Nix

Язык, пакетный менеджер и экосистема

Александр Бантьев @balsoft, typeable.io

Раздел 1

Вступление

Вступление

Основано на https://nixos.org/nixos/nix-pills . Огромная благодарность @Lethalman и @grahamc за написание и поддержку этого материала!

Я хотел бы, чтобы к концу презентации всем стало понятно

- Что такое Nix
- Как собрать пакеты из `contractor` с помощью nix
- Что находится в `contractor/nix/pkgs`
- Что находится в `contractor/nix/overlay.nix`
- Как писать несложные выражения на nix, не совершая неочевидных ошибок

Вы можете прекратить слушать после любой секции, и я надеюсь, что вы унесете какую-то полезную для себя информацию.

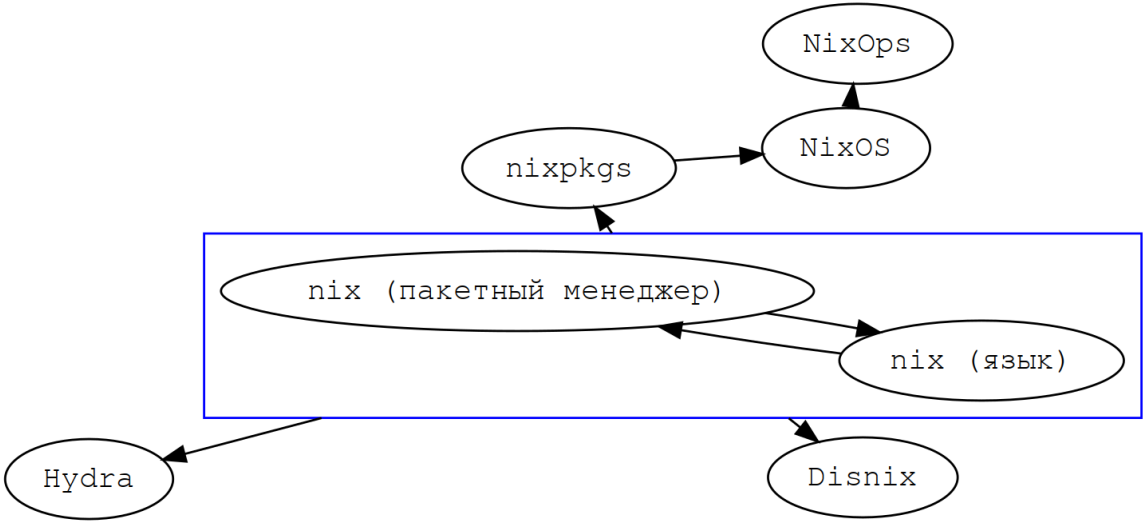
Исходники, собранный pdf и заметки докладчика для этой презентации находятся на https://github.com/typeable/nix-pills-ru

Что такое nix?

Nix – это

1. Язык
   - Декларативный
   - Функциональный
   - Чистый
   - Ленивый
   - Динамически типизированный
2. Пакетный менеджер
   - Атомарный
   - Повторяемый
   - Source/Binary

Экосистема


NixOps
NixOS
nixpkgs
nix (пакетный менеджер)
nix (язык)
Hydra
Disnix

Раздел 2

Nix как пакетный менеджер

Nix как пакетный менеджер

> *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.*

— *nixos.org/nix*

Хранилище (nix store)

> *In the beginning was the Store, and the Store was with Nix, and the Store was Nix.*
>
> — *Nix manual, 1:1*

```
$ ls /nix/store | wc -l
46251
$ nix add-to-store nix.conf
/nix/store/za34q8y0jcx3qsrnbrd005mh8zcnlrlr-nix.conf
```

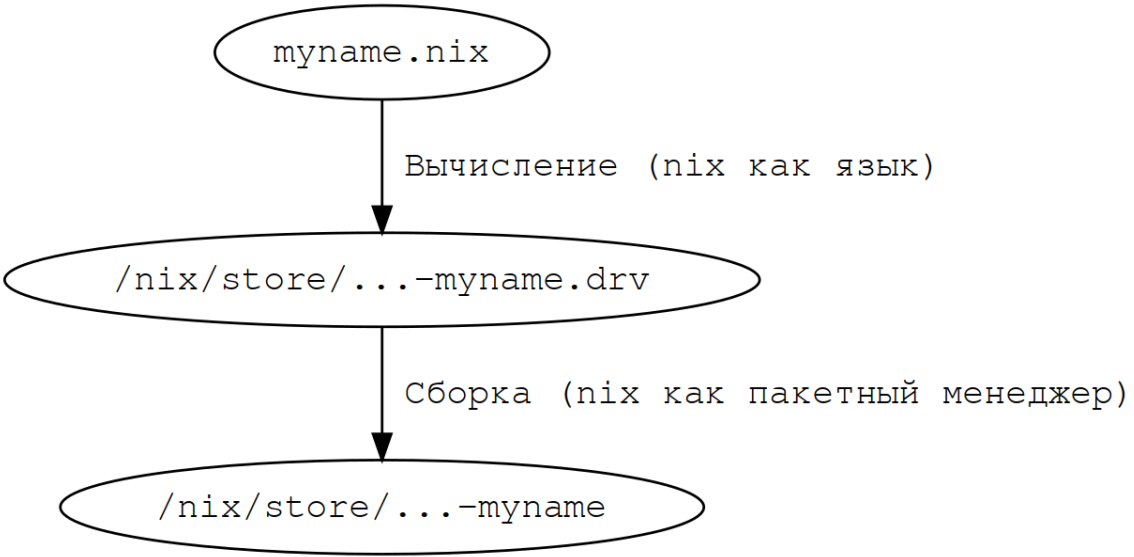
Формат имени

```
<hash>-<name>[-version][-output]
```

Например,

```
/nix/store/04mm93j20jlnwvhh21db674b3vy6fs54-pandoc-2.7.1-data
          |                 hash                 | name | ver | out |
```

Принцип работы



Деривации (derivations, `.drv` files)

`/nix/store/z3hhlxbckx4g3n9sw91nnvlkjvyw754p-myname.drv`

```
Derive
([("out"
  ,"/nix/store/40s0qmrfb45vlh6610rk29ym318dswdr-myname","","")]
,[]
,[]
,"mysystem"
,"mybuilder"
,[]
,[("builder","mybuilder")
 ,("name","myname")
 ,("out","/nix/store/40s0qmrfb45vlh6610rk29ym318dswdr-myname")
 ,("system","mysystem") ] )
```

Краткая справка по инструментарию

Собрать (build/realize) деривацию

```
$ nix-store -r /nix/store/z3hhlxbckx4g3n9sw91nnvlkjvyw754p-myname.drv
```

Вычислить (evaluate) и затем собрать содержимое файла `default.nix`

```
$ nix build -f default.nix # или nix-build default.nix
```

Вычислить, собрать и запустить интерактивное окружение

```
$ nix run -f default.nix # или nix-shell default.nix
```

Раздел 3

Nix как язык

Nix как язык

Рассмотрим основы синтаксиса Nix.

Краткая инструкция для желающих повторить:

```
$ curl https://nixos.org/nix/install | sh
<...>
$ nix repl
```

Числа, арифметика и комментарии

```
nix-repl> :t 4 # Показать тип выражения
an integer
nix-repl> 1 + 2 - 3 * 2
-3
nix-repl> 1 / 2
0
nix-repl> builtins.add 1 2
3
nix-repl> # Однострочный комментарий
nix-repl> /* Многострочный комментарий */
```

Замечание

Арифметические операторы лучше обносить пробелами! У символов / и – есть особые значения в Nix

```
nix-repl> 6/2
/home/balsoft/6/2
```

Булевы, булева арифметика и if

```
nix-repl> :t true
a boolean
nix-repl> true && false
false
nix-repl> true || false
true
nix-repl> if true || false then 3 else 4
3
nix-repl> if false then 5 else 6
6
```

Строки (strings)

```
nix-repl> "foo"
"foo"
nix-repl> :t "foo"
a string
nix-repl> "Привет"
"Привет"
nix-repl> ''foo''
"foo"
nix-repl> ''
          foo
          bar
          ''
"foo\nbar\n"
```

Переменные и стороковая интерполяция

```
nix-repl> foo = "hello"
nix-repl> "${foo}, world!"
"hello, world!"
```

Замечание

Названия переменных могут содержать знак -

```
nix-repl> foo-bar = 10
nix-repl> foo-bar
10
```

Но не могут содержать не-ASCII!

```
nix-repl> привет
error: syntax error, unexpected $undefined, at (string):1:1
```

Как избежать интерполяции?

Довольно часто встречаются ситуации, когда мы хотим вставить в строку последовательность `${...}` (например, в bash-код).

```
nix-repl> "\${foo}"
"${foo}"

nix-repl> ''test ''${foo} test''
"test ${foo} test"
```

Пути (paths) и URL

```
nix-repl> /bin/sh
/bin/sh
nix-repl> :t /bin/sh
a path
nix-repl> ./hello # Относительно текущего .nix файла или PWD для repl
/home/balsoft/hello
nix-repl> https://typeable.io # Синтаксический сахар для строк
"https://typeable.io"
nix-repl> :t https://typeable.io
a string
```

Замечание

Пути, как и названия переменных, не могут содержать не-ASCII. Я бы рассматривал это, как баг.

```
nix-repl> /home/balsoft/Документы
error: path '/home/balsoft/' has a trailing slash
```

NIX_PATH и <nixpkgs>

```
$ NIX_PATH=home=$HOME nix repl
nix-repl> <home>
/home/balsoft
^D
$ NIX_PATH=/home nix repl
nix-repl> <balsoft>
/home/balsoft
$ nix repl
nix-repl> <nixpkgs>
/nix/store/zzjhgd7p1y879z0y9pz70vjd45yfi9iz-nixpkgs
```

Списки (lists)

Списки в Nix иммутабельны и могут содержать значения любых типов вперемешку.

```
nix-repl> [ 2 "foo" true (2+3) ]
[ 2 "foo" true 5 ]
```

Замечание

Обратите внимание: элементы списка не разделены запятыми!

```
nix-repl> [ 1 + 2 ]
error: syntax error, unexpected '+', at (string):1:5
nix-repl> [ (1 + 2) ]
[ 3 ]
nix-repl> [ import ./hello.nix {} ]
[ «primop-app» /home/balsoft/hello.nix { ... } ]
```

Множества аттрибутов (attribute sets, attrsets)

```
nix-repl> s = { foo = "bar"; a-b = "baz"; "123" = "num"; }
nix-repl> :t s
a set
nix-repl> s
{ "123" = "num"; a-b = "baz"; foo = "bar"; }
nix-repl> s.a-b
"baz"
nix-repl> s."123"
"num"
nix-repl> s.${toString (122 + 1)}
"num"
```

Множества аттрибутов

```
nix-repl> s ? "123"
true
nix-repl> s ? a-b
true
nix-repl> s ? abc
false
nix-repl> s.abc or "not found" # Note that `or` is a keyword
"not found"
nix-repl> s // { a = 10; }
{ "123" = "num"; a = 10; a-b = "baz"; foo = "bar"; }
nix-repl> rec { a = 3; b = a + 4; } # Recursive attrset
{ a = 3; b = 7; }
```

Let, with и inherit

```
nix-repl> let a = 3; b = 4; in a + b
7
nix-repl> let a = 4; b = a + 5; in b # Let definitions are recursive
9
nix-repl> let a = 3; a = 8; in a
error: attribute `a' at (string):1:12 already defined at (string):1:5
nix-repl> longName = { a = 3; b = 4; }
nix-repl> with longName; a + b
7
nix-repl> let a = 10; b = 20; in { inherit a; b = b + 1; }
{ a = 10; b = 21; }
nix-repl> { inherit (s) "123" foo; }
{ "123" = "num"; foo = "bar"; }
```

Функции (functions)

```
nix-repl> x: x*2
«lambda»
nix-repl> double = x: x * 2
nix-repl> :t double
a function
nix-repl> double 3
6
nix-repl> greet = end: name: "Hello, ${name} ${end}"
nix-repl> greet "dredozubov" "!"
"Hello, dredozubov !"
nix-repl> greetExclamaition = greet "!"
nix-repl> greetExclamaition "s9gf4ult"
"Hello, s9gf4ult !"
```

Паттерн-матчинг

```
nix-repl> greet = { name, end }: "Hello, ${name} ${end}"
nix-repl> greet { name = "ak3n"; exc = "!"; }
"Hello, ak3n !"
nix-repl> greet = { name, end ? "!" }: "Hello, ${name} ${end}"
nix-repl> greet { name = "ak3n"; }
"Hello, ak3n !"
nix-repl> greet { name = "ak3n"; end = "."; }
"Hello, ak3n ."
nix-repl> greet = { name, end ? "!", ... }: "Hello, ${name} ${end}"
nix-repl> greet { name = "ak3n"; end = "."; extra = "foo"; }
"Hello, ak3n ."
nix-repl> foo = {a, ...}@args: args // { a = a + 3; }
nix-repl> foo { a = 10; b = 11; }
{ a = 13; b = 11; }
```

Подключения файлов (import)

number.nix

```
3
```

function.nix

```
a: a + 3
```

Обратно в наш уютный repl

```
nix-repl> a = import ./number.nix
nix-repl> f = import ./function.nix
nix-repl> f a
6
```

Деривации (derivation)

```
nix-repl> d = derivation {
            name = "myname";
            builder = "mybuilder";
            system = "mysystem"; }
nix-repl> :t d
a set
nix-repl> d
«derivation /nix/store/z3hhlxbckx4g3n9sw91nnvlkjvyw754p-myname.drv»
```

Деривации: сборка

```
nix-repl> :b d
[...]
error: a `mysystem' is required to build ..., but I am a `x86_64-linux'
nix-repl> d = derivation {
            name = "myname";
            builder = "mybuilder";
            system = builtins.currentSystem; }
nix-repl> :b d
[...]
[...]: executing 'mybuilder': No such file or directory
```

Деривации: что внутри?

```
nix-repl> builtins.attrNames d
[ "all" "builder" "drvAttrs" "drvPath" "name"
 "out" "outPath" "outputName" "system" "type" ]
nix-repl> d.drvAttrs
{ builder = "mybuilder"; name = "myname"; system = "x86_64-linux"; }
nix-repl> (d == d.out)
true
nix-repl> { type = "derivation"; } # Типизация истинных джентельменов
«derivation ???»
nix-repl> toString d
"/nix/store/qaq9ir9w2a34dzkjmk3igfbibi5q59sd-myname"
nix-repl> toString { outPath = "blah"; }
"blah"
nix-repl> toString { a = 10; }
error: cannot coerce a set to a string, at (string):1:1
```

Деривации: добавляем зависимости

```
nix-repl> :l <nixpkgs> # nixpkgs -- это огромная куча деривaций!
Added 10082 variables.
nix-repl> coreutils
«derivation /nix/store/...-coreutils-8.30.drv»
nix-repl> toString coreutils
"/nix/store/...-coreutils-8.30"
nix-repl> d = derivation {
            name = "myname";
            builder = "${coreutils}/bin/true";
            system = builtins.currentSystem; }
nix-repl> :b d
builder for '...-myname.drv' failed to produce output path '...-
myname'
```

Деривации: смотрим на зависимости

```
$ nix show-derivation /nix/store/...-myname.drv
{
  "/nix/store/apvmj79y84wivsnjc2vkv2q79a37nf7l-myname.drv": {
    [...]
    "inputDrvs": {
      "...-coreutils-8.30.drv": [
        "out"
      ]
    },
    "builder": "...-coreutils-8.30/bin/true",
    [...]
  }
}
```

Деривации: рабочая деривация!

builder.sh

```
declare -xp
echo foo > $out
```

Уютный nix repl

```
nix-repl> :l <nixpkgs>
Added 10082 variables.
nix-repl> d = derivation {
            name = "foo";
            builder = "${bash}/bin/bash";
            args = [ ./builder.sh ];
            system = builtins.currentSystem; }
nix-repl> :b d
this derivation produced the following outputs:
  out -> /nix/store/82jrv90z3aj1lmzr437jpp7m9krnrrzf-foo
```

Деривации: посмотрим на env

```
$ nix log /nix/store/...-foo
[...]
declare -x HOME="/homeless-shelter"
declare -x NIX_BUILD_CORES="0"
declare -x NIX_BUILD_TOP="/build"
declare -x NIX_LOG_FD="2"
declare -x NIX_STORE="/nix/store"
declare -x PATH="/path-not-set"
declare -x PWD="/build"
declare -x TEMP="/build"
declare -x builder="/nix/store/...-bash-4.4-p23/bin/bash"
declare -x name="foo"
declare -x out="/nix/store/82jrv90z3aj1lmzr437jpp7m9krnrrzf-foo"
declare -x system="x86_64-linux"
```

Деривации: опакетим простую программу на C

simple.c

```
void main() {
  puts("Simple!");
}
```

simple_builder.sh

```
export PATH="$coreutils/bin:$gcc/bin" # Переменные из derivation
mkdir -p $out/bin # Создадим выходную директорию
gcc -o $out/bin/simple $src # Соберем нашу программу
```

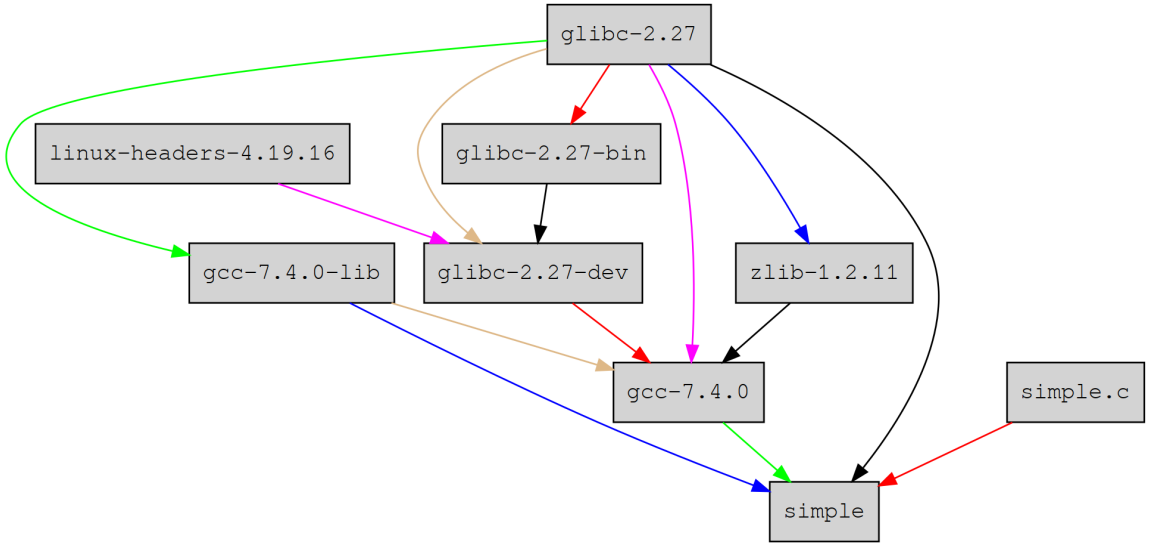
simple.nix

```
with (import <nixpkgs> {});
derivation {
  name = "simple";
  builder = "${bash}/bin/bash";
  args = [ ./simple_builder.sh ]; # Сборщик
  inherit gcc coreutils; # Зависимости
  src = ./simple.c;
  system = builtins.currentSystem;
}
```

Поехали!

```
$ nix build -f simple.nix
[1 built, 0.0 MiB DL]
$ ./result/bin/simple
Simple!
```

Посмотрим граф зависимостей (`nix-store -q result --graph`)



Раздел 4

Стандартные паттерны в Nix. nixpkgs.

mkDerivation

Неудобства `derivation`

- Очень большая часть написанного кода будет повторяться из пакета в пакет
  - Распаковка (чаще всего достаточно `tar -xf`)
  - Добавление зависимостей (`gcc`, `coreutils`, `bash`, `awk`, `make`, …)
  - Сборка (очень часто можно обойтись `./configure && make`)
  - Установка (`make install`)
- Требует как минимум два файла – builder и expression
- После вычисления уже нельзя поменять отдельные аргументы, а это может быть удобно

Создадим свой `derivation` ~~с блекджеком и шлюхами~~ без этих недостатков!

mkDerivation: зависимости и стандартный сборщик

generic-builder.sh

```
set -e
unset PATH
for p in $buildInputs $baseInputs; do
  export PATH=$p/bin${PATH:+:}$PATH
done
tar -xf $src
for d in *; do
  if [ -d "$d" ]; then
    cd "$d"
    break
  fi
done
./configure --prefix=$out
make
make install
```

generic-builder.nix

```
pkgs: attrs:
  with pkgs;
  let defaultAttrs = {
    builder = "${bash}/bin/bash";
    args = [ ./builder.sh ];
    baseInputs = [ gnutar gzip gnumake gcc binutils-unwrapped
                   coreutils gawk gnused gnugrep findutils ];
    buildInputs = [];
    system = builtins.currentSystem;
  };
  in
derivation (defaultAttrs // attrs)
```

mkDerivation: добавляем фазы

```
unset PATH
for p in $baseInputs $buildInputs; do
  export PATH=$p/bin${PATH:+:}$PATH
done

function unpackPhase() {
  tar -xzf $src

  for d in *; do
    if [ -d "$d" ]; then
      cd "$d"
      break
    fi
  done
}
```

```
function configurePhase() {
  ./configure --prefix=$out
}
function buildPhase() {
  make
}
function installPhase() {
  make install
}
function genericBuild() {
  unpackPhase
  configurePhase
  buildPhase
  installPhase
  fixupPhase
}
```

builder.sh

```
set -e
source $setup
genericBuild
```

generic-builder.nix

```
pkgs: attrs:
  with pkgs;
  let defaultAttrs = {
    builder = "${bash}/bin/bash";
    args = [ ./builder.sh ];
    setup = ./setup.sh;
    baseInputs = [ gnutar gzip gnumake gcc binutils-unwrapped
                   coreutils gawk gnused gnugrep  findutils ];
    buildInputs = [];
    system = builtins.currentSystem;
  };
  in
derivation (defaultAttrs // attrs)
```

mkDerivation: воспользуемся нашей функцией!

default.nix

```
let
  pkgs = import <nixpkgs> {};
  mkDerivation = import ./generic-builder.nix pkgs;
in
mkDerivation {
  name = "hello";
  src = ./hello-2.10.tar.gz;
};
```

Пакеты как функции от зависимостей: паттерн inputs

hello.nix

```
{ mkDerivation }:
mkDerivation {
  name = "hello";
  src = ./hello-2.10.tar.gz;
}
```

default.nix

```
let
  pkgs = import <nixpkgs> {};
  mkDerivation = import ./generic-builder.nix pkgs;
in
{
  hello = import ./hello.nix { inherit mkDerivation };
}
```

Сборка с библиотеками: NIX_CFLAGS_COMPILE, NIX_LDFLAGS

generic-builder.sh

```
[...]
for p in $baseInputs $buildInputs; do
  [...]
  if [ -d $p/include ]; then
    export NIX_CFLAGS_COMPILE="-I $p/include${NIX_CFLAGS_COMPILE:+ }\
$NIX_CFLAGS_COMPILE"
  fi
  if [ -d $p/lib ]; then
    export NIX_LDFLAGS="-rpath $p/lib -L \
$p/lib${NIX_LDFLAGS:+ }$NIX_LDFLAGS"
  fi
done
[...]
```

graphviz.nix

```
{ mkDerivation, gdSupport ? true, gd, fontconfig, libjpeg, bzip2 }:

mkDerivation {
  name = "graphviz";
  src = ./graphviz-2.38.0.tar.gz;
  buildInputs = if gdSupport
  then [ gd fontconfig libjpeg bzip2 ]
  else [];
}
```

default.nix

```
let
  pkgs = import <nixpkgs> {};
  mkDerivation = import ./generic-builder.nix pkgs;
in
{
  hello = import ./hello.nix { inherit mkDerivation };
  graphviz = import ./graphviz.nix { inherit mkDerivation
      gd fontconfig libjpeg bzip2; };
  graphvizCore = import ./graphviz.nix {
    inherit mkDerivation gd fontconfig libjpeg bzip2;
    gdSupport = false;
  };
}
```

Паттерн callPackage

default.nix

```
let
  pkgs = import <nixpkgs> {};
  mkDerivation = import ./generic-builder.nix pkgs;
in
{
  hello = callPackage ./hello.nix { };
  graphviz = callPackage ./graphviz.nix { };
  graphvizCore = callPackage ./graphviz.nix { gdSupport = false; };
}
```

```
nix-repl> add = { a ? 3, b }: a+b
nix-repl> builtins.functionArgs add
{ a = true; b = false; }

nix-repl> values = { a = 3; b = 5; c = 10; }
nix-repl> builtins.intersectAttrs values (builtins.functionArgs add)
{ a = true; b = false; }
nix-repl> builtins.intersectAttrs (builtins.functionArgs add) values
{ a = 3; b = 5; }
```

Простой вариант `callPackage`

```
nix-repl> callPackage = set: f:
          f (builtins.intersectAttrs
              (builtins.functionArgs f)
              set)
nix-repl> callPackage values add
8
nix-repl> with values; add { inherit a b; }
8
```

callPackage: добавляем оверрайды

```
nix-repl> callPackage = set: f: overrides:
            f ((builtins.intersectAttrs
                  (builtins.functionArgs f)
                  set)
                // overrides)
nix-repl> callPackage values add { }
8
nix-repl> callPackage values add { b = 12; }
15
```

callPackage: используем!

```
let
  nixpkgs = import <nixpkgs> {};
  allPkgs = nixpkgs // pkgs;
  callPackage = path: overrides:
    let f = import path;
    in f ((builtins.intersectAttrs
             (builtins.functionArgs f)
             allPkgs)
           // overrides);
  pkgs = with nixpkgs; {
    mkDerivation = import ./generic-builder.nix nixpkgs;
    hello = callPackage ./hello.nix { };
    graphviz = callPackage ./graphviz.nix { };
    graphvizCore = callPackage ./graphviz.nix { gdSupport = false; };
  };
in pkgs
```

Паттерн override

```
mygraphviz = callPackage ./graphviz.nix { gd = customgd; }
```

VS.

```
mygraphviz = pkgs.graphviz.override { gd = customgd }
```

Паттерн override: makeOverridable

lib.nix

```
rec {
  makeOverridable = f: origArgs:
    let
      origRes = f origArgs;
    in
      origRes // {
        override = newArgs:
          makeOverridable f (origArgs // newArgs);
      };
}
```

Паттерн `override`: потестируем!

```
nix-repl> :l lib.nix
Added 1 variables.
nix-repl> f = { a, b }: { result = a+b; }
nix-repl> res = makeOverridable f { a = 3; b = 5; }
nix-repl> res2 = res.override { a = 10; }
nix-repl> res2
{ override = «lambda»; result = 15; }
nix-repl> res2.override { b = 20; }
{ override = «lambda»; result = 30; }
```

Паттерн override: объединяем с callPackage

lib.nix

```
rec {
  [...]
  callPackage = pkgs: path: overrides:
    let f = import path;
    in makeOverridable
      f ((builtins.intersectAttrs
            (builtins.functionArgs f)
            pkgs)
          // overrides);
}
```

Паттерн override: репозиторий

default.nix

```
let
  lib = import ./lib.nix;
  nixpkgs = import <nixpkgs> {};
  callPackage = lib.callPackage allPkgs;
  allPkgs = nixpkgs // pkgs;
  pkgs = with nixpkgs; rec {
    mkDerivation = import ./generic-builder.nix nixpkgs;
    hello = callPackage ./hello.nix { };
    graphviz = callPackage ./graphviz.nix { };
    graphvizCore = graphviz.override { gdSupport = false };
  };
in pkgs
```

nixpkgs: краткая справка

Общая конфигурация для всех пакетов

- `import <nixpkgs> { config = {...}; }`
- `$NIXPKGS_CONFIG`
- `~/.config/nixpkgs/config.nix`

Перегрузка пакетов

```
config.packageOverrides = oldPkgs: {
  graphviz = oldPkgs.graphviz.override { gd = customgd; };
}
```

Оверлеи

```
nixpkgs.overlays = [ (self: super: {
  graphviz = super.graphviz.override { gd = customgd; };
} ) ];
```

Что осталось за кадром в этом разделе

- `stdenv`
- `lib`
- `fix`

https://nixos.org/nixos/nix-pills и https://nixos.org/nixpkgs/manual – для тех, кому надо или интересно!