unused components.
  - Transparent source/binary deployment model.
  - Atomic upgrades/rollbacks.
  - Simple component language with variability support.
  - Mechanism, not policy; lots of different deployment policies can be defined using basic Nix mechanisms (e.g., channels).
  - Not just for software deployment but also service deployment.

The Nix Deployment System

- Central idea: store all components in isolation.
- Unique paths:

```
/nix/store/jjp9pirx8b3nqs9k...-firefox
```

  which is an 160-bit **cryptographic hash** of **all** inputs used to build the component:

  - Sources
  - Libraries
  - Compilers
  - Build scripts
  - Build parameters
  - System type
  - ...

- **Prevent** undeclared **build time** dependencies.
- **Scan** for **runtime** dependencies.
- Deploy only **closures** under the **depends-on** relation.

Nix store

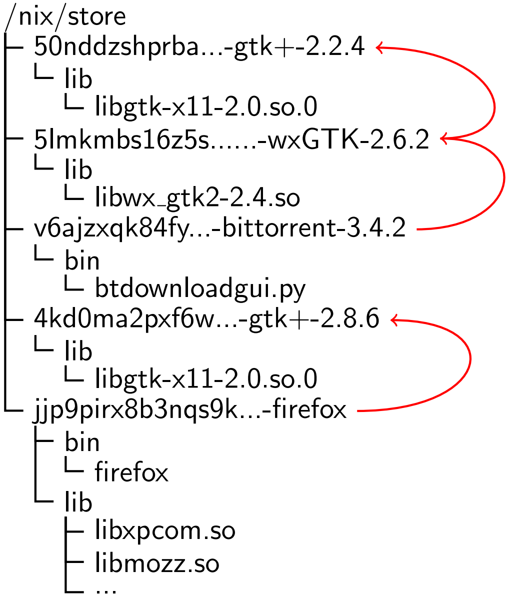
```
/nix/store
├─ 50nddzshprba...-gtk+-2.2.4
│  └─ lib
│     └─ libgtk-x11-2.0.so.0
├─ 5lmkmbs16z5s......-wxGTK-2.6.2
│  └─ lib
│     └─ libwx_gtk2-2.4.so
├─ v6ajzxqk84fy...-bittorrent-3.4.2
│  └─ bin
│     └─ btdownloadgui.py
├─ 4kd0ma2pxf6w...-gtk+-2.8.6
│  └─ lib
│     └─ libgtk-x11-2.0.so.0
└─ jjp9pirx8b3nqs9k...-firefox
   ├─ bin
   │  └─ firefox
   └─ lib
      ├─ libxpcom.so
      ├─ libmozz.so
      └─ ...
```

Nix store

```
/nix/store
├ 50nddzshprba...-gtk+-2.2.4
│ └ lib
│   └ libgtk-x11-2.0.so.0
├ 5lmkmbs16z5s......-wxGTK-2.6.2
│ └ lib
│   └ libwx_gtk2-2.4.so
├ v6ajzxqk84fy...-bittorrent-3.4.2
│ └ bin
│   └ btdownloadgui.py
├ 4kd0ma2pxf6w...-gtk+-2.8.6
│ └ lib
│   └ libgtk-x11-2.0.so.0
└ jjp9pirx8b3nqs9k...-firefox
  ├ bin
  │ └ firefox
  └ lib
    ├ libxpcom.so
    ├ libmozz.so
    └ ...
```

Unique paths for different versions

Nix store



```
/nix/store
├ 50nddzshprba...-gtk+-2.2.4
│ └ lib
│   └ libgtk-x11-2.0.so.0
├ 5lmkmbs16z5s......-wxGTK-2.6.2
│ └ lib
│   └ libwx_gtk2-2.4.so
├ v6ajzxqk84fy...-bittorrent-3.4.2
│ └ bin
│   └ btdownloadgui.py
├ 4kd0ma2pxf6w...-gtk+-2.8.6
│ └ lib
│   └ libgtk-x11-2.0.so.0
└ jjp9pirx8b3nqs9k...-firefox
  ├ bin
  │ └ firefox
  └ lib
    ├ libxpcom.so
    ├ libmozz.so
    └ ...
```

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Function arguments

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Function arguments

Build attributes

Nix expressions

hello/builder.sh

```
source $stdenv/setup

PATH=$perl/bin:$PATH

tar xvfz $src
cd hello-*
./configure --prefix=$out
make
make install
```

Nix expressions

system/all-packages.nix

```
hello = import ../applications/misc/hello/ex-1 {
  inherit fetchurl stdenv perl;
};

perl = import ../development/interpreters/perl {
  inherit fetchurl stdenv;
};

fetchurl = import ../build-support/fetchurl {
  inherit stdenv; ...
};

stdenv = ...;
```

Nix expressions

system/all-packages.nix

```
hello = import ../applications/misc/hello/ex-1 {
  inherit fetchurl stdenv perl;
};

perl = import ../development/interpreters/perl {
  inherit fetchurl stdenv;
};

fetchurl = import ../build-support/fetchurl {
  inherit stdenv; ...
};

stdenv = ...;
```

Variability

```
bittorrent = import ../tools/networking/bittorrent {
  inherit fetchurl stdenv wxGTK;
};

wxGTK = import ../development/libraries/wxGTK {
  inherit fetchurl stdenv pkgconfig;
  gtk = gtkLibs22.gtk;
};

firefox = import ../applications/browsers/firefox {
  inherit fetchurl stdenv pkgconfig perl zip libIDL libXi;
  gtk = gtkLibs24.gtk;
};
```

Implementing a Build Farm with Nix

Why is this useful for a build farm?

- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Implementing a Build Farm with Nix

Why is this useful for a build farm?

- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Implementing a Build Farm with Nix

Why is this useful for a build farm?

- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Implementing a Build Farm with Nix

Why is this useful for a build farm?

- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Implementing a Build Farm with Nix

Why is this useful for a build farm?

- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Implementing a Build Farm with Nix

Why is this useful for a build farm?

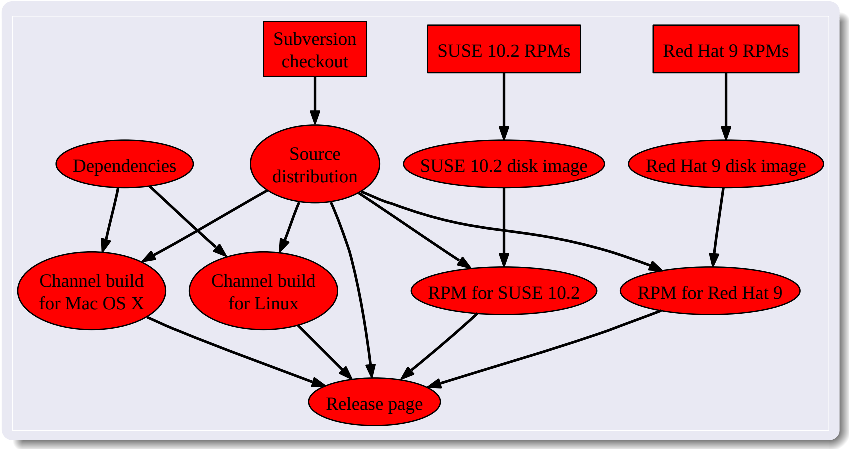
- The Nix expression language is ideal for describing the build tasks to be performed.
- The Nix expression language makes it easy to describe variant compositions.
- Nix manages the storage of components.
- Nix supports distributed builds in a transparent way.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Building a release

What goes into a release?

- A source distribution.
- Binary distributions for a number of platforms. (Test sets are also run on each platform.)
  - RPM packages for Red Hat 9, Fedora Core, SUSE Linux, ...
  - Windows binaries
  - Nix channel builds for Linux, Mac OS X, Windows, ...
  - ...
- Build logs, analysis results, etc.

Nix expressions for a release



Making Stratego/XT Releases (1)

Building a source distribution

`svnToSourceTarball` is a function that checks out sources from a specific revision from a Subversion repository (as specified by `input`.

```
# Bring in some standard packages (compilers, etc.)
pkgs = import .../all-packages.nix;
pkgsLinux = pkgs {system = "i686-linux"};

strategoxtTarball = input: svnToSourceTarball input {
  stdenv = pkgsLinux.stdenv;
  buildInputs = [pkgsLinux.autoconf pkgsLinux.automake ...];
};
```

Making Stratego/XT Releases (2)

Performing a Nix channel build for Linux

`nixBuild` performs a channel build from a source distribution.

```
strategoxtBinary = input: nixBuild
  (strategoxtTarball input)
{
  stdenv = pkgsLinux.stdenv;
  buildInputs = [pkgsLinux.aterm pkgsLinux.sdf];
};
```

Making Stratego/XT Releases (3)

Building an RPM

`umlBuild` performs an RPM build from a source distribution in a User-Mode Linux virtual machine.

```
strategoxtRPM = input: diskImage: umlBuild diskImage
  (strategoxtTarball input);

redhatDiskImage = fillWithRPMs {
  fetchurl {url = ftp://.../RedHat/basesystem-8.0-2.rpm;}
  fetchurl {url = ftp://.../RedHat/bash-2.05b-20.i386.rpm;}
  fetchurl {url = ftp://.../RedHat/gcc-3.2.2-5.i386.rpm;}
  ...
};
suseDiskImage = fillWithRPMs { ... };
```

That is, we *generate virtual machines* on the fly from a specification.

Making Stratego/XT Releases (4)

Building a release page

makeReleasePage creates an HTML release page and other files that should be uploaded to a server.

```
strategoxtRelease = input: makeReleasePage {
  stdenv = pkgsLinux.stdenv;
  sourceTarball = strategoxtTarball input;
  binaries = [(strategoxtBinary input)];
  rpms = [
    (strategoxtRPM input suseDiskImage)
    (strategoxtRPM input redhatDiskImage)
  ];
};
```

Making Stratego/XT Releases (5)

Building for Multiple Platforms

```
pkgs = (import .../all-packages);
pkgsLinux = pkgs {system = "i686-linux";};
pkgsDarwin = pkgs {system = "powerpc-darwin";};

strategoxtBinary = pkgs: input: nixBuild
  (strategoxtTarball input)
{
  stdenv = pkgs.stdenv;
  buildInputs = [pkgs.aterm pkgs.sdf];
};

strategoxtBinaries = input: [
  (strategoxtBinary pkgsLinux input)
  (strategoxtBinary pkgsDarwin input)
];
```

Conclusion

The Nix build farm:

- Manages the complexity of the build environment.
- Has a functional component language that makes it easy to specify the configurations to build/test.
- Ensures reproducibility.
- Supports multi-platform builds.
- Is efficient: only changed subexpressions are rebuilt.
- Produces actual releases.

More information

`http://nix.cs.uu.nl/`

Example Nix build farms:

- `http://nix.cs.uu.nl/dist/`
- `http://buildfarm.st.ewi.tudelft.nl/`