Buckaroo Documentation

LoopPerfect Limited

Nov 11, 2018

Contents

Buckaroo is a dependency manager for C/C++. Using Buckaroo, it is easy to add external modules to your project in a controlled and cross-platform manner.

To use Buckaroo, you will also need to install Buck. Buck is a build system developed and maintained by Facebook, and it is used by all Buckaroo packages.

CHAPTER 1

Contents

1.1 Installation

Buckaroo is available for macOS, Linux and Windows (preview).

1.1.1 macOS

Buckaroo can be installed using Homebrew.

Add Facebook's tap so that Homebrew can find Buck.

```
brew tap facebook/fb
brew tap loopperfect/lp
brew install loopperfect/lp/buckaroo
```

The Homebrew formula will install Buck and Java, if required.

Verify your installation with:

```
buckaroo version
```

Finally, fetch the cookbook with:

```
buckaroo update
```

1.1.2 Linux

Debian and Ubuntu via .deb

We provide a *.deb* file for Debian and Ubuntu. To install it, download buckaroo.deb and double-click to install.

Alternatively, you can use the command-line:

```
wget -O buckaroo.deb 'https://github.com/LoopPerfect/buckaroo/releases/download/v1.4.
↪1/buckaroo.deb'
sudo dpkg -i buckaroo.deb
```

Finally, fetch the cookbook with:

```
buckaroo update
```

Linuxbrew

Buckaroo can be installed using Linuxbrew.

Add Facebook's tap so that Linuxbrew can find Buck.

```
brew tap facebook/fb
brew tap loopperfect/lp
brew install loopperfect/lp/buckaroo
```

The Linuxbrew formula will install Buck and Java, if required.

Verify your installation with:

```
buckaroo version
```

Finally, fetch the cookbook with:

```
buckaroo update
```

1.1.3 Windows (preview)

Ensure that you have Buck installed, then clone the Buckaroo source-code from GitHub:

```
git clone git@github.com:njlr/buckaroo.git
cd buckaroo
git checkout tags/v1.0.0
```

Build Buckaroo with Buck:

```
buck build :buckaroo-cli
```

Buck will output a runnable Jar file in the output folder:

```
java -jar .\\buck-out\\gen\\buckaroo-cli.jar
```

Ensure that this command is on your PATH.

Finally, fetch the cookbook with:

```
buckaroo update
```

1.1.4 Analytics

By default, Buckaroo will report usage statistics to our servers. These logs allow us to improve Buckaroo by targeting real-world usage. All logs are transferred over HTTPS and are not shared with any third-party.

What is Shared?

The analytics events are as follows:

```
{
  session, // Random UUID generated on installation
  data: {
    os, // The OS name, e.g. "macOS"
    version, // The version of Buckaroo installed
    command // The command sent to Buckaroo
  }
}
```

If in doubt, please refer to the source-code of Buckaroo or drop us an email.

Disabling Analytics

If you wish to disable analytics, follow these steps:

1. Launch Buckaroo at least once:

```
buckaroo version
```

2. Open the *buckaroo.json* file in your Buckaroo home folder:

```
open ~/.buckaroo/config.json
```

3. Remove the property *"analytics"*. For example:

```
{
  "cookBooks": [
    {
      "name": "buckaroo-recipes",
      "url": "git@github.com:loopperfect/buckaroo-recipes.git"
    }
  ]
}
```

1.2 Quickstart

1.2.1 Creating a Project

The fastest way to get started is to use the Quickstart feature.

First, create a directory for your project:

```
mkdir my-project
cd my-project
```

Now, run the Buckaroo quickstart command:

```
buckaroo quickstart
```

This command will generate a hello-world application, folders for your source-code and a Buck build file. Buckaroo does not require a particular project layout, so feel free to tweak the Buck file.

Let's verify that the project is working as expected:

```
buck run :main
```

You now have everything ready to start installing dependencies.

1.2.2 Adding a dependency

Once you have a project file, we can start adding dependencies. Let's add range-v3 by Eric Niebler. range-v3 is a powerful range library for C++ 11 and up.

```
buckaroo install ericniebler/range-v3
```

Buckaroo will have downloaded the range-v3 source-code from GitHub and installed it locally in your project folder. We can now use the library in a sample application!

1.2.3 Sample Application

Our example requires some C++ 14 features, so if your compiler does not enable them by default we will need to update the project's *.buckconfig* file.

Update *.buckconfig* to:

```
[cxx]
  cxxflags = -std=c++14
```

Now, let's update the main.cpp file to a simple range-v3 example:

```
#include <iostream>
#include <vector>
#include <range/v3/all.hpp>

int main() {
  auto const xs = std::vector<int>({ 1, 2, 3, 4, 5 });
  auto const ys = xs
    | ranges::view::transform([](auto x) { return x * x; })
    | ranges::to_vector;
  for (auto const& i : ys) {
    std::cout << i << std::endl;
  }
  return 0;
}
```

Run the project again and you will see a list of square numbers, computed by range-v3.

```
buck run :main
```

1.2.4 .gitignore

If you are tracking your project with Git, add the following to your .gitignore:

```
/buck-out/
/.buckd/
/buckaroo/
BUCKAROO_DEPS
.buckconfig.local
```

1.2.5 Explore Buckaroo

range-v3 is just one of the many packages already available for Buckaroo. You can browse them on buckaroo.pm, request more on the wishlist or *create your own*!

1.3 Command Line Interface

Buckaroo provides a command-line interface for managing your project's dependencies. The following commands are the recommended way to use Buckaroo.

1.3.1 Init

```
buckaroo init
```

Init is used to generate a project file in the current directory.

1.3.2 Quickstart

```
buckaroo quickstart
```

Quickstart is similar to Init, but also generates the necessary boiler-plate for a new C++ project. You should use Quickstart when starting a new project.

1.3.3 Resolve

```
buckaroo resolve
```

Reads the project file in the working directory and runs the dependency resolution algorithm, storing the results in the lock file. If there is already a lock file present, then it is overwritten. No dependencies are actually installed.

1.3.4 Install

Install adds and installs dependencies to your project.

```
buckaroo install google/gtest
```

or just

```
buckaroo i google/gtest
```

Install can be used to add dependencies to your project. Since introducing a new dependency can result in a versioning conflict, the resolution process is run again. This may overwrite your lock file.

Furthermore you can also ommit the organization name and list more than one package at the same time

```
buckaroo install gtest boost/asio
```

Install can also fetch *buckaroo compatible projects* from GitHub, BitBucket and GitLab using the following syntax:

```
buckaroo install github+loopperfect/neither
buckaroo install bitbucket+org/example
buckaroo install gitlab+org/example
```

If you do not supply a module name, then the existing dependencies of the project are fetched and installed. If there is no lock file present, then the resolution process will be run first.

```
buckaroo install
```

1.3.5 Resolve

Resolves the dependencies and regenerates the lock-file (buckaroo.lock.json). The lock-file specifies the exact versions of all dependencies to ensure the reproducibility of your project.

Resolve is automatically called when installing or uninstalling a dependency. You may want to run this if one of your dependencies updates.

1.3.6 Uninstall

```
buckaroo uninstall google/gtest
```

or

```
buckaroo u google/gtest
```

Uninstall can be used to remove a dependency from your project. Note that the remaining dependencies are recomputed since their resolved versions may have changed as a result. This may overwrite your lock file.

1.3.7 Upgrade

```
buckaroo upgrade
```

Upgrades the installed dependencies to the latest compatible version.

1.3.8 Update

```
buckaroo update
```

Updates the cook-books installed on your system. This allows you to use benefit from recipe improvements, additions and fixes since you first installed Buckaroo.

1.3.9 Version

```
buckaroo version
```

Outputs the version of Buckaroo that is installed.

1.3.10 Help

```
buckaroo help
```

1.4 Integrations

Buckaroo has integration with leading Git hosting providers, which allows for recipes to exist directly in source-control. To use a hosted project as a recipe, the following conditions must be met:

- The project must have at least one tag, which is a semantic version (e.g. *"v1.0"*)
- The project must contain a `buckaroo.json` file at the root level.

If you would like to create a recipe from a project, see *the packaging guide*.

1.4.1 GitHub

A GitHub dependency is referred to with the prefix `github+`. So, for example, to install `LoopPerfect/neither` you would run:

```
buckaroo install github+loopperfect/neither
```

Buckaroo will generate the Git URL from the project name and owner, fetch the repository and read the `buckaroo.json` file.

1.4.2 BitBucket

A BitBucket dependency is referred to with the prefix `bitbucket+`. So, for example, to install `org/example` you would run:

```
buckaroo install bitbucket+org/example
```

1.4.3 GitLab

A GitLab dependency is referred to with the prefix `gitlab+`. So, for example, to install `org/example` you would run:

```
buckaroo install gitlab+org/example
```

1.4.4 Transitive Dependencies

VCS-hosted recipes can even have transitive dependencies! These are specified using the usual syntax in the project file of the repository. When resolving dependencies, Buckaroo will read this project file to generate further recipes to resolve. Note that the lock file of the dependency is ignored, and the cookbook of the current system is used to resolve the dependencies of the recipe.

1.4.5 What if a project is deleted, or a tag is changed?

Once a VCS-hosted dependency has been resolved, the exact commit hash of the tag is written to the lock file. This means that future installations will use the same version that was resolved on the first install. If this version becomes unavailable later, the resolution process can be re-run to fetch the current available version.

1.5 Creating a Package

This is a guide for creating a Buckaroo recipe from a GitHub, BitBucket or GitLab project. This is the quickest way to create and manage a package that already lives in source-control. It is also recommended, since it allows you to control updates to your recipe.

For this guide, we will be adding Buckaroo support for an example project. However, the steps are generic, so you should be able to follow them using your own project.

If you would like to jump straight to a working GitHub recipe, take a look at LoopPerfect/neither. The important files are `BUCK` and `buckaroo.json`.

1.5.1 Requirements

To follow this guide, you will need the following:

- Git
- A GitHub account
- Buckaroo (see *Installation*)
- Buck

The steps are nearly identical for BitBucket and GitLab.

1.5.2 1. Fork the example project

Fork the example project on GitHub.

1.5.3 2. Fetch the Project

Next, fetch your fork from GitHub using Git:

```
git clone https://github.com/<YOUR_GITHUB_USERNAME>/buckaroo-github-example.git
cd buckaroo-github-example
```

1.5.4 3. Ensure that the project builds with Buck

All Buckaroo packages build with Buck, so we need to make sure that the `BUCK` files are correct.

In our example, we have a single library called `example`. Try building it using Buck:

```
buck build :example
```

If you are using your own project that does not yet build with Buck, then you will need to write the appropriate Buck files. Head over to the Buck website for more guidance, or you can follow this article on Hackernoon.

Note: To work with Buckaroo, your Buck target must be marked `PUBLIC`. To do this, simply add the following to your target definition:

```
visibility = [
  'PUBLIC',
]
```

1.5.5 4. Create the Project File

Now that we are sure the project can build with Buck, we need to create a project file. The project file gives Buckaroo various bits of meta-data about your recipe.

```
buckaroo init
```

This will create a `buckaroo.json` file. Open it, and ensure that the target field points to your Buck target.

For our example, the target is `"example"`. The final `buckaroo.json` file might look like this:

```
{
  "name": "buckaroo-github-example",
  "target": "example"
}
```

Commit the project file to GitHub:

```
git add buckaroo.json
git commit -m "Adds Buckaroo project file"
git push
```

1.5.6 5. Create a release

Head over to the GitHub web-page for your project and create a release. It is important that you name the release in a format that Buckaroo understands. Buckaroo expects a version number prefixed with a `"v"`. Some valid names are:

- `"v1.0.0"`
- `"v0.2"`
- `"v3"`

For this guide, we will name the release `"v0.1.0"`.

1.5.7 6. Test your package

Create a new folder alongside your project directory:

```
cd ../
mkdir test
cd test
```

Create a Buckaroo project in this directory:

```
buckaroo quickstart
```

Now we can install the GitHub recipe:

```
buckaroo install github+<YOUR_GITHUB_USERNAME>/buckaroo-github-example
```

You should see a few changes to your working directory:

- `buckaroo.lock.json` will contain an exact version of your GitHub recipe
- `buckaroo/` will contain a copy of your recipe
- `BUCKAROO_DEPS` will have been generated, ready to include in your `BUCK` file.

Open `test/src/main.cpp` and update it to use your recipe:

```
#include <iostream>
#include <sum.hpp>

int main() {
  std::cout << "sum(1, 2) = " << sum(1, 2) << std::endl;
  return 0;
}
```

Now run your program using Buck:

```
buck run :test
```

If everything has worked correctly, you will see the following output:

```
sum(1, 2) = 3
```

1.6 Anatomy of a Recipe

A Buckaroo recipe is a file that describes a module that can be imported into your projects. Recipes are encoded as JSON.

1.6.1 Example

For example, here's a recipe for the Beast HTTP library:

```
{
  "name": "Beast",
  "license": "BSL-1.0",
  "url": "https://github.com/vinniefalco/beast",
```

(continues on next page)

(continued from previous page)

```
  "github": {
    "owner": "vinniefalco",
    "project": "beast"
  },
  "versions": {
    "0.0.1": {
      "source": {
        "url": "https://github.com/vinniefalco/Beast/archive/
↪6d5547a32c50ec95832c4779311502555ab0ee1f.zip",
        "sha256": "8a600dfa3394164f79ad7dfa6942d8d4b6c6c7e5b8cc9b5f82519b682db25aae",
        "subPath": "Beast-6d5547a32c50ec95832c4779311502555ab0ee1f"
      },
      "buck": {
        "url": "https://raw.githubusercontent.com/njlr/Beast/
↪71d1bde64bf5ee52579441b00ad446959231d8d2/BUCK",
        "sha256": "1dce5d9f5c883e193e54c2dfc033a5485b9e8e87bb34d8c38ab03ed148ccd968"
      },
      "dependencies": {
        "boost/range": "1.63.0",
        "boost/intrusive": "1.63.0",
        "boost/lexical-cast": "1.63.0",
        "boost/math": "1.63.0",
        "boost/system": "1.63.0",
        "boost/asio": "1.63.0"
      }
    }
  }
}
```

Name

```
"name": "Beast",
```

This is the friendly name of the package.

Note that the file-name of the recipe is the identifier used by the `install` command, so the `name` field may contain spaces, etc.

Meta-data

```
"license": "BSL-1.0",
"url": "https://github.com/vinniefalco/beast",
"github": {
  "owner": "vinniefalco",
  "project": "beast"
},
```

This optional meta-data is used to generate the pages on buckaroo.pm. Note that the license should be a comma-separated list of SPDX codes.

Versions

```
"versions": {
```

The `versions` element is a dictionary of semantic versions to source locations.

```
"0.0.1": {
  "source": {
    "url": "https://github.com/vinniefalco/Beast/archive/
↪6d5547a32c50ec95832c4779311502555ab0ee1f.zip",
    "sha256": "8a600dfa3394164f79ad7dfa6942d8d4b6c6c7e5b8cc9b5f82519b682db25aae",
    "subPath": "Beast-6d5547a32c50ec95832c4779311502555ab0ee1f"
  },
  "buck": {
    "url": "https://raw.githubusercontent.com/njlr/Beast/
↪71d1bde64bf5ee52579441b00ad446959231d8d2/BUCK",
    "sha256": "1dce5d9f5c883e193e54c2dfc033a5485b9e8e87bb34d8c38ab03ed148ccd968"
  },
  "dependencies": {
    "boost/range": "1.63.0",
    "boost/intrusive": "1.63.0",
    "boost/lexical-cast": "1.63.0",
    "boost/math": "1.63.0",
    "boost/system": "1.63.0",
    "boost/asio": "1.63.0"
  }
}
```

Each version object contains information on how to fetch the source-code. In case the source-code does not contain a `BUCK` file, one may be injected using the information in the `buck` element.

Source

```
"source": {
  "url": "https://github.com/vinniefalco/Beast/archive/
↪6d5547a32c50ec95832c4779311502555ab0ee1f.zip",
  "sha256": "8a600dfa3394164f79ad7dfa6942d8d4b6c6c7e5b8cc9b5f82519b682db25aae",
  "subPath": "Beast-6d5547a32c50ec95832c4779311502555ab0ee1f"
},
```

Every `source` should point to a zip-file containing the source-code of the module. It is hashed to ensure integrity, so always choose a URL that is stable.

The `subPath` element can be used to change the root of the source-code.

For example, suppose the zip-file has this structure:

```
.
+-- module.zip
    +-- sources
        +-- include
            +-- math.hpp
            +-- utils.hpp
```

If the `subPath` is `"sources/include"`, then the extracted code would be:

```
.
+-- math.hpp
+-- utils.hpp
```

This feature is particularly useful for GitHub, which puts source-code into a sub-folder called `<project>-<commit>`.

Buck

```
"buck": {
  "url": "https://raw.githubusercontent.com/njlr/Beast/
↪71d1bde64bf5ee52579441b00ad446959231d8d2/BUCK",
  "sha256": "1dce5d9f5c883e193e54c2dfc033a5485b9e8e87bb34d8c38ab03ed148ccd968"
},
```

The `buck` element points to a remote Buck definition. If present, the remote `BUCK` file gets saved into the root folder of the source-code. This allows us to support packages that do not currently support Buck.

The `buck` element is only required when the source-code does not already contain a `BUCK` file.

Dependencies

```
"dependencies": {
  "boost/range": "1.63.0",
  "boost/intrusive": "1.63.0",
  "boost/lexical-cast": "1.63.0",
  "boost/math": "1.63.0",
  "boost/system": "1.63.0",
  "boost/asio": "1.63.0",
  "github+loopperfect/neither": "*"
}
```

The `dependencies` element defines the modules that the recipe requires to build. These are of the following format:

```
(<source>+)?<owner>/<project>: <version>
```

The source section is optional and is used to refer to recipes outside of the official cookbook. The most common source is GitHub (see github), and more will be added over time.

A small DSL is provided for versioning:

```
*                     // Any version
v1                    // Exactly version 1.0.0
2                     // Exactly version 2.0.0
1.2                   // Exactly version 1.2.0
1.2.3                 // Exactly version 1.2.3
1.0.1 - 4.3           // Between 1.0.1 and 4.3 inclusive
[ 7.2, 7.3, 8 ]       // Either 7.2.0, 7.3.0 or 8.0.0
>= 4.7                // Version 4.7 or greater
<= 6.5.1              // Version 6.5.1 or greater
```

When multiple versions of a dependency can be resolved, the higher version is always chosen.

1.7 Caching

Buckaroo will cache files downloaded from the network and clones of Git repositories. This prevents duplicate work, and can even enable Buckaroo to work offline in some cases.

1.7.1 Clearing the Cache

If you would like to reclaim some disc space, it is safe to delete the Buckaroo cache folder. The location of the cache folder will vary by platform:

macOS

```
~/Library/Caches/Buckaroo
```

Linux

```
~/.cache/Buckaroo
```

Windows

```
C:\Users\<USERNAME>\Local Settings\Application Data\Buckaroo\Cache
```

Any Other Platform

```
~/.buckaroo/caches
```