Software Build Systems and Dependencies

Cooperating and Competing with Distributions

Dept: Intelligent Cloud Technologies Lab, Huawei Munich Research Center
Author: Klaus T. Aehlig
Date: Fall 2023

HUAWEI

Background: `just`

- generic build system
  - high-level build description, provided by rules
  - remote execution
  - separation of physical and logical paths (“staging”)
  - multi-repository builds based on local names; target-level caching

Background: `just`

- generic build system
  - high-level build description, provided by rules
  - remote execution
  - separation of physical and logical paths (“staging”)
  - multi-repository builds based on local names; target-level caching
- open source
  - open since Nov 2022; Relase 1.0.0 Dec 12, 2022; active development *(1.1.0 May 19, 2023, 1.2.0 Aug 25, 2023, …)*
  - Apache 2.0 license
  - `https://github.com/just-buildsystem/justbuild`
  - Packed in AUR, Nixpkgs, Spack, Void Linux
    pending: Debian

Linux Distributions

Of course, there are differences, but generally …

Linux Distributions

Of course, there are differences, but generally …

- Support stable releases/branches/…

Linux Distributions

Of course, there are differences, but generally …

- Support stable releases/branches/…
- This at least includes security fixes—handled by a security team
  - get early access to vulnerability reports ⇝ need to establish trust
    (handle them handle properly, without premature disclosure)
  - deliberately small team, also encourge/accept that only one member be contacted

Linux Distributions

Of course, there are differences, but generally …

- Support stable releases/branches/…
- This at least includes security fixes—handled by a security team
  - get early access to vulnerability reports ⇝ need to establish trust
    (handle them handle properly, without premature disclosure)
  - deliberately small team, also encourge/accept that only one member be contacted

  ⇝ effort for a single report must be managable
  (what can be automated, like rebuild everything depending on this, is not a problem)

Linux Distributions

Of course, there are differences, but generally …

- Support stable releases/branches/…
- This at least includes security fixes—handled by a security team
  - get early access to vulnerability reports ⇝ need to establish trust
    (handle them handle properly, without premature disclosure)
  - deliberately small team, also encourge/accept that only one member be contacted

  ⇝ effort for a single report must be managable
    (what can be automated, like rebuild everything depending on this, is not a problem)
- Each upstream archive/tree packaged only at a single place—no embedded copies!
- Build offline! No fetches of dependencies during the build.
  (Also for complience reasons!)

Linux Distributions

Of course, there are differences, but generally …

- Support stable releases/branches/…
- This at least includes security fixes—handled by a security team
  - get early access to vulnerability reports ⇝ need to establish trust
    (handle them handle properly, without premature disclosure)
  - deliberately small team, also encourge/accept that only one member be contacted

  ⇝ effort for a single report must be managable
    (what can be automated, like rebuild everything depending on this, is not a problem)
- Each upstream archive/tree packaged only at a single place—no embedded copies!
- Build offline! No fetches of dependencies during the build.
  (Also for complience reasons!)
- In package build, dependencies passed as inputs
  …possibly explicit, but often in form of the “ambient environment”
  (which might well be a controlled `chroot` *in which the build happens!)*

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”
- Dependencies should be updated regularly—as part of the history
  - reconstruct old versions and work with branches (production, staging, head, …)
  - bisect over dependency updates—an update might be the cause of a breakage

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”
- Dependencies should be updated regularly—as part of the history
  - reconstruct old versions and work with branches (production, staging, head, …)
  - bisect over dependency updates—an update might be the cause of a breakage
- This also include first-party dependencies.

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”
- Dependencies should be updated regularly—as part of the history
  - reconstruct old versions and work with branches (production, staging, head, …)
  - bisect over dependency updates—an update might be the cause of a breakage
- This also include first-party dependencies.
- It should just work, also for cooperation partners.
  (and they should get the same binary out)

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”
- Dependencies should be updated regularly—as part of the history
  - reconstruct old versions and work with branches (production, staging, head, …)
  - bisect over dependency updates—an update might be the cause of a breakage
- This also include first-party dependencies.
- It should just work, also for cooperation partners.
  (and they should get the same binary out)
- Ex-post proof that a binary was built from certain sources. (specific use case)

TLM and Software Development in Industry

Your milage may vary, but the following are not unheard of.

- SWEs should not waste time installing dependencies
- Everyone to use the same dependencies, no “works on this machine”
- Dependencies should be updated regularly—as part of the history
  - reconstruct old versions and work with branches (production, staging, head, …)
  - bisect over dependency updates—an update might be the cause of a breakage
- This also include first-party dependencies.
- It should just work, also for cooperation partners.
  (and they should get the same binary out)
- Ex-post proof that a binary was built from certain sources. (specific use case)
- “So, why doesn’t the build tool just download the dependencies?”
  (We have known-good hashes to verify the downloads, so all is fine.)

Existing Approaches

- “I’m just a build system”
  - Just traverses a graph in topological order
  - example: `make`

Existing Approaches

- “I’m just a build system”
  - Just traverses a graph in topological order
  - example: `make`
- “Let me inspect the environment for you”
  - search the environment for the required dependencies
    (trying all “standard paths”, “standard names”, heuristics . . .)
  - interpolate the found locations into the build description
  - example: `autotools`

Existing Approaches

- "I'm just a build system"
  - Just traverses a graph in topological order
  - example: `make`
- "Let me inspect the environment for you"
  - search the environment for the required dependencies
    (trying all "standard paths", "standard names", heuristics ...)
  - interpolate the found locations into the build description
  - example: `autotools`

That's basically the world of traditional Linux distributions.

Existing Approaches (cont'd)

- "Trust us, we're the experts"
  - Host pre-built JDKs for all OS/architectures
  - embedd URLs and hashes into the build tool
    (You're updating your build tool regularly, aren't you?)
  - download as needed
  - example: `bazel`

Existing Approaches (cont'd)

- "Trust us, we're the experts"
  - Host pre-built JDKs for all OS/architectures
  - embedd URLs and hashes into the build tool
    (You're updating your build tool regularly, aren't you?)
  - download as needed
  - example: `bazel`
- "Let me download the right `bazel` version for you"
  - With frequent incompatible changes, the buildtool becomes itself a dependency
  - Have a wrapper, that inspects `.bazelversion`, downloads the needed version of `bazel` if not present, and run it
  - example: `bazelisk`

Existing Approaches (cont'd again)

- "You'll only need this programming language anyway"
  - tailor towards one language
  - keep exhaustive collection of packages for that language *(and handle dependency resolution, etc)*
  - encourage every one to download from there, ignoring the distribution
  - examples: `pip`, `cargo`

Existing Approaches (cont'd again)

- "You'll only need this programming language anyway"
  - tailor towards one language
  - keep exhaustive collection of packages for that language *(and handle dependency resolution, etc)*
  - encourage every one to download from there, ignoring the distribution
  - examples: `pip`, `cargo`
- "Let me manage everything for you"
  - attempt a collection for everything you might possibly need
  - encourge users of your build system to take everything from there
  - example: `Bazel Central Registry`

Flexible Repository Configuration

- Already have abstract repository configuration
  - Build descriptions only use local names for other repositories
    (association to global names in the `"bindings"` *of the repository configuration)*
  - names of target files configurable
  - roots can be taken from various places

Flexible Repository Configuration

- Already have abstract repository configuration
  - Build descriptions only use local names for other repositories
    (association to global names in the `"bindings"` *of the repository configuration)*
  - names of target files configurable
  - roots can be taken from various places

⇝ Easy to rebind a dependency, switch between alternative definitions, etc

Flexible Repository Configuration

- Already have abstract repository configuration
  - Build descriptions only use local names for other repositories
    (association to global names in the `"bindings"` *of the repository configuration)*
  - names of target files configurable
  - roots can be taken from various places

⇝ Easy to rebind a dependency, switch between alternative definitions, etc

- Also easy to do programatically, as it is simply a JSON file

Support pkg-config for Dependencies and Dependents

```
$ cat TARGETS
{ "fmt": {"type": ["@", "rules", "CC/pkgconfig", "system_library"], "name": ["fmt"]}
, "hello":
  { "type": ["@", "rules", "CC", "library"]
  , "name": ["hello"]
  , "hdrs": ["hello.hpp"]
  , "srcs": ["hello.cpp"]
  , "deps": ["fmt"]
  }
, "": {"type": ["@", "rules", "CC", "install-with-deps"], "targets": ["hello"]}
}
$
```

Support pkg-config for Dependencies and Dependents

```
$ just-mr analyse fmt --dump-actions actions-fmt.json
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","analyse","-C","/worker/build/62d9cc5ae42928f8/...
INFO: Requested target is [["@","","","fmt"],{}]
INFO: Result of target [["@","","","fmt"],{}]: {
        "artifacts": {
        },
        "provides": {
          "compile-args": [
            "@fmt.cflags"
          ],
          "compile-deps": {
          },
          "link-args": [
            "@fmt.ldflags"
          ],
          "link-deps": {
          },
          "package": {
            "cflags-files": {"fmt.cflags":{"data":{"id":"c3291488ce224adfd7363c5a0...
            "ldflags-files": {"fmt.ldflags":{"data":{"id":"d49ced3c1f6735822eb14bb...
          }
        },
        "runfiles": {
          }
        }
INFO: Dumping actions for target [["@","","","fmt"],{}]
$
```

Support pkg-config for Dependencies and Dependents

```
$ cat actions-fmt.json
[
  {
    "command": ["/bin/sh","-c","pkg-config '--cflags' 'fmt' > 'fmt.cflags'"],
    "env": {
      "PATH": "/bin:/usr/bin"
    },
    "output": ["fmt.cflags"]
  },
  {
    "command": ["/bin/sh","-c","pkg-config '--libs' 'fmt' > ldflags.raw"],
    "env": {
      "PATH": "/bin:/usr/bin"
    },
    "output": ["ldflags.raw"]
  },
  {
    "command": ["/bin/sh","-c","./add_rpath $(cat ldflags.raw) > 'fmt.ldflags'"],
    "input": {
      "add_rpath": {
        "data": {
          "path": "CC/pkgconfig/add_rpath",
          "repository": "rules"
        },
        "type": "LOCAL"
      },
      "ldflags.raw": {
        "data": {
          "id": "68c1d7dafa4e91467154ac91ed5252943af483
          "path": "ldflags.raw"
        },
        "type": "ACTION"
      }
    },
    "output": ["fmt.ldflags"]
  }
]
$
```

Support pkg-config for Dependencies and Dependents

```
$ just-mr analyse hello
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","analyse","-C","/worker/build/62d9cc5ae42928f8/...
INFO: Requested target is [["@","","","hello"],{}]
INFO: Result of target [["@","","","hello"],{}]: {
        "artifacts": {
          "libhello.a": {"data":{"id":"8d5b08509547e6eae6c21cfa46cf4ec2da045321","...
        },
        "provides": {
          "compile-args": [
            "@fmt.cflags"
          ],
          "compile-deps": {
          },
          "link-args": [
            "libhello.a",
            "@fmt.ldflags"
          ],
          "link-deps": {
          },
          "package": {
            "cflags-files": {"fmt.cflags":{"data":{"id":"c3291488ce224adfd7363c5a0...
            "ldflags-files": {"fmt.ldflags":{"data":{"id":"d49ced3c1f6735822eb14bb...
            "name": "hello"
          },
          "run-libs": {
          },
          "run-libs-args": [
          ]
        },
        "runfiles": {
          "hello.hpp": {"data":{"path":"hello.hpp","rep
        }
      }
$
```

Support pkg-config for Dependencies and Dependents

```
$ just-mr build -P lib/pkgconfig/hello.pc
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","build","-C","/worker/build/62d9cc5ae42928f8/root/home/.cache/just/protocol-dependent/generation-0/git-sha1
INFO: Requested target is [["@","","",""],{}]
INFO: Analysed target [["@","","",""],{}]
INFO: Discovered 5 actions, 1 trees, 1 blobs
INFO: Building [["@","","",""],{}].
INFO: Processed 5 actions, 0 cache hits.
INFO: Artifacts built, logical paths are:
        include/hello.hpp [358d39118af999403eb19bc97647340e08c64725:119:f]
        lib/libhello.a [39be270eeff3c3a52871a120bc4318c100802a4a:5982:f]
        lib/pkgconfig/fmt.cflags [8b137891791fe96927ad78e64b0aad7bded08bdc:1:f]
        lib/pkgconfig/fmt.ldflags [734287fcf96cc358652d4c91c277824a311de558:7:f]
        lib/pkgconfig/hello.pc [fb57462962400668bd67ba9bbaf36a3b81d5df2b:283:f]
prefix=/
libdir=${prefix}/lib
includedir=${prefix}/include
Name: hello
Version: unknown
Description: Pkg-config for hello, generated by JustBuild
URL: unknown
Cflags: -I${includedir} @${prefix}/lib/pkgconfig/fmt.cflags
Libs: ${libdir}/libhello.a @${prefix}/lib/pkgconfig/fmt.ldflags
$
```

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules

```
$ just-mr describe --main rules CC defaults
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/628797c6a...
[["@","rules","CC","defaults"],{}] is defined by user-defined rule ["@","ru...

 | A rule to provide defaults.
 | All CC targets take their defaults for CC, CXX, flags, etc from
 | the target ["CC", "defaults"]. This is probably the only sensible
 | use of this rule. As targets form a different root, the defaults
 | can be provided without changing this directory.
String fields
- "CC"
   | The C compiler to use
- "CXX"
   | The C++ compiler to use
- "CFLAGS"
   | Flags for C compilation. Specifying this field overwrites
   | values from "base".
- "CXXFLAGS"
   | Flags for C++ compilation. Specifying this field overwrites
   | values from "base".
- "LDFLAGS"
   | Linker flags for linking the final CC library. Specifying this field
   | overwrites values from "base".
- "ADD_CFLAGS"
   | Additional compilation flags for C. Specifying this field
   | extends values from "base".
- "ADD_CXXFLAGS"
   | Additional compilation flags for C++. Specifying this field
   | extends values from "base".
- "ADD_LDFLAGS"
   | Additional linker flags for linking the final CC library. Specifying
   | this field extends values from "base".
- "AR"
   | The archiver tool to use
- "PATH"
   | Path for looking up the compilers. Individual paths are joined
   | with ":". Specifying this field extends values from "base".
- "SYSTEM_TOOLS"
   | List of tools ("CC", "CXX", or "AR") that should be taken from
   | the system instead of from "toolchain" (if specified).
Target fields
- "base"
   | Other targets (using the same rule) to inherit values from.
- "toolchain"
   | Optional toolchain directory. A collection of artifacts that provide
   | the tools CC, CXX, and AR (if needed). Note that only artifacts of
   | the specified targets are considered (no runfiles etc.). Specifying
   | this field extends artifacts from "base". If the toolchain
   | supports cross-compilation, it should perform a dispatch on the
   | configuration variable "BUILD_ARCH" to determine for which
   | architecture to generate code for.
- "deps"
   | Optional CC libraries any CC library and CC binary implicitly depend
   | on. Those are typically "libstdc++" or "libc++" for C++ targets.
   | Specifying this field
Variables taken from the
- "ARCH"
- "HOST_ARCH"
- "TARGET_ARCH"
Result
- Artifacts
- Runfiles

$
```

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules

```
$ just-mr describe --main rules CC/proto defaults
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/628797c6a...
[["@","rules","CC/proto","defaults"],{}] is defined by user-defined rule ["...

 | A rule to provide protoc/GRPC defaults.
 | Used to implement ["CC/proto", "defaults"] for CC proto libraries
 | and ["CC/proto", "service defaults"] for CC proto service libraries
 | (GRPC).
String fields
- "PROTOC"
   | The proto compiler. If "toolchain" is empty, this field's value is
   | considered the proto compiler name that is looked up in "PATH". If
   | "toolchain" is non-empty, this field's value is assumed to be the
   | relative path to the proto compiler in "toolchain". Specifying this
   | field overwrites values from "base".
- "LDFLAGS"
   | Linker flags for linking the final CC library. Specifying this field
   | overwrites values from "base".
- "ADD_LDFLAGS"
   | Additional linker flags for linking the final CC library. Specifying
   | this field extends values from "base".
- "GRPC_PLUGIN"
   | The GRPC plugin for the proto compiler. If "toolchain" is empty,
   | this field's value is considered to be the absolute system path to the
   | plugin. If "toolchain" is non-empty, this field's value is assumed
   | to be the relative path to the plugin in "toolchain". Specifying
   | this field overwrites values from "base".
- "PATH"
   | Path for looking up the proto compiler. Individual paths are joined
   | with ":". Specifying this field extends values from "base".
Target fields
- "base"
   | Other targets (using the same rule) to inherit values from. If
   | multiple targets are specified, for values that are overwritten (see
   | documentation of other fields) the last specified value wins.
- "toolchain"
   | Optional toolchain directory. A collection of artifacts that provide
   | the protobuf compiler and the GRPC plugin (if needed). Note that only
   | artifacts of the specified targets are considered (no runfiles etc.).
   | Specifying this field extends artifacts from "base".
- "deps"
   | Optional CC libraries the resulting CC proto libraries implicitly
   | depend on. Those are typically "libprotobuf" for CC proto libraries
   | and "libgrpc++" for CC proto service libraries. Specifying this
   | field extends dependencies from "base".
Variables taken from the configuration
- "ARCH"
- "HOST_ARCH"
Result
- Artifacts
- Runfiles

$
```

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules

```
$ just-mr describe --main rules patch defaults
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/628797c6a...
[["@","rules","patch","defaults"],{}] is defined by user-defined rule ["@",...

 | A rule to provide defaults.
 | All targets take their defaults for PATCH from the target
 | ["patch", "defaults"]. This is probably the only sensible
 | use of this rule. As targets form a different root, the defaults
 | can be provided without changing this directory.
 String fields
 - "PATCH"
   | The patch binary to use
 - "PATH"
   | Path for looking up the compilers. Individual paths are joined
   | with ":". Specifying this field extends values from "base".
 - "SYSTEM_TOOLS"
   | List of tools ("PATCH") that should be taken from
   | the system instead of from "toolchain" (if specified).
 Target fields
 - "base"
   | Other targets (using the same rule) to inherit values from.
 - "toolchain"
   | Optional toolchain directory. A collection of artifacts that provide
   | the tool PATCH. Note that only artifacts of
   | the specified targets are considered (no runfiles etc.). Specifying
   | this field extends artifacts from "base".

Variables taken from the configuration
- "ARCH"
- "HOST_ARCH"
Result
- Artifacts
- Runfiles

$
```

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules

Tool Defaults

- Targets implicitly depend on the toolchain
  … provided by the respective "`defaults`" target of the rules
- Those defaults …
  - support inheriting from other defaults
  - specify names of the tools
  - specify path where to find them (if taken from host)
  - set flags, as well as flags to add on top of what is inherited
  - allow tools to be built by other targets

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools
- want to bundle—but writing `just` target files is effort!

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools
- want to bundle—but writing `just` target files is effort!

⇝ call the foreign tool
(one huge action, but updates kind-of rare, so shared caching saves)

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

```
$ just-mr describe --main rules --rule CC/foreign/make data
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/62c162591...
 | Data produced by Configure and Make build and install.
 |
 | All variables accessible to commands and options are: "TMPDIR",
 | "LOCALBASE", "CC", "CXX", "CFLAGS", "CXXFLAGS", "LDFLAGS",
 | "AR", and "PREFIX". "LOCALBASE" contains the path to the
 | installed artifacts from "deps".
String fields
- "subdir"
  | The subdirectory that contains the configure and Makefile. Individual
  | directory components are joined with "/".
- "configure"
  | Run ./configure if non-empty.
- "configure_options"
  | The configure options (the "--prefix" option is automatically set.
  | Variables can be accessed via "${<varname>}", e.g., "${TMPDIR}"
  | for variable "$TMPDIR".
- "targets"
  | The Make targets to build in the specified order
  | (default: ["install"]).
- "prefix"
  | The prefix used for the Make target. The path will be made absolute
  | and individual directory components are joined with "/". If no
  | prefix is specified, the value from the config variable "PREFIX" is
  | taken, with the default value being "/".
- "options"
  | Make options for the configuration phase
  | (e.g., ["-f", "Makefile", "ARCH=x86", "LD=${CC}"]). Variables
  | can be accessed via "${<varname>}", e.g., "${TMPDIR}" for
  | variable "$TMPDIR".
- "jobs"
  | Number of jobs to run simultaneously. If omitted, Make's default
  | number is used.
- "pre_cmds"
  | List of commands executed in the project directory before calling
  | Configure or Make. Useful for renaming files or directories. Note
  | that data between "pre_cmds" and "post_cmds" can be exchanged via
  | "$TMPDIR", which is uniquely reserved for this action.
- "post_cmds"
  | List of commands executed in the install directory after successful
  | installation but before the output files are collected. Useful for
  | renaming files or directories. Note that data between "pre_cmds" and
  | "post_cmds" can be exchanged via "$TMPDIR", which is uniquely
  | reserved for this action.
- "out_files"
  | Paths to the produced output files. The paths are considered relative
  | to the install directory.
  | Note that "out_files" and "out_dirs" may not overlap.
- "out_dirs"
  | Paths to the produced output directories. The paths are considered
  | relative to the install directory.
  | Note that "out_files" and "out_dirs" may not overlap.
Target fields
- "project"
  | The Make project directory. It should contain a single tree artifact
- implict dependency
  - ["@","rules","CC","defaults"]
- implict dependency
  - ["@","rules","CC/foreign","expand_exec"]
- implict dependency
  - ["@","rules","CC/foreign","defaults"]
Variables taken from the configuration
- "ARCH"
- "HOST_ARCH"
- "CC"
  | The name of the C compiler to be used.
  | If null, the respective value from ["CC", "defa
- "CXX"
  | The name of the C++ compiler to be used.
  | If null, the respective value from ["CC", "defa
- "CFLAGS"
  | The flags for CC to be used instead of the defa
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice
- "CXXFLAGS"
  | The flags for CXX to be used instead of the def
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "LDFLAGS"
  | The linker flags to be used instead of the defa
  | For libraries that should be linked in a non-st
  | adapting the default target ["CC", "defaults"]
  | choice
- "ADD_CFLAGS"
  | The flags to add to the default ones for CC.
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ADD_CXXFLAGS"
  | The flags to add to the default ones for CXX.
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ADD_LDFLAGS"
```

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

```
$ just-mr describe --main rules --rule CC/foreign/make library
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/62c162591...
 | Library produced by Configure and Make build and install.
 |
 | All variables accessible to commands and options are: "TMPDIR",
 | "LOCALBASE", "CC", "CXX", "CFLAGS", "CXXFLAGS", "LDFLAGS",
 | "AR", and "PREFIX". "LOCALBASE" contains the path to the
 | installed artifacts from "deps".
String fields
- "subdir"
  | The subdirectory that contains the configure and Makefile. Individual
  | directory components are joined with "/".
- "name"
  | The name of the library (without leading "lib" or trailing file name
  | extension), also used as name for pkg-config files.
- "version"
  | The library version, used for pkg-config files. Individual version
  | components are joined with ".".
- "stage"
  | The logical location of the public headers and library files.
  | Individual directory components are joined with "/".
- "configure"
  | Run ./configure if non-empty.
- "configure_options"
  | The configure options (the "--prefix" option is automatically set.
- "targets"
  | The Make targets to build in the specified order
  | (default: ["install"]).
- "prefix"
  | The prefix used for the Make target. The path will be made absolute
  | and individual directory components are joined with "/". If no
  | prefix is specified, the value from the config variable "PREFIX" is
  | taken, with the default value being "/".
- "options"
  | Make options for the build phase.
  | (e.g., ["-f", "Makefile", "ARCH=x86"])
- "jobs"
  | Number of jobs to run simultaneously. If omitted, Make's default
  | number is used.
- "pre_cmds"
  | List of commands executed in the project directory before calling
  | Configure or Make. Useful for renaming files or directories. Note
  | that data between "pre_cmds" and "post_cmds" can be exchanged via
  | "$TMPDIR", which is uniquely reserved for this action.
- "post_cmds"
  | List of commands executed in the install directory after successful
  | installation but before the output files are collected. Useful for
  | renaming files or directories (e.g., in case of SONAME mismatch). Note
  | that data between "pre_cmds" and "post_cmds" can be exchanged via
  | "$TMPDIR", which is uniquely reserved for this action.
- "out_hdrs"
  | Paths to produced public header files. The path is considered
  | relative to the include directory, which be set via "hdr_prefix".
  | Note that "out_hdrs" and "out_hdr_dirs" may not overlap.
- "out_hdr_dirs"
  | Paths to produced public header directories. The path is considered
  | relative to the include directory, which be set via "hdr_prefix".
  | Note that "out_hdrs" and "out_hdr_dirs" may not overlap.
- "out_libs"
  | Paths to produced library files. The path is considered relative
  | to the library directory, which be set via "lib_prefix".
  | Order matters in the case of one library depending on another.
- "cflags"
  | List of compile flags set for this target and its consumers.
- "ldflags"
  | Additional linker flags that are required for t
  | produced libraries.
- "pkg-config"
  | Pkg-config file for optional infer of public cf
  | multiple files are specified (e.g., one depends
  | first one is used as entry. Note that if this f
  | tool "pkg-config" must be available in "PATH",
  | from ["CC", "defaults"] or the "ENV" variable.
- "hdr_prefix"
  | Prefix where headers will be installed by Make.
  | components are joined with "/". Defaults to "in
- "lib_prefix"
  | Prefix where libraries will be installed by Mak
  | directory components are joined with "/". Defau
  | not set.
- "pc_prefix"
  | Prefix where pkg-config files will be installed
  | directory components are joined with "/". Defau
  | "lib/pkgconfig" if not set.
Target fields
- "project"
  | The Make project directory. It should contain a
- "deps"
  | Public dependency on other CC libraries.
- implict dependency
  - ["@","rules","CC","prebuilt/read_pkgconfig.py"]
- implict dependency
  - ["@","rules","CC","defaults"]
- implict dependency
  - ["@","rules","CC/foreign","expand_exec"]
- implict dependency
  - ["@","rules","CC/foreign","defaults"]
```

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

```
$ just-mr describe --main rules CC/foreign defaults
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/62c162591...
[["@","rules","CC/foreign","defaults"],{}] is defined by user-defined rule ...

 | A rule to provide defaults for foreign rules.
 | All foreign rules take their defaults for MAKE, CMAKE, etc from
 | the target ["CC/foreign", "defaults"].
String fields
- "MAKE"
  | The make binary to use
- "CMAKE"
  | The cmake binary to use
- "PATH"
  | Path for looking up the tools. Individual paths are joined with
  | with ":". Specifying this field extends values from "base".
- "SYSTEM_TOOLS"
  | List of tools ("MAKE", "CMAKE") that should be taken from
  | the system instead of from "toolchain" (if specified).
Target fields
- "base"
  | Other targets (using the same rule) to inherit values from.
- "toolchain"
  | Optional toolchain directory. A collection of artifacts that provide
  | the tools MAKE, CMAKE. Note that only artifacts of
  | the specified targets are considered (no runfiles etc.). Specifying
  | this field extends artifacts from "base".
Variables taken from the configuration
- "ARCH"
- "HOST_ARCH"
Result
- Artifacts
- Runfiles
$
```

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

```
$ just-mr describe --main rules --rule CC/foreign/cmake data
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/62c162591...
 | Data produced by CMake configure, build, and install.
 |
 | All variables accessible to commands and options are: "TMPDIR",
 | "LOCALBASE", "CC", "CXX", "CFLAGS", "CXXFLAGS", "LDFLAGS",
 | and "AR". "LOCALBASE" contains the path to the installed artifacts
 | from "deps".
String fields
- "subdir"
  | The subdirectory that contains the entry CMakeLists.txt. Individual
  | directory components are joined with "/".
- "options"
  | CMake options for the configuration phase.
  | (e.g., ["-GNinja", "-Ax64"])
- "defines"
  | CMake defines for the configuration phase.
  | (e.g., ["CMAKE_BUILD_TYPE=Release"])
- "targets"
  | The CMake targets to build in the specified order
  | (default: ["install"]).
- "jobs"
  | Number of jobs to run simultaneously. If omitted, CMake's default
  | number is used.
- "pre_cmds"
  | List of commands executed in the project directory before calling
  | CMake. Useful for renaming files or directories. Note that data
  | between "pre_cmds" and "post_cmds" can be exchanged via
  | "$TMPDIR" which is uniquely reserved for this action.
- "post_cmds"
  | List of commands executed in the install directory after successful
  | installation but before the output files are collected. Useful for
  | renaming files or directories. Note that data between "pre_cmds" and
  | "post_cmds" can be exchanged via "$TMPDIR", which is uniquely
  | reserved for this action. The CMake source and build directory can be
  | accessed via "$CMAKE_SOURCE_DIR" and "$CMAKE_BINARY_DIR",
  | respectively.
- "out_files"
  | Paths to the produced output files. The paths are considered relative
  | to the install directory.
  | Note that "out_files" and "out_dirs" may not overlap.
- "out_dirs"
  | Paths to the produced output directories. The paths are considered
  | relative to the install directory.
  | Note that "out_files" and "out_dirs" may not overlap.
Target fields
- "project"
  | The CMake project directory. It should contain a single tree artifact
- implict dependency
  - ["@","rules","CC","defaults"]
- implict dependency
  - ["@","rules","CC/foreign","expand_exec"]
- implict dependency
  - ["@","rules","CC/foreign","defaults"]
Variables taken from the configuration
- "ARCH"
- "HOST_ARCH"
- "CC"
  | The name of the C compiler to be used.
  | If null, the respective value from ["CC", "defaults"] will be taken.
- "CXX"
  | The name of the C++ compiler to be used.
  | If null, the respective value from ["CC", "defaults"] will be taken.
- "CFLAGS"
  | The flags for CC to be used instead of the defa
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice
- "CXXFLAGS"
  | The flags for CXX to be used instead of the def
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "LDFLAGS"
  | The linker flags to be used instead of the defa
  | For libraries that should be linked in a non-st
  | adapting the default target ["CC", "defaults"]
  | choice
- "ADD_CFLAGS"
  | The flags to add to the default ones for CC.
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ADD_CXXFLAGS"
  | The flags to add to the default ones for CXX.
  | For libraries that should be built in a non-sta
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ADD_LDFLAGS"
  | The linker flags to add to the default ones.
  | For libraries that should be linked in a non-st
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ENV"
  | The environment for any action generated.
  | If null, the respective value from ["CC", "defa
- "AR"
```

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools

```
$ just-mr describe --main rules --rule CC/foreign/shell data
INFO: Performing repositories setup
INFO: Found 2 repositories to set up
INFO: Setup finished, exec ["just","describe","-C","/worker/build/62c162591...
 | Data produced by generic shell commands with toolchain support.
 |
 | All variables accessible to commands and options are: "TMPDIR",
 | "LOCALBASE", "WORKDIR", "DESTDIR", "CC", "CXX", "CFLAGS",
 | "CXXFLAGS", "LDFLAGS", and "AR". "LOCALBASE" contains the path
 | to the staged artifacts from "localbase" and the installed artifacts
 | from "deps". Furthermore, the variable "ACTION_DIR" points to the
 | current action directory, if needed for achieving reproducibility.
String fields
- "cmds"
  | List of commands to execute by "sh". Multiple commands will be
  | joined with the newline character.
- "outs"
  | Paths to the produced output files in "DESTDIR".
- "out_dirs"
  | Paths to the produced output directories in "DESTDIR".
Target fields
- "project"
  | The project directory. It should contain a single tree artifact.
  | It's path can be accessed via the "WORKDIR" variable.
- "localbase"
  | Artifacts to stage to "LOCALBASE".
- "deps"
  | CC targets to install to "LOCALBASE".
- implict dependency
  - ["@","rules","CC","defaults"]
- implict dependency
  - ["@","rules","CC/foreign","expand_exec"]
- implict dependency
  - ["@","rules","CC/foreign","defaults"]
Variables taken from the configuration
- "CC"
  | The name of the C compiler to be used.
  | If null, the respective value from ["CC", "defaults"] will be taken.
- "CXX"
  | The name of the C++ compiler to be used.
  | If null, the respective value from ["CC", "defaults"] will be taken.
- "CFLAGS"
  | The flags for CC to be used instead of the default ones.
  | For libraries that should be built in a non-standard way; usually
  | adapting the default target ["CC", "defaults"] is the better
  | choice
- "CXXFLAGS"
  | The flags for CXX to be used instead of the default ones.
  | For libraries that should be built in a non-standard way; usually
  | adapting the default target ["CC", "defaults"] is the better
  | choice.
- "LDFLAGS"
  | The linker flags to be used instead of the default ones.
  | For libraries that should be linked in a non-standard way; usually
  | adapting the default target ["CC", "defaults"] is the better
  | choice
- "ADD_CFLAGS"
  | The flags to add to the default ones for CC.
  | For libraries that should be built in a non-standard way; usually
  | adapting the default target ["CC", "defaults"] is the better
  | choice.
- "ADD_CXXFLAGS"
  | The flags to add to the default ones for CXX.
  | For libraries that should be built in a non-standard way; usually
  | adapting the default target ["CC", "defaults"] is the better
  | choice.
- "ADD_LDFLAGS"
  | The linker flags to add to the default ones.
  | For libraries that should be linked in a non-st
  | adapting the default target ["CC", "defaults"]
  | choice.
- "ENV"
  | The environment for any action generated.
  | If null, the respective value from ["CC", "defa
- "AR"
  | The archive tool to used for creating the libra
  | If null, the respective value from ["CC", "defa
- "PREFIX"
  | The absolute path that is used as prefix inside
  | files. The default value for this variable is "
  | is ignored if the field "prefix" is set.
- "BUILD_POSITION_INDEPENDENT"
  | Build position independent code.
- "TIMEOUT_SCALE"
  | The scaling of the timeout for the invocation o
  | Defaults to 10.
Result
- Artifacts
- Runfiles

$
```

Interact with Foreign Build Tools

- many interesting libraries are built using other build tools
- want to bundle—but writing `just` target files is effort!

⇝ call the foreign tool
(one huge action, but updates kind-of rare, so shared caching saves)

Toolchain

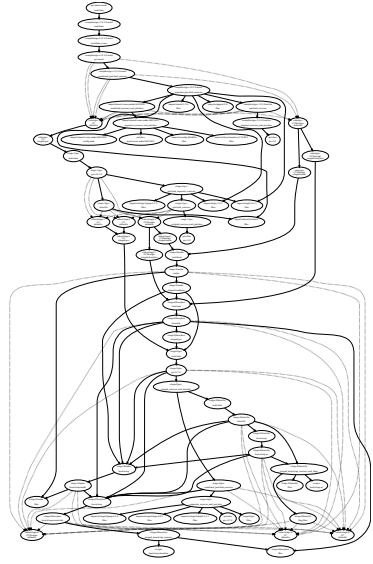
- Cooperation partners want to get the same binaries
  … but work in different environments

Toolchain

- Cooperation partners want to get the same binaries
  … but work in different environments
- ⇝ Bootstrap all tools
  … by first building the production compiler
  using the host `C` compiler

Toolchain

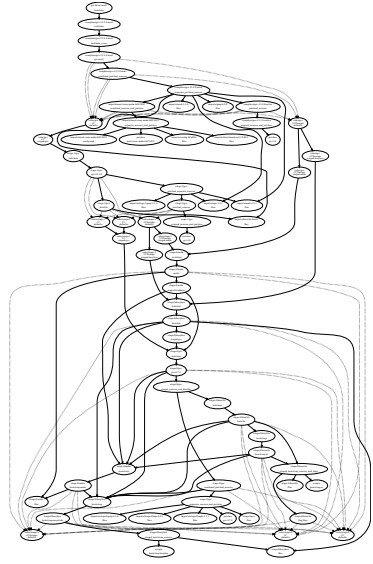
- Cooperation partners want to get the same binaries
  … but work in different environments
- ⇝ Bootstrap all tools
  … by first building the production compiler
  using the host C compiler
- Details are a bit more complicated
  e.g., modern C compilers are written in C++



HUAWEI

Toolchain

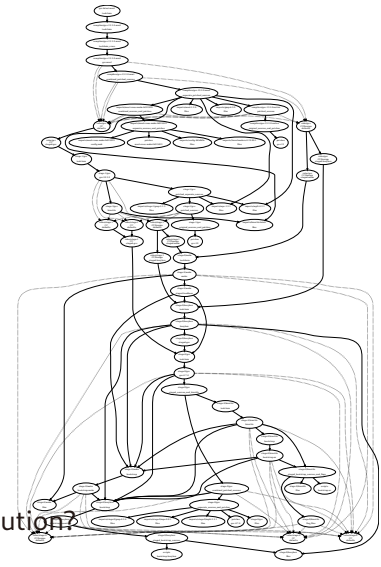
- Cooperation partners want to get the same binaries
  … but work in different environments
- ⇝ Bootstrap all tools
  … by first building the production compiler
  using the host `C` compiler
- Details are a bit more complicated
  e.g., modern `C` compilers are written in `C++`
- Now we have
  - rules for foreign build systems
  - bootstrapped modern `gcc`, `clang`, `make`, `cmake`, `busybox`, `python3`
  - an easy way to transitively import dependencies: `just-import-git`, `just-deduplicate-repositories`



Toolchain

- Cooperation partners want to get the same binaries
  … but work in different environments
- ⇝ Bootstrap all tools
  … by first building the production compiler
  using the host `C` compiler
- Details are a bit more complicated
  e.g., modern `C` compilers are written in `C++`
- Now we have
  - rules for foreign build systems
  - bootstrapped modern `gcc`, `clang`, `make`, `cmake`, `busybox`, `python3`
  - an easy way to transitively import dependencies: `just-import-git`, `just-deduplicate-repositories`

Is that the beginning of a new (“the distributed”) distribution?
And if so, is that good, bad, or `xkcd/927`?



Thank You!

- Sources
  - https://github.com/just-buildsystem/justbuild
  - https://github.com/just-buildsystem/rules-cc
  - https://github.com/just-buildsystem/bootstrappable-toolchain
- Background
  - https://bootstrappable.org/
  - https://reproducible-builds.org/