NAME

neopo − A lightweight solution for local Particle development

SYNOPSIS

neopo [COMMAND] [PROJECT] [OPTIONS] [-v/q]
neopo create <project> [platform] [version]
neopo configure <platform> <version> [project]
neopo build [project] [-v/q]
neopo particle [OPTIONS]
neopo script [file]

DESCRIPTION

Neopo is a Particle development management utility that simplifies the installation and usage of Particle's toolchains on a variety of distributions. It features options to build or flash projects, iterable commands, a scripting interface, and Particle Workbench/CLI compatibility. On Linux, several packages required for Workbench to operate successfully are installed as dependencies for neopo, providing incredible value to Workbench users, even if they do not choose to use neopo's interfaces.

Neopo was designed with extensibility and portability in mind in an attempt to create a useful and unobtrusive local Particle development environment. While neopo excels at installing Particle toolchains and building Particle projects with its simple but powerful command-line options, it additionally provides shortcuts for commonly performed tasks and two scripting interfaces.

Users can create simple scripts to automate testing and deployment of firmware and execute them with **neopo script** or leverage neopo's Python API to integrate local Particle tools into existing Python solutions.

COMMANDS

When using neopo from the command line, options must be used in the exact order specified in the documentation, much like Particle CLI or Git. Arguments denoted in square brackets **[]** are optional, while arguments denoted in angle brackets <> are mandatory.

GENERAL COMMANDS

help When neopo runs with **help,** or without arguments, brief documentation for all user-accessible commands is displayed.

install [-f, -s]
Install Particle toolchains and setup the neopo environment. It takes an optional parameter of **-f** which can force toolchains to be reinstalled, rather than skipped if the latest versions are already installed. To skip the installation of all dependencies, use the **-s** parameter instead.

versions
List all known versions of Device OS and supported platforms available to neopo. To refresh, run **update.** Custom Device OS versions and platforms are included.

update Refresh the Workbench cache and download Particle toolchains if there are newer versions available.

get <version>
Download a specific release of Device OS version for later use. Any dependencies of the Device OS release will be downloaded as well.

remove <version>
Delete an installed Device OS release if possible. Useful for systems with limited storage space.

particle [OPTIONS]
Access the Particle CLI distribution used internally by neopo. By using the **neopo particle** prefix, any Particle CLI command can be accessed. For convenience, a **particle** alias is included with neopo.

BUILD COMMANDS

compile/build [project] [-v/q]
Compile the application firmware of a given Particle project, or the current directory if it's a project. Settings applied using **configure** will be passed on to the compiler. The verbosity of the output can be increased with the -v flag, or decreased with the -q flag.

flash [project] [-v/q]
Compile application firmware and flash to a connected device using DFU. On Linux the udev rules file required for non-root access to Particle devices over USB can be installed using: **neopo particle usb configure**

flash-all [project] [-v/q]
Compile application and system firmware and flash all parts to a connected device using DFU. Incredibly useful when an application targets a newer release of Device OS as it eliminates the need for the device to download the release from the cloud.

clean [project] [-v/q]
Clean application firmware. Usually unnecessary but can eliminate some build errors.

PROJECT COMMANDS

create <project> [platform] [version]
Create a new Particle project at the specified project path. The created project is compatible with neopo, Particle Workbench, and Particle CLI. If Git is installed, the project is additionally initialized as a repository with support for TravisCI.

configure <platform> <version> [project]
Configure the device platform and Device OS version a project should use. The platform and version are checked for compatibility and the specified version of Device OS will be downloaded if not already installed.

To find Device OS versions and their supported platforms use **versions.** The tab completion function can fill in platforms and versions for this command.

run <target> [project] [-v/q]
Run a specified makefile target for a project. Includes common targets presented in **BUILD COMMANDS** in addition to other, less frequently used targets. The -v and -q flags are supported. Running without arguments will list available targets. The tab completion function can suggest targets for this command.

export <target> [project] [-v/q]
Export a makefile target to a shell script. Instead of running the target on the project, a shell script is created at *bin/neopo-<target>.sh* within the project. The script may be used to run the target on the project directly without using neopo.

flags <string> [project]
Set the EXTRA_CFLAGS variable to be used during compilation of a project. Useful for passing additional definitions to the preprocessor.

settings [project]
View configured settings for a project. The device platform, Device OS version, and EXTRA_CFLAGS will be printed.

libs [project]
Verify or install Particle libraries specified in **project.properties** for a project. This command is useful when working with projects that use the cloud compiler because it allows you to quickly download the same libraries locally.

SPECIAL COMMANDS

bootloader <platform> <version> [-v/-q]
An experimental command to build and flash the bootloader for a specified platform and Device OS version. After building the bootloader it is flashed over serial using particle-cli.

iterate <command> [OPTIONS] [-v/q]
An advanced command used to run an iterable command for all connected devices. For each connected device, the deviceID is printed, the device is put into DFU mode, and the specified iterable command is executed. This command was originally designed for quickly flashing multiple connected devices, but there are many ways it can be used.

The following commands are iterable: **compile, build, flash, flash-all, clean, run, script.**

SCRIPT INTERFACE

One of the powerful features of neopo is the scripting interface. Neopo scripts are a list of commands to run sequentially, with each command placed on its own line. Empty lines and lines starting with **#** are skipped. Any neopo command can be used in a neopo script, **even Particle commands.** F or sophisticated scripts the Python module should be used instead.

script [file]
Execute a script with neopo. If a filename is not provided, neopo will accept a script piped in from standard input.

```
$ neopo script myFile
$ cat myFile | neopo script
```

To relay information to the user, the **print** command can be used, and to wait for user interaction or acknowledgement, the **wait** command can be used.

Here is an example neopo script:

```
# Configure the current project
configure argon 1.5.2

# Prompt the user to plug in a device
print "Please plug in your device."
wait
```

```
# Flash firmware to the device
flash

# Prompt the user to wait for the device to connect
print "Please wait for your device to connect to the cloud."
wait

# Subscribe to incoming messages
particle subscribe
```

PYTHON INTERFACE

Neopo is distributed as a Python module. After installation, not only will neopo be available as a command-line program, but it will additionally be accessible within Python. Users are encouraged to experiment with neopo in Python scripts or the REPL.

Here is the script example implemented in Python:

```
import neopo
neopo.configure("argon", "1.5.2", "myProject")
print("Please plug in your device.")
neopo.script_wait()
neopo.flash("myProject")

print("Please wait for your device to connect to the cloud.")
neopo.script_wait()
neopo.particle("subscribe")
```

To directly use Particle CLI within Python, one can explicitly import the particle() function:

```
from neopo import particle
particle("help")
particle("serial monitor")

device = "myFooMachine"
function = "myBarFunction"
particle(["call", device, function])
```

CONFIGURATION

There are several environment variables that may be exported or passed to neopo to alter its configuration. More environment variables may be added as neopo becomes more extensible.

NEOPO_LOCAL

When set, neopo will use*~/.local/shar e/neopo* for neopo resources, Particle CLI, and toolchains, rather than*~/.particle* and *~/.neopo*

```
$ export NEOPO_LOCAL=1
$ neopo install
$ neopo particle
```

NEOPO_PATH

When set, neopo will use a specific path for its dependencies, rather than*~/.local/shar e/neopo*

$ NEOPO_PATH="build/neopo" neopo install

NEOPO_PARALLEL
When set, neopo will download dependencies in parallel, slightly improving overall download time.

$ NEOPO_PARALLEL=1 neopo install

AUTHOR
Nathan Robinson <nrobinson2000@me.com>

COPYRIGHT

REPORTING BUGS
nrobinson2000/neopo on GitHub: <https://github.com/nrobinson2000/neopo>

SEE ALSO
Online Documentation: <https://neopo.xyz/docs/full-docs>
Particle Developer Forum: <https://community.particle.io>
Workbench Documentation: <https://docs.particle.io/workbench>
Particle CLI Documentation: <https://docs.particle.io/reference/developer-tools/cli>

NOTES
On Arch-based distributions using the **neopo-git** package from the AUR is recommended.

There are several additional steps required to complete the install of neopo. These steps are contained in a script located at /usr/share/neopo/scripts/POSTINSTALL. For convenience, this script can be executed using: **neopo setup**

On x86_64, this consists of installing the ncurses package from the AUR to support use of the Particle Debugger in Workbench. On aarch64, this consists of replacing the armv7l Nodejs distribution with an aarch64 Nodejs distribution.

Using neopo on armv7l and aarch64 is incredibly feasible. On average, builds run only a few times slower than on Linux x86_64 systems, which is still much faster than using Particle Workbench on Windows. Hopefully Particle will differentiate between armv7l and aarch64 in future releases so that using Particle on aarch64 will become more accessible.

Using Particle Workbench on aarch64 is possible through the use of neopo. After installing *visual-studio-code-bin* from the AUR, the Workbench extensions can be installed and prepared with: **neopo setup-workbench**