np

np is a build system for C++. It focuses on ease-of-use, performance, and optional basic integration with the Visual Studio Code IDE.

Designed for use with GCC on any user-oriented Linux distribution.

np stands for "New Project" since this script was intended to be very simple, small, and only for my own niche use cases, but I decided to expand it and make it useful for others as well.

Feature Overview

- Easy project management (sources, headers, output directories)
    - Automatic `.gitignore` editing support
- Multithreaded compilation (one thread for every source file)
- Doesn't rebuild files that haven't changed since last build (if configured correctly; full rebuild mechanism available) aka Minimal Recompile
- Easy external dependency management
    - For example if your project depends on Boost, then the script will try to see if it is already installed correctly. If not then it will automatically download a source tarball from the official Boost website and build it from scratch on the spot.
- Tested and works on any standard user-oriented arm64 or x86-64 Linux distribution with necessary dependencies (other architectures possible but not guaranteed)

Dependencies

It's OK if the version numbers are a little off from what you have, but if the script fails or produces incorrect output then get the specific version mentioned below and try again.

- Basic text manipulation tools that are built-in by default into many distros
    - `sed`
- `jq`
- `sha256sum`

If I were you and I was *somehow* working in an environment where I didn't have these *basic* tools available *at all* in any form or fashion and couldn't install the binaries in any way, **I would seriously rethink my life choices.**

Everything in this repository is public domain, but I'd appreciate it if you'd give credit to me for my scripts :)

Program Flow

Take inputs

- Read `np.options.json`. Skip the configuration screens whose values were already written in the config file. (This makes it a little less repetitive for, e.g., people who always name the first source file `main.cpp` or always use C++14 for everything.)
- Name of the project (a computer-friendly string)
  - Defaults to `cpp_project`
  - If any characters other than the American English alphabet, numbers, and underscore (_) are detected, the script will ask the user to enter a new name.
  - Cannot start with a number or uppercase letter
- Root directory of the new project (defaults to `np/` relative to the current working directory)
- Main source file name (defaults to `main.cpp`)
- C++ standard version (defaults to C++17)
- Compiler specifics *(repeatable for multiple architectures, e.g. one config for arm64 and another config for x86_64)*
  - Include directories (optional, default compiler includes always used even if this field is empty)
  - Library directories (optional, default compiler includes always used even if this field is empty)
  - Compiler executable name prefix & postfixes (for people with strange custom versions of GCC installed; also show a preview of what executable names would look like and if they're found in $PATH)
- Ask if the user wants Visual Studio Code integration (this determines if the script needs to attempt to modify Code's JSON configuration files or not)
  - If the user selects "Yes" but doesn't specify a path to the `.vscode` folder, the script will assume it is a subdirectory of the current working directory.
  - If no `.vscode` folder exists, one will be created automatically.

Create the project

- Create the root directory of the project at where it's supposed to be. If this fails (e.g. no permissions) then abort. If it succeeds, make absolutely sure the script has full R/W access to the directory.
- Create project subdirectories (`bin`, `obj`, `src`)
- Copy template source file into `src`
- Copy the `np` script itself into the project's root directory.
- Create a `.np` subdirectory of the root to store build system private files.
- Write all of the inputs from the last section to a JSON file (for the script's future reference if the project's information needs to be modified)
- Run the `np` script which was just copied and pass argument `--generate` (`np --generate`) with the working directory inside the project's root folder
  - This will generate the build script for the project. Note that the build script does not stay the same throughout the project's development; when a new source file is added, for example, it needs to be regenerated after editing `np.project.json`. See "Project Configuration" on Page 3.
- If the user has specified that they want VS Code integration, then modify `tasks.json` and `launch.json` accordingly.
  - This adds Code configuration items that enable the user to build and run their project with a push of a button from VS Code's 'Run & Debug' UI.
  - It also makes the debugging experience a lot more interactive and easier since VS Code handles that. All we need to do is modify the JSON configuration files (`tasks.json`, `launch.json`) properly.

Compilation process

The build script needs to be regenerated every time the number of available threads for compilation changes. This is just due to the way that multithreaded compilation scheduling works.

- Find out how many threads we have available.
    - Defaults to the total number of threads on the system, but it can be configured to a lower number manually.