cget Documentation

Release 0.1.0

Paul Fultz II

March 02, 2017

Contents

Contents:

CHAPTER 1

Introduction

Cmake package retrieval. This can be used to download and install cmake packages. The advantages of using `cget` are:

- Non-intrusive: There is no need to write special hooks in cmake to use `cget`. One cmake file is written and can be used to install a package with `cget` or standalone.
- Works out of the box: Since it uses the standard build and install of cmake, it already works with almost all cmake packages. There is no need to wait for packages to convert over to support `cget`. Standard cmake packages can be already installed immediately.
- Decentralized: Packages can be installed from anywhere, from github, urls, or local files.

Installing cget

`cget` can be simply installed using `pip` (you can get pip from here):

```
pip install cget
```

Or installed directly with python:

```
python setup.py install
```

On windows, you may want to install pkgconfig-lite to support packages that use pkgconfig. This can be installed with `cget` as well:

```
cget install pfultz2/pkgconfig
```

Quickstart

We can also install cmake packages directly from source files, for example zlib:

```
cget install http://zlib.net/zlib-1.2.8.tar.gz
```

However, its much easier to install recipes so we don't have to remember urls:

```
cget install pfultz2/cget-recipes
```

Then we can install packages such as boost:

```
cget install boost
```

Or curl:

```
cget install curl
```

Usage

Installing a package

Any library that uses cmake to build can be built and installed as a package with `cget`. A source for package can be from many areas (see *Package source*). We can simply install `zlib` with its URL:

```
cget install http://zlib.net/zlib-1.2.8.tar.gz
```

We can install the package from github as well, using a shorten form. For example, installing John MacFarlane's implementation of CommonMark in C called cmark:

```
cget install jgm/cmark
```

Removing a package

A package can be removed by using the same source name that was used to install the package:

```
cget install http://zlib.net/zlib-1.2.8.tar.gz
cget remove http://zlib.net/zlib-1.2.8.tar.gz
```

If an alias was specified, then the name of the alias must be used instead:

```
cget install zlib,http://zlib.net/zlib-1.2.8.tar.gz
cget remove zlib
```

Testing packages

The test suite for a package can be ran before installing it, by using the `--test` flag. This will either build the `check` target or run `ctest`. So if we want to run the tests for zlib we can do this:

```
cget install --test http://zlib.net/zlib-1.2.8.tar.gz
```

Setting the prefix

By default, the packages are installed in the local directory `cget`. This can be changed by using the `--prefix` flag:

```
cget install --prefix /usr/local zlib:http://zlib.net/zlib-1.2.8.tar.gz
```

The prefix can also be set with the `CGET_PREFIX` environment variable.

Integration with cmake

By default, cget creates a cmake toolchain file with the settings necessary to build and find the libraries in the cget prefix. The toolchain file is at `$CGET_PREFIX/cget.cmake`. If another toolchain needs to be used, it can be specified with the `init` command:

```
cget init --toolchain my_cmake_toolchain.cmake
```

Also, the C++ version can be set for the toolchain as well:

```
cget init --std=c++14
```

Which is necessary to use modern C++ on many compilers.

CHAPTER 2

Package source

Directory

This will install the package that is located at the directory:

```
cget install ~/mylibrary/
```

There must be a `CMakeLists.txt` in the directory.

File

An archived file of the package:

```
cget install zlib-1.2.8.tar.gz
```

The archive will be unpacked and installed.

URL

A url to the package:

```
cget install http://zlib.net/zlib-1.2.8.tar.gz
```

The file will be downloaded, unpacked, and installed.

Github

A package can be installed directly from github using just the namespace and repo name. For example, John MacFarlane's implementation of CommonMark in C called cmark can be installed like this:

```
cget install jgm/cmark
```

A tag or branch can specified using the `@` symbol:

```
cget install jgm/cmark@0.24.1
```

Recipe

A recipe name can also be installed. See Using recipes for more info.

Aliasing

Aliasing lets you pick a different name for the package. So when we are installing `zlib`, we could alias it as `zlib`:

```
cget install zlib,http://zlib.net/zlib-1.2.8.tar.gz
```

This way the package can be referred to as `zlib` instead of `http://zlib.net/zlib-1.2.8.tar.gz`.

Requirements file

`cget` will install all packages listed in the `requirements.txt` file. Each requirement is listed on a new line.

`<package-source>`

This specifies the package source (see *Package source*) that will be installed.

`-H, --hash`
This specifies a hash checksum that should checked before installing the packaging. The type of hash needs to be specified with a colon first, and then the hash. So for md5, it would be something like `md5:6fc67d80e915e63aacb39bc7f7da0f6c`.

`-b, --build`
This is a dependency that is only needed for building the package. It is not installed as a dependent of the package, as such, it can be removed after the package has been installed.

`-t, --test`
`cget` will only install the dependency if the tests are going to be run. This dependency is also treated as a build dependency so the it can be removed after the package has been installed.

`-D, --define` `VAR=VALUE`
Extra configuration variables to pass to CMake.

`-X, --cmake`
This specifies an alternative cmake file to be used to build the library. This is useful for packages that don't have a cmake file.

CHAPTER 4

Commands

build

This will build a package, but it doesn't install it. This is useful over using raw cmake as it will use the cmake toolchain that was initialized by cget which sets cmake up to easily find the dependencies that have been installed by cget.

`<package-source>`

This specifies the package source (see *Package source*) that will be built.

`-p, --prefix` `PATH`
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

`-v, --verbose`
Enable verbose mode.

`-B, --build-path` `PATH`
Set the path for the build directory to use when building the package.

`-t, --test`
Test package after building. This will set the `BUILD_TESTING` cmake variable to true. It will first try to run the `check` target. If that fails it will call `ctest` to try to run the tests.

`-c, --configure`
Configure cmake. This will run either `ccmake` or `cmake-gui` so the cmake variables can be set.

`-C, --clean`
Remove build directory.

`-P, --path`
Show path to build directory.

`-D, --define` `VAR=VALUE`
Extra configuration variables to pass to CMake

`-T, --target` `TARGET`
Cmake target to build.

`-y, --yes`
Affirm all questions.

`-G, --generator` `GENERATOR`
Set the generator for CMake to use.

clean

This will clear the directory used by cget. This will remove all packages that have been installed, and any toolchain settings.

`-p, --prefix` `PATH`
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

`-v, --verbose`
Enable verbose mode.

`-y, --yes`
Affirm all questions.

init

This will initialize the cmake toolchain. By default, the `install` command will initialize a cmake toolchain if one doesn't exists. This allows setting different variable, such as setting C++ compiler or standard version.

`-p, --prefix` `PATH`
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

`-v, --verbose`
Enable verbose mode.

`-B, --build-path` `PATH`
Set the path for the build directory to use when building the package.

`-t, --toolchain` `FILE`
Set cmake toolchain file to use.

`--cxx` `COMPILER`
Set c++ compiler.

`--cxxflags` `FLAGS`
Set additional c++ flags.

`--ldflags` `FLAGS`
Set additional linker flags.

`--std` `TEXT`
Set C++ standard if available.

`-D, --define` `VAR=VALUE`
Extra configuration variables to pass to CMake.

`--shared`
Set toolchain to build shared libraries by default.

`--static`
Set toolchain to build static libraries by default.

install

A package can be installed using the `install` command. When a package is installed, `cget` configures a build directory with cmake, and then builds the `all` target and the `install` target. So, essentially, `cget` will run the equivalent of these commands on the package to install it:

```
mkdir build
cd build
cmake -DCMAKE_TOOLCHAIN_FILE=$CGET_PREFIX/cget/cget.cmake -DCMAKE_INSTALL_PREFIX=$CGET_PREFIX ..
cmake --build .
cmake --build . --target install
```

However, `cget` will always create the build directory out of source. The `cget.cmake` is a toolchain file that is setup by `cget`, so that cmake can find the installed packages. Other setting can be added about the toolchain as well(see init).

`<package-source>`

This specifies the package source (see *Package source*) that will be installed.

`-p, --prefix` `PATH`
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

`-v, --verbose`
Enable verbose mode.

`-B, --build-path` `PATH`
Set the path for the build directory to use when building the package.

`-U, --update`
Update package. This will rebuild the package even its already installed and replace it with the newly built package.

`-t, --test`
Test package before installing. This will set the `BUILD_TESTING` cmake variable to true. It will first try to run the `check` target. If that fails it will call `ctest` to try to run the tests.

`--test-all`
Test all packages including its dependencies before installing by running ctest or check target.

`-f, --file` `FILE`
Install packages listed in the file.

`-D, --define` `VAR=VALUE`
Extra configuration variables to pass to CMake.

`-G, --generator` `GENERATOR`
Set the generator for CMake to use.

`-X, --cmake`
This specifies an alternative cmake file to be used to build the library. This is useful for packages that don't have a cmake file.

`--debug`
Install the debug version of the package.

`--release`
Install the release version of the package.

list

This will list all packages that have been installed.

-p, --prefix PATH
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

-v, --verbose
Enable verbose mode.

pkg-config

This will run pkg-config, but will search in the cget directory for pkg-config files. This useful for finding dependencies when not using cmake.

-p, --prefix PATH
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

-v, --verbose
Enable verbose mode.

remove

This will remove a package. If other packages depends on the package to be removed, those packages will be removed as well.

<package-name>
This is the name of the package to be removed.

-p, --prefix PATH
Set prefix where packages are installed. This defaults to a directory named `cget` in the current working directory. This can also be overidden by the `CGET_PREFIX` environment variable.

-v, --verbose
Enable verbose mode.

-y, --yes
Affirm all questions.

CHAPTER 5

Using cget

Installing cmake packages

When package is installed from one of the package sources(see *Package source*) using the install command, `cget` will run the equivalent cmake commands to install it:

```
mkdir build
cd build
cmake -DCMAKE_TOOLCHAIN_FILE=$CGET_PREFIX/cget/cget.cmake -DCMAKE_INSTALL_PREFIX=$CGET_PREFIX ..
cmake --build .
cmake --build . --target install
```

However, `cget` will always create the build directory out of source. The `cget.cmake` is a toolchain file that is setup by `cget`, so that cmake can find the installed packages. Other settings can be added about the toolchain as well(see init).

The `cget.cmake` toolchain file can be useful for cmake projects to use. This will enable cmake to find the dependencies installed by `cget` as well:

```
cmake -DCMAKE_TOOLCHAIN_FILE=$CGET_PREFIX/cget/cget.cmake ..
```

Instead of passing in the toolchain, `cget` provides a build command to take of this already(see build). This will configure cmake with `cget.cmake` toolchain file and build the project:

```
cget build
```

By default, it will build the `all` target, but a target can be specified as well:

```
cget build --target some_target
```

For projects that don't use cmake, then its matter of searching for the dependencies in `CGET_PREFIX`. Also, it is quite common for packages to provide `pkg-config` files for managing dependencies. So, `cget` provides a `pkg-config` command that will search for the dependencies that `cget` has installed. For example, `cget pkg-config` can be used to link in the dependencies for zlib without needing cmake:

```
cget install zlib,http://zlib.net/zlib-1.2.8.tar.gz
g++ src.cpp `cget pkg-config zlib --cflags --libs`
```

Installing non-cmake packages

Using custom cmake

For packages that don't support building with cmake. A cmake file can be provided to build the package. This can either build the sources or bootstrap the build system for the package:

```
cget install SomePackage --cmake mycmake.cmake
```

Header-only libraries

For libraries that are header-only, `cget` provides a cmake file `header` to install the headers. For example, Boost.Preprocessor library can be installed like this:

```
cget install boostorg/preprocessor --cmake header
```

By default, it installs the headers in the 'include' directory, but this can be changed by setting the `INCLUDE_DIR` cmake variable:

```
cget install boostorg/preprocessor --cmake header -DINCLUDE_DIR=include
```

Binaries

For binaries, `cget` provides a cmake file `binary` which will install all the files in the package without building any source files. For example, the clang binaries for ubuntu can be installed like this:

```
cget install clang,http://llvm.org/releases/3.9.0/clang+llvm-3.9.0-x86_64-linux-gnu-ubuntu-16.04.tar.
```

Boost

A cmake `boost` is provided to install boost libraries as well:

```
cget install boost,http://downloads.sourceforge.net/project/boost/boost/1.62.0/boost_1_62_0.tar.bz2 -
```

Libraries can be selected with cmake variables `BOOST_WITH_` and `BOOST_WITHOUT_`. For example, just Boost.Filesystem(and it dependencies) can be built as:

```
cget install boost,http://downloads.sourceforge.net/project/boost/boost/1.62.0/boost_1_62_0.tar.bz2 -
```

Also, everthing cam be built except Boost.Python like the following:

```
cget install boost,http://downloads.sourceforge.net/project/boost/boost/1.62.0/boost_1_62_0.tar.bz2 -
```

CHAPTER 6

Using recipes

Many times a package doesn't list its dependencies in a requirements.txt file, or it requires special defines or custom cmake(see *Using custom cmake*). A recipe helps simplify this, by allowing a package to be installed with a simple recipe name without needing to update the original package source.

Structure of a recipe

A recipe is a directoy which contains a 'package.txt' file and an optional 'requirements.txt' file. Both files follow the format describe in Requirements file. The 'package.txt' file list only one package, which is the package to be installed. The 'requirements.txt' list packages to be installed as dependecies, which can also reference other recipes.

All recipe directories are searched under the `$CGET_PREFIX/etc/cget/recipes/` directory. A cmake package can install additional recipes through cget.

For example, we could build a simple recipe for zlib so we don't have to remember the url everytime. By adding the file `$CGET_PREFIX/etc/cget/recipes/zlib/package.txt` with the url like this:

```
http://zlib.net/zlib-1.2.8.tar.gz
```

We can now install zlib with just `cget install zlib`. Additionally, we can set additional options as well. For example, if we want to install boost, we can write `$CGET_PREFIX/etc/cget/recipes/boost/package.txt` to use the boost cmake(see *Boost*):

```
http://downloads.sourceforge.net/project/boost/boost/1.62.0/boost_1_62_0.tar.bz2 --cmake boost
```

We can also make zlib a dependency of boost by writing a `$CGET_PREFIX/etc/cget/recipes/boost/requirements.txt` file listing zlib:

```
zlib
```

So, now we can easily install boost with `cget install boost` and it will install zlib automatically as well.

Getting recipes

The cget-recipes repository maintains a set of recipes for many packages. It can be easily installed with:

```
cget install pfultz2/cget-recipes
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index