Trekking through dependencies

Introducing ttrek

Neofytos Dimitriou - July, 2024

Demo

Loading…

Why?

For the fun of it but…

In the past

- Bash Scripts
  - Designed to install specific versions of packages
- Build Tools
  - Designed with one-time execution approach
- OS Package Managers
  - apt-get reports Tcl 8.6.11 on my desktop at the moment (June 2024)
  - Global installation
- Other
  - vcpkg, conan, xrepo

ttrek

It employs a SAT solver for dependency resolution
Has its own JSON specification file
It creates a virtual environment directory tree

resolvo in ttrek 1/4

Dependency Resolution

- C++ bindings to resolvo are used under the hood
- SAT solver tailored for dependency resolution
- Can explain visually why dependencies are not satisfiable

```
The following packages are incompatible
└─ tcl 9.0.0-beta.2 can be installed with any of the following options:
   └─ tcl 9.0.0-beta.2
└─ tcl 8.6.14 is locked, but another version is required as reported above
```

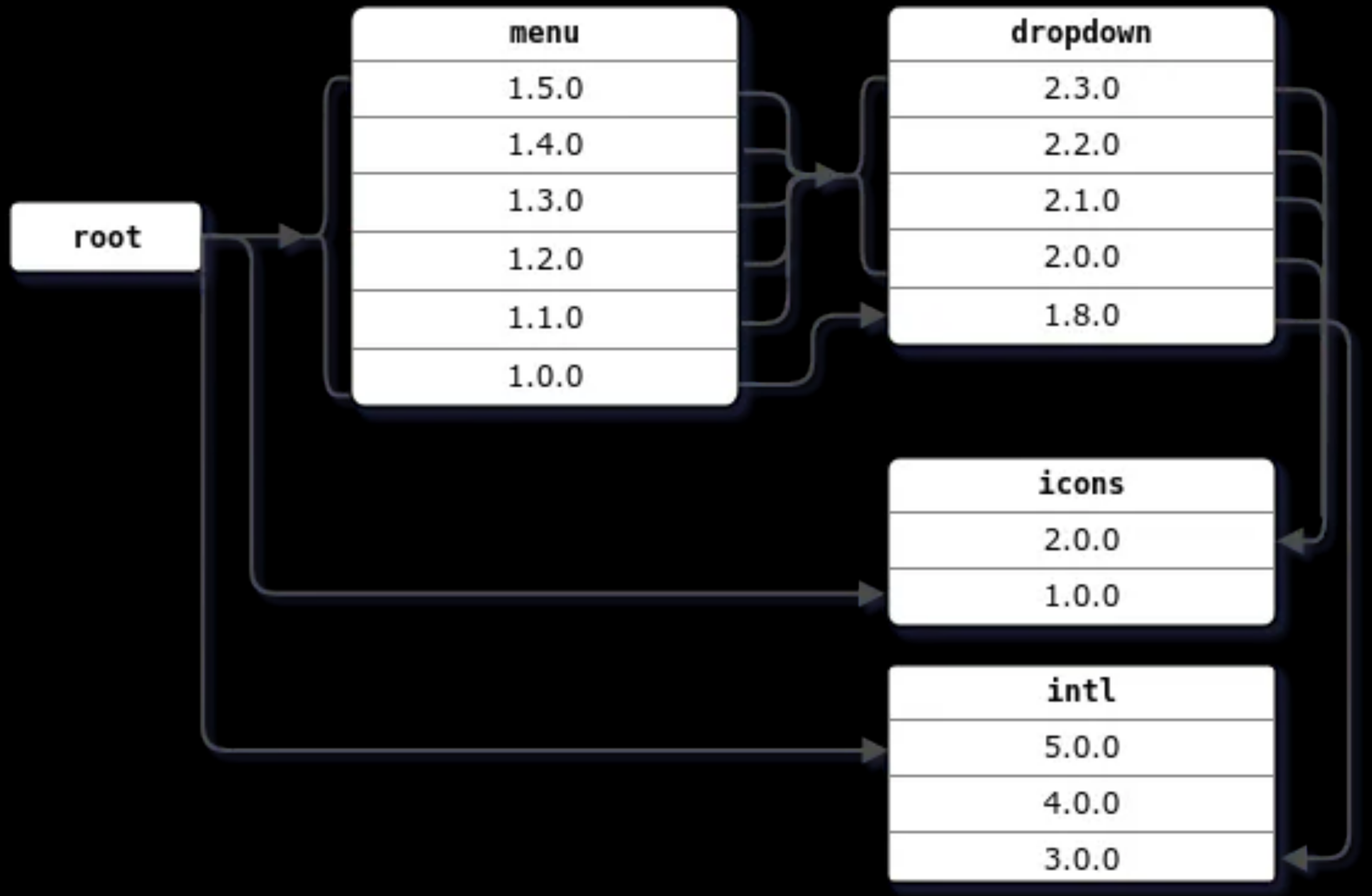
resolvo in ttrek 2/4

Example: unsat after backtracking

```
$ ./ttrek install example-bt-b example-bt-c example-bt-e
The following packages are incompatible
├─ example-bt-c * can be installed with any of the following options:
│  └─ example-bt-c 1.0.0 | 2.0.0 would require
│     └─ example-bt-d 2.0.0, which can be installed with any of the following options:
│        └─ example-bt-d 2.0.0
└─ example-bt-b * cannot be installed because there are no viable options:
   └─ example-bt-b 7.0.0 | 6.0.0 would require
      └─ example-bt-d 1.0.0, which cannot be installed because there are no viable options:
         └─ example-bt-d 1.0.0, which conflicts with the versions reported above.
```

resolvo in ttrek 3/4

Example: pubgrub article package graph



resolvo in ttrek 4/4

Example: pubgrub article unsat

```
$ ./ttrek install menu icons@1.0.0 intl@5.0.0
The following packages are incompatible
├─ menu * can be installed with any of the following options:
│  └─ menu 1.5.0 would require
│     └─ dropdown >=2.0.0, <3.0.0, which can be installed with any of the following options:
│        └─ dropdown 2.0.0 would require
│           └─ icons 2.0.0, which can be installed with any of the following options:
│              └─ icons 2.0.0
├─ intl 5.0.0 can be installed with any of the following options:
│  └─ intl 5.0.0
└─ icons 1.0.0 cannot be installed because there are no viable options:
   └─ icons 1.0.0, which conflicts with the versions reported above.
```

ttrek.json

The Specification File

- Blueprint for dependencies
- Outlines packages and version ranges
- Specifies additional build instructions (work in progress)

```
{
    "name": "aws-sdk-cpp",
    "version": "1.11.157",
    "scripts": {
    },
    "dependencies": {
        "openssl": "^3.0.13",
        "curl": "^8.7.1"
    },
    "devDependencies": {
    },
    "build": {
        "linux.x86_64": [
            {
                "cmd": "git",
                "url": "https://github.com/aws/aws-sdk-cpp",
                "branch": "1.11.157",
                "recurse-submodules": true,
                "shallow-submodules": true
            },
            { "cmd": "cd" },
            {
                "cmd": "cmake_config",
                "options": [
                    { "name": "BUILD_SHARED_LIBS", "value": "ON" },
                    { "name": "CMAKE_BUILD_TYPE", "value": "Release" },
                    { "name": "BUILD_ONLY", "value": "s3;dynamodb;lambda;
```

Virtual Environment

Local Vs Global Vs User Mode

- Problem:
  - Not possible for one installation to meet the requirements of every application
  - If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing version 1.0 or 2.0 will leave one application unable to run.
- Solution:
  - Create a virtual environment: a self-contained directory tree that contains the packages. In our case, ttrek-venv directory tree under the project directory.

ttrek

The Client

- ttrek init — creates ttrek-venv directory tree
- ttrek install — three strategies: latest or locked or favored (default)
- ttrek update — three strategies: latest (default) or locked or favored
- ttrek uninstall
- ttrek run — runs scripts in ttrek-venv/local/bin with proper env vars
- ttrek ls — list installed packages
- trrek search — search registry packages (still in progress)

ttrek.sh

The Registry Website

- Back to Bash Scripts :)
- ~~Generated Install Scripts~~
- Browsing packages
- Distributing the ttrek client

```
git -C $BUILD_DIR clone --depth 1 --branch 1.11.157 \
  --recurse-submodules --shallow-submodules \
  https://github.com/aws/aws-sdk-cpp
cd $BUILD_DIR/aws-sdk-cpp
mkdir build
cd build
cmake .. \
  -DBUILD_SHARED_LIBS=ON \
  -DCMAKE_BUILD_TYPE=Release \
  -DBUILD_ONLY="s3;dynamodb;lambda;sqs;iam;transfer;sts
  -DENABLE_TESTING=OFF \
  -DAUTORUN_UNIT_TESTS=OFF \
  -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR \
  -DCMAKE_PREFIX_PATH=$INSTALL_DIR/ > $BUILD_LOG_DIR/aw
cmake --build . --config=Release > $BUILD_LOG_DIR/aws-s
cmake --install . --config=Release > $BUILD_LOG_DIR/aws
```

The Result 1/3

```
$ ./ttrek install twebserver
The following packages will be installed:
openssl@3.2.1
zlib@1.3.1
tcl@9.0.0-beta.2
twebserver@1.47.43
Do you want to proceed? [y/N] y
```

```
$ ./ttrek install openssl@3.0.13
The following packages will be installed:
openssl@3.0.13
twebserver@1.47.43 (reverse dependency)
Do you want to proceed? [y/N]
```

```
{
    "name": "twebserver",
    "version":  "1.47.43",
    "scripts":  {
    },
    "dependencies": {
        "openssl": "^3.0.13",
        "tcl": ">=8.6.14"
    },
```

The Result 2/3

```
$ ./ttrek update
The following packages will be installed:
openssl@3.2.1
twebserver@1.47.43 (reverse dependency)
Do you want to proceed? [y/N] y

Package [1/2]: openssl v3.2.1; Stage [1/4]: Getting sources...
Package [1/2]: openssl v3.2.1; Stage [2/4]: Configuring sources...
Package [1/2]: openssl v3.2.1; Stage [3/4]: Building...
Package [1/2]: openssl v3.2.1; Stage [4/4]: Installing...
Package [1/2]: openssl v3.2.1; Stage: Done.
Added dependency openssl to spec: ^3.2.1
Package [2/2]: twebserver v1.47.43; Stage [1/4]: Getting sources...
```

The Result 3/3

```
$ ./ttrek uninstall zlib -autoremove
The following packages will be uninstalled:
openssl
twebserver
tcl
zlib
Do you want to proceed? [y/N]
```

Many thanks to Konstantin Kushnir!

But the programmer is a magician, and his whole magic is in this, that he does say “give me the dependency tree for x, y, z”, and lo! It is the dependency tree for x, y, z.

- Adolfo Ochagavia (The magic of dependency resolution)

Coming Soon

https://ttrek.sh

Dilemma 1

Source Vs Pre-built Binaries

- Pre-built binaries:
  - Easier to distribute / More expensive to maintain
  - Automatic generation possible?
  - Auto-publish to OS package managers?
- Building from source code:
  - More cost-effective
  - Reverse Dependencies are easy to deal with when building from source code
  - External dependencies (e.g. cmake) make it harder to use

Dilemma 2

Local Vs Remote Registry

- “sync” sub-command
- Related issues for discussion:
  - Who maintains and controls the registries?
  - How are contributors and packages authenticated?
  - Will there be oversight for security reasons (malware, etc)?