Clickable Documentation

Release 6.4.0

Brian Douglass

Nov 07, 2019

Contents

Build and compile Ubuntu Touch apps easily from the command line. Deploy your apps to your Ubuntu Touch device for testing or test them on any desktop Linux distribution. Get logs for debugging and directly access a terminal on your device.

Clickable is fully Open Source and can be found on GitLab. Clickable is developed by Brian Douglass and Jonatan Hatakeyama Zeidler with a huge thank you to all the contributors.

CHAPTER 1

Using Clickable

1.1 Install

1.1.1 Install Via Pip (Recommended)

- Install docker, adb, git, and pip3
- Run (may need sudo): `pip3 install git+https://gitlab.com/clickable/clickable.git`

1.1.2 Install Via PPA (Ubuntu)

- Add the PPA to your system: `sudo add-apt-repository ppa:bhdouglass/clickable`
- Update your package list: `sudo apt-get update`
- Install clickable: `sudo apt-get install clickable`

1.1.3 Install Via AUR (Arch Linux)

- Using your favorite AUR helper, install the clickable-git package
- Example: `pacaur -S clickable-git`

1.2 Getting Started

- Run `clickable create` to get started with a new app.
- Choose from the list of *app templates*.
- Provide all the needed information about your new app.
- When the app has finished generating, enter the newly created directory containing your app.

- Run `clickable` to compile your app and install it on your phone.

1.2.1 Getting Logs

To get logs from you app simply run *clickable logs*. This will give you output from C++ (`QDebug() << "message"`) or from QML (`console.log("message")`) in addition to any errors or warnings.

1.2.2 Running on the Desktop

Running the app on the desktop just requires you to run `clickable desktop`. This is not as complete as running the app on your phone, but it can help speed up development.

1.2.3 Accessing Your Device

If you need to access a terminal on your Ubuntu Touch device you can use `clickable shell` to open up a terminal to your device from your computer. This is a replacement for the old `phablet-shell` command.

1.2.4 Ubuntu Touch SDK Api Docs

For more information about the Ubuntu Touch QML or HTML SDK check out the docs over at UBports.

1.2.5 Run Automatic Review

Apps submitted to the OpenStore will undergo automatic review, to test your app before submitting it, run `clickable review` after you've compiled a click. This runs the `click-review` command against your click within the clickable container (no need to install it on your computer).

1.2.6 Handling dependencies

For more information about compiling, using and deploying app dependencies, check out the docs over at UBports.

1.2.7 Publishing to the OpenStore

If this is your first time publishing to the OpenStore, you need to signup for an account. You can signup with your GitHub, GitLab, or Ubuntu account.

If your app is new to the OpenStore you must first create your app by entering the name from your manifest.json and the app's title on the OpenStore's submission page.

If your app already exists you can use the `clickable publish` command to upload your compiled click file to the OpenStore. In order to publish to the OpenStore you need to grab your api key from the OpenStore. After you have your api key you need to let Clickable know about it. You can either pass it as an argument every time: `clickable publish --apikey XYZ` Or you can set it as an environment variable: `export OPENSTORE_API_KEY=XYZ` (you can add this to your `~/.bashrc` to keep it set).

1.3 Usage

1.3.1 Getting Started

You can get started with using clickable with an existing Ubuntu Touch app. You can use clickable with apps generated from the old Ubuntu Touch SDK IDE or you can start fresh by running `clickable create`.

To run the default set of sub-commands, simply run `clickable` in the root directory of your app's code. Clickable will attempt to auto detect the *build template* and other configuration options.

Running the default sub-commands will:

1) Clean the build directory (by default the build directory is `./build/`)
2) Compile the app
3) Build the click package (can be found in the build directory)
4) Install the app on your phone (By default this uses adb, see below if you want to use ssh)
5) Kill the running app on the phone
6) Launch the app on your phone

1.3.2 Configuration

If you need more advanced usage options, you may specify a configuration file in the *clickable.json format* with `--config`. If not specified, clickable will look for an optional configuration file called `clickable.json` in the current directory.

1.3.3 Connecting to a device over ssh

By default the device is connected to via adb. If you want to access a device over ssh you need to either specify the device IP address or hostname on the command line (ex: `clickable logs --ssh 192.168.1.10`) or you can use the `CLICKABLE_SSH` env var.

1.3.4 Multiple connected devices

By default clickable assumes that there is only one device connected to your computer via adb. If you have multiple devices attached to your computer you can specify which device to install/launch/etc on by using the flag `--serial-number` or `-s` for short. You can get the serial number by running `clickable devices`.

1.3.5 Debugging

GDB Debugging via Clickable is only available in desktop mode and can be started by running `clickable desktop --gdb`.

Alternatively a GDB Server can be started with `clickable desktop --gdbserver <port>` (just choose any port, e.g. `3333`). Check for an option to do GDB Remote Debugging in your IDE and connect to `localhost:<port>`. To connect a GDB Client run `gdb <app-binary> -ex 'target remote localhost:<port>'`.

1.4 Commands

From the root directory of your project you have the following sub-commands available:

You can combine the commands together like `clickable build install launch`

1.4.1 clickable

Runs the default sub-commands specified in the "default" config. A dirty build without cleaning the build dir can be achieved by running `clickable --dirty`.

1.4.2 clickable desktop

Compile and run the app on the desktop.

Note: ArchLinux user might need to run `xhost +local:clickable` before using desktop mode.

Run `clickable desktop --verbose` to show the executed docker command.

Run `clickable desktop --dark-mode` to set the dark mode preference.

Run `clickable desktop --lang <language code>` to test using a different language.

1.4.3 clickable desktop --nvidia

`clickable` checks automatically if nvidia-drivers are installed and turns on nvidia mode. The `--nvidia` flag lets you manually enforce nvidia mode.

Depending on your docker version, the docker execution will change and you need to provide additional system requirements:

docker < 19.03 system requirements

- nvidia-modprobe
- nvidia-docker

On Ubuntu, install these requirements using `apt install nvidia-modprobe nvidia-docker`.

docker >= 19.03 system requirements

- nvidia-container-toolkit

On Ubuntu, install these requirements using `apt install nvidia-container-toolkit`.

Run clickable with the `--verbose` flag to see the executed command for your system.

1.4.4 clickable create

Generate a new app from a list of *app template options*.

`clickable create <app template name>`

Generate a new app from an *app template* by name.

1.4.5 clickable shell

Opens a shell on the device via ssh. This is similar to the `phablet-shell` command.

1.4.6 clickable clean-libs

Cleans out all library build dirs.

1.4.7 clickable build-libs

Builds the dependency libraries specified in the clickable.json.

1.4.8 clickable clean

Cleans out the build dir.

1.4.9 clickable build

Builds the project using the specified template, build dir, and build commands. Then it takes the built files and compiles them into a click package (you can find it in the build dir).

1.4.10 clickable build --output=/path/to/some/diretory

Takes the built files and compiles them into a click package, outputting the compiled click to the directory specified by `--output`.

1.4.11 clickable review

Takes the built click package and runs click-review against it. This allows you to review your click without installing click-review on your computer.

1.4.12 clickable test

Run your test suite in with a virtual screen. By default this runs qmltestrunner, but you can specify a custom command by setting the *test* property in your clickable.json.

1.4.13 clickable install

Takes a built click package and installs it on a device.

```
clickable install ./path/to/click/app.click
```

Installs the specified click package on the device

1.4.14 clickable launch

Launches the app on a device.

```
clickable launch <app name>
```

Launches the specified app on a device.

1.4.15 clickable logs

Follow the apps log file on the device.

1.4.16 clickable log

Dumps the apps log file on the device.

1.4.17 clickable publish

Publish your click app to the OpenStore. Check the *Getting started doc* for more info.

```
clickable publish "changelog message"
```

Publish your click app to the OpenStore with a message to add to the changelog.

1.4.18 clickable run "some command"

Runs an arbitrary command in the clickable container.

1.4.19 clickable update

Update the docker container for use with clickable.

1.4.20 clickable no-lock

Turns off the device's display timeout.

1.4.21 clickable writable-image

Make your Ubuntu Touch device's rootfs writable. This replaces to old `phablet-config writable-image` command.

1.4.22 clickable devices

Lists the serial numbers and model names for attached devices. Useful when multiple devices are attached and you need to know what to use for the `-s` argument.

1.4.23 clickable <custom command>

Runs a custom command specified in the "scripts" config

1.4.24 clickable <any command> --container-mode

Runs all builds commands on the current machine and not in a container. This is useful from running clickable from within a container.

1.4.25 clickable <any command> --verbose

Have Clickable print out debug information about whatever command(s) are being run.

1.5 clickable.json Format

Example:

```
{
    "template": "cmake",
    "scripts": {
        "fetch": "git submodule update --init"
    },
    "dependencies_target": [
        "libpoppler-qt5-dev"
    ]
}
```

1.5.1 Placeholders & Environment Variables

The following placeholders can be used in the clickable.json. They are also provided as environment variables during build. When passing `--debug` to Clickable, `DEBUG_BUILD=1` is set as an environment variable, additionally.

Placeholder	Output
$ARCH_TRIPLET	Target architecture triplet (`arm-linux-gnueabihf`, `aarch64-linux-gnu`, `x86_64-linux-gnu` or `all`)
$ROOT	Value of `root_dir`
$BUILD_DIR	Value of `build_dir`
$SRC_DIR	Value of `src_dir`
$INSTALL_DIR	Value of `install_dir`
$CLICK_LD_LIBRARY_PATH	`$INSTALL_DIR/lib/$ARCH_TRIPLET` or `$INSTALL_DIR/lib` for architecture independent apps (will be in `LD_LIBRARY_PATH` on runtime)
$CLICK_QML2_IMPORT_PATH	`$INSTALL_DIR/lib/$ARCH_TRIPLET` or `$INSTALL_DIR/qml` for architecture independent apps, which is not in `QML2_IMPORT_PATH` at runtime (otherwise will be in `QML2_IMPORT_PATH` on runtime)
$CLICK_PATH	`$INSTALL_DIR/lib/$ARCH_TRIPLET/bin` or `$INSTALL_DIR` for architecture independent apps (will be in `PATH` on runtime)
$<lib>_LIB_INSTALL_DIR	Value of `install_dir` from `<lib>` (see *libraries*)

Parameters accepting placeholders: `root_dir`, `build_dir`, `src_dir`, `install_dir`, `gopath`, `cargo_home`, `scripts`, `build`, `build_args`, `make_args`, `postmake`, `postbuild`, `prebuild`, `install_lib`, `install_bin`, `install_qml`, `install_data`. This is an ordered list. Parameters that are used as placeholders themselfs accept only predecessors. Ex: `$ROOT` can be used in `src_dir`, but not vice-versa.

Example:

```
{
    "template": "cmake",
    "build_dir": "$ROOT/build/$ARCH_TRIPLET/myApp"
}
```

1.5.2 clickable_minimum_required

Optional, a minimum Clickable version number required to build the project. Ex: `"6"` or `"5.4.0"`

1.5.3 arch

Optional, the default is armhf. You may better specify this as a cli arg (ex: `--arch arm64`)

1.5.4 template

Optional, see *build template* for the full list of options. If left blank the template will be auto detected.

1.5.5 prebuild

Optional, a custom command to run from the root dir, before a build.

1.5.6 build

Optional, a custom command to run from the build dir. This only takes effect if using the `custom` template. It's even required in that case.

1.5.7 postmake

Optional, a custom command to execute from the build directory, after make (during build).

1.5.8 postbuild

Optional, a custom command to execute from the build dir, after build and before click packaging.

1.5.9 build_args

Optional, arguments to pass to qmake or cmake. When using `--debug`, `CONFIG+=debug` is additionally appended for qmake and `-DCMAKE_BUILD_TYPE=Debug` for cmake and cordova builds. Ex: `CONFIG+=ubuntu`

Can be specified as a string or a list of strings.

1.5.10 make_args

Optional, arguments to pass to make, e.g. a target name. To avoid configuration conflicts, the number of make jobs should not be specified here, but using `make_jobs` instead, so it can be overriden by the according environment variable.

Can be specified as a string or a list of strings.

1.5.11 make_jobs

Optional, the number of jobs to use when running make, equivalent to make's `-j` option. If left blank this defaults to the number of CPU cores.

1.5.12 launch

Optional, a custom command to launch the app, used by `clickable launch`.

1.5.13 build_dir

Optional, a custom build directory. Defaults to `$ROOT/build/$ARCH_TRIPLET/app`. Thanks to the architecture triplet, builds for different architectures can exist in parallel.

1.5.14 src_dir

Optional, a custom source directory. Defaults to `$ROOT`

1.5.15 install_dir

Optional, a custom install directory (used to gather data that goes into the click package). Defaults to `$BUILD_DIR/install`

1.5.16 install_lib

Optional, additional libraries that should be installed with the app and be in `LD_LIBRARY_PATH` at runtime. The destination directory is `$CLICK_LD_LIBRARY_PATH`. Ex:

```
"install_lib": [
    "/usr/lib/$ARCH_TRIPLET/libasound.so*"
]
```

Can be specified as a string or a list of strings. Supports wildcards as this actually calls `cp -r <path> $CLICK_LD_LIBRARY_PATH` in a bash.

1.5.17 install_qml

Optional, additional QML files or directories that should be installed with the app and be in `QML2_IMPORT_PATH` at runtime. The destination directory is `$CLICK_QML2_IMPORT_PATH`. Ex:

```
"install_qml": [
    "/usr/lib/$ARCH_TRIPLET/qt5/qml/QtGraphicalEffects"
]
```

Can be specified as a string or a list of strings. Supports wildcards as this actually calls `cp -r <path> $CLICK_QML2_IMPORT_PATH` in a bash.

1.5.18 install_bin

Optional, additional executables that should be installed with the app and be in `PATH` at runtime. The destination directory is `$CLICK_PATH`. Ex:

```
"install_bin": [
    "/usr/bin/htop"
]
```

Can be specified as a string or a list of strings. Supports wildcards as this actually calls `cp -r <path> $CLICK_PATH` in a bash.

1.5.19 install_data

Optional, additional files or directories that should be installed with the app. Needs to be specified as a dictionary with absolute source paths as keys and destinations as value. Ex:

```
"install_data": {
    "$ROOT/packaging/manifest.json": "$INSTALL_DIR",
    "$ROOT/packaging/myapp.desktop": "$INSTALL_DIR"
},
```

Supports wildcards as this actually calls `cp -r <src> <dst>` in a bash.

1.5.20 kill

Optional, a custom process name to kill (used by `clickable launch` to kill the app before relaunching it). If left blank the process name will be assumed.

1.5.21 scripts

Optional, an object detailing custom commands to run. For example:

```
"scripts": {
    "fetch": "git submodule update --init",
    "echo": "echo $ARCH_TRIPLET"
}
```

That enables the use of `clickable fetch` and `clickable echo`.

1.5.22 default

Optional, sub-commands to run when no sub-commands are specified (running simply `clickable`). Defaults to `clean build install launch`. The `--dirty` cli argument removes `clean` from that list.

Can be specified as a string or a list of strings.

1.5.23 dirty

Optional, whether or not do a dirty build, avoiding to clean the build directory before building. You may also specify this as a cli arg (`--dirty`). The default is `false`.

1.5.24 dependencies_build

Optional, a list of dependencies that will be installed in the build container.

Add dependencies here that are part of your build tool chain.

Can be specified as a string or a list of strings.

1.5.25 dependencies_target

Optional, a list of dependencies that will be installed in the build container. These will be assumed to be `dependency:arch` (where `arch` is the target architecture), unless an architecture specifier is already appended.

Add dependencies here that your app actually depends on.

Can be specified as a string or a list of strings.

1.5.26 dependencies_ppa

Optional, a list of PPAs, that will be enabled in the build container. Ex:

```
"dependencies_ppa": [
    "ppa:bhdouglass/clickable"
]
```

Can be specified as a string or a list of strings.

1.5.27 docker_image

Optional, the name of a docker image to use. When building a custom docker image it's recommended to use one of the Clickable images as a base. You can find them on Docker Hub.

1.5.28 ignore

Optional, a list of files to ignore when building a `pure` template Example:

```
"ignore": [
    ".clickable",
    ".git",
    ".gitignore",
    ".gitmodules"
]
```

Can be specified as a string or a list of strings.

1.5.29 gopath

Optional, the gopath on the host machine. If left blank, the `GOPATH` env var will be used.

1.5.30 cargo_home

Optional, the Cargo home path on the host machine that is used for caching (namely its subdirs `registry`, `git` and `.package-cache`). Defaults to `~/.clickable/cargo`.

1.5.31 root_dir

Optional, specify a different root directory for the project. For example, if you clickable.json file is in `platforms/ubuntu_touch` and you want to include code from root of your project you can set `root_dir: "../.."`. Alternatively you can run clickable from the project root in that case via `clickable -c platforms/ubuntu_touch/clickable.json`.

1.5.32 test

Optional, specify a test command to be executed when running `clickable test`. The default is `qmltestrunner`.

1.5.33 libraries

Optional, dependencies to be build by running `clickable build-libs`. It's a dictionary of dictionaries similar to the clickable.json itself. Example:

```
"libraries": {
    "opencv": {
        "template": "cmake",
        "make_jobs": "4",
        "build_args": [
            "-DCMAKE_BUILD_TYPE=Release",
            "-DBUILD_LIST=core,imgproc,highgui,imgcodecs",
            "-DBUILD_SHARED_LIBS=OFF"
        ]
        "prebuild": "git submodule update --init --recursive"
    }
}
```

The keywords `install_dir`, `prebuild`, `build`, `postbuild`, `postmake`, `make_jobs`, `make_args`, `build_args`, `docker_image`, `dependencies_build`, `dependencies_target` and `dependencies_ppa`, can be used for a library the same way as described above for the app.

In addition to the *placeholders* described above, the following placeholders are available:

Placeholder	Output
$NAME	The library name (key name in the `libraries` dictionary)

A single library can be build by specifying its name as `clickable build-libs lib1 --arch arm64` to build the library with name `lib1` for the architecture `arm64`. `clickable clean-libs lib1 --arch arm64` cleans the libraries build dir.

template

Required, but only `cmake`, `qmake` and `custom` are allowed.

src_dir

Optional, library source directory. Must be relative to the project root. Defaults to `$ROOT/libs/$NAME`

build_dir

Optional, library build directory. Must be relative to the project root. Defaults to `$ROOT/build/$ARCH_TRIPLET/$NAME`. Thanks to the architecture triplet, builds for different architectures can exist in parallel.

1.5.34 Removed keywords

The following keywords are no longer supported:

- `dependencies` (use `dependencies_target` and `dependencies_build` instead)
- `specificDependencies`
- `dir` (use `build_dir` instead)
- `lxd`
- `premake` (use `prebuild`, `postmake` or `postbuild` instead)
- `ssh` (use program option `--ssh` or environment variable `CLICKABLE_SSH` instead)
- `chroot`
- `sdk`
- `package`
- `app`

1.6 Environment Variables

Environment variables will override values in the clickable.json and can be overridden by command line arguments.

In contrast to the environment variables described here that configure Clickable, there are *environment variables* set by Clickable to be used during build.

1.6.1 CLICKABLE_ARCH

Overrides the clickable.json's *arch*.

1.6.2 CLICKABLE_TEMPLATE

Overrides the clickable.json's *template*.

1.6.3 CLICKABLE_BUILD_DIR

Overrides the clickable.json's *dir*.

1.6.4 CLICKABLE_DEFAULT

Overrides the clickable.json's *default*.

1.6.5 CLICKABLE_MAKE_JOBS

Overrides the clickable.json's *make_jobs*.

1.6.6 GOPATH

Overrides the clickable.json's *gopath*.

1.6.7 CARGO_HOME

Overrides the clickable.json's *cargo_home*.

1.6.8 CLICKABLE_DOCKER_IMAGE

Overrides the clickable.json's *docker_image*.

1.6.9 CLICKABLE_BUILD_ARGS

Overrides the clickable.json's *build_args*.

1.6.10 CLICKABLE_MAKE_ARGS

Overrides the clickable.json's *make_args*.

1.6.11 OPENSTORE_API_KEY

Your api key for *publishing to the OpenStore*.

1.6.12 CLICKABLE_CONTAINER_MODE

Same as *–container-mode*.

1.6.13 CLICKABLE_SERIAL_NUMBER

Same as *–serial-number*.

1.6.14 CLICKABLE_SSH

Same as *–ssh*.

1.6.15 CLICKABLE_OUTPUT

Override the output directory for the resulting click file

1.6.16 CLICKABLE_NVIDIA

Same as *–nvidia*.

1.6.17 CLICKABLE_DIRTY

Overrides the clickable.json's *dirty*.

1.6.18 CLICKABLE_DEBUG_BUILD

Same as `--debug`

1.6.19 CLICKABLE_TEST

Overrides the clickable.json's *test*.

1.6.20 CLICKABLE_DARK_MODE

Same as `--dark-mode`

1.7 App Templates

1.7.1 Pure QML App (built using CMake)

An app template that is setup for a purely QML app. It includes a CMake setup to allow for easy translations.

Check it out on GitLab.

1.7.2 C++/QML App (built using CMake)

An app template that is setup for a QML app with a C++ plugin. It includes a CMake setup to allow for easy translation.

Check it out on GitLab.

1.7.3 Python/QML App (built using CMake)

An app template that is setup for an app using Python with QML. It includes a CMake setup to allow for easy translation.

Check it out on GitLab.

1.7.4 HTML App

An app template that is setup for a local HTML app.

Check it out on GitLab.

1.7.5 Simple Webapp

An app template that is setup as a thin wrapper around an existing website.

Check it out on GitLab.

1.7.6 Go/QML App

An app template that is setup for a QML app with a Go backend.

Check it out on GitLab.

1.7.7 C++/QML App (built using CMake with a main.cpp)

An app template that is setup for a QML app with a C++ plugin. It includes a CMake setup to allow for easy translation. It also includes a main.cpp to build a custom binary rather than relying on qmlscene.

Check it out on GitLab.

1.7.8 Rust/QML App

An app template that is setup for a QML app with a Rust backend.

Check it out on GitLab.

1.8 Build Templates

1.8.1 pure-qml-qmake

A purely qml qmake project.

1.8.2 qmake

A project that builds using qmake (has more than just QML).

1.8.3 pure-qml-cmake

A purely qml cmake project

1.8.4 cmake

A project that builds using cmake (has more than just QML)

1.8.5 custom

A custom build command will be used.

1.8.6 cordova

A project that builds using cordova

1.8.7 pure

A project that does not need to be compiled. All files in the project root will be copied into the click.

1.8.8 python

A project that uses python and does not need to be compiled.

1.8.9 go

A project that uses go version 1.6.

1.8.10 rust

A project that uses rust. Debug builds can be enabled by specifying `--debug`.

1.9 Continuous Integration

1.9.1 Clickable CI Docker Images

Two docker images are available for easily using Clickable with a continuous integration setup. They can be found on Docker hub: `clickable/ci-16.04-armhf` and `clickable/ci-16.04-amd64`. The images come with Clickable pre installed and already setup in container mode.

1.9.2 GitLab CI Tutorial

For a full guide to setting up GitLab CI with a Clickable app check out this blog post. This method can also be adapted for other CI solutions.

1.10 Changelog

1.10.1 Changes in v6.4.0

- Use the system timezone when in desktop mode

1.10.2 Changes in v6.3.2

- Fixed issues logging process errors
- Fixed issues parsing desktop files

1.10.3 Changes in v6.3.1

- Updated *clickable create* to use a new template for a better experience
- Fixed desktop mode issue when the command already exits in the PATH
- Added a prompt for autodetecting the template type
- Improved Clickable's logging

1.10.4 Changes in v6.2.1

- Fixed env vars in libs

1.10.5 Changes in v6.2.0

- Replaced the `--debug` argument with `--verbose`
- Switched the `--debug-build` argument to `--debug`
- Initial support for running Clickable on MacOS
- Added new desktop mode argument `--skip-build` to run an app in desktop mode without recompiling

1.10.6 Changes in v6.1.0

- Apps now use host locale in desktop mode
- Added `--lang` argument to override the language when running in desktop mode
- Added support for multimedia in desktop mode
- Make app data, config and cache persistent in desktop mode by mounting phablet home folder to ~/.clickable/home
- Added arm64 support and docker images (does not yet work for apps built with qmake)
- *Added placeholders and env vars to commands and scripts run via clickable*
- *Added option to install libs/qml/binaries from the docker image into the click package*
- Switched to a clickable specific Cargo home for Rust apps
- Click packages are now deleted from the device after installing

- Fixed `dependencies_build` not being allowed as a string
- Fixed issues finding the manifest file

1.10.7 Changes in v6.0.3

- Fixed building go apps
- Fixed post build happening after the click is built

1.10.8 Changes in v6.0.2

- Fixed container mode

1.10.9 Changes in v6.0.1

- Added back click-build with a warning to not break existing apps

1.10.10 Changes in v6.0.0

New features

- When publishing an app for the first time a link to create it on the OpenStore will be shown
- Desktop mode can now use the dark theme with the `--dark-mode` argument
- Automatically detect when nvidia drivers are used for desktop mode
- Use native docker nvidia integration rather than nvidia-docker (when the installed docker version supports it)
- The UBUNTU_APP_LAUNCH_ARCH env var is now set for desktop mode
- Added remote gdb debugging in desktop mode via the `--gdbserver <port>` argument
- Added configurable `install_dir`
- Libraries get installed when using `cmake` or `qmake` build template (into `install_dir`)

Breaking Changes

- The `click-build` command has been merged into the `build` command
- Removed deprecated configuration properties: `dependencies`, `specificDependencies`, and `dir`
- Removed deprecated library configuration format
- Removed deprecated lxd support
- Moved the default build directory from `build` to `build/<arch triplet>/app`
- Moved the default library build directory to `build/<arch triplet>/<lib name>`
- Removed deprecated vivid support

Bug Fixes

- Fixed utf-8 codec error
- Use separate cached containers when building libraries
- Automatically rebuild the cached docker image for dependencies

1.10.11 Changes in v5.14.1

- Limit make processes to the number of cpus on the system
- Fix missing directory for newer Rust versions
- Fix placeholders not being absolute

1.10.12 Changes in v5.14.0

- Added check for outdated containers when using custom dependencies
- Fixed building libraries

1.10.13 Changes in v5.13.3

- Fixed the update command so it updates all available Docker images

1.10.14 Changes in v5.13.2

- Fixed libraries not building after latest update

1.10.15 Changes in v5.13.1

- Follow up fixes for dependencies not being used for the first run

1.10.16 Changes in v5.13.0

- Added new *debugging with gdb* argument
- Added new *test* command for running tests inside the container
- When running in desktop mode, cache/share/config directories are automatically created
- Fixed hidden build directories causing errors when looking for the manifest
- Fixed issue with cordova building
- Fixed dependencies not being used the first time clickable is run

1.10.17 Changes in v5.12.3

- Fixed slowdown when running clickable in a non-project directory

1.10.18 Changes in v5.12.2

- Fixed `scripts` breaking Clickable

1.10.19 Changes in v5.12.1

- Fixed issues with build dir

1.10.20 Changes in v5.12.0

- clickable.json supports *placeholders* now
- Add new `src_dir` configuration option
- Make build-libs respect `root_dir`, too
- Fix build-libs for architecture all
- When no `kill` configuration option is specified Clickable will use the Exec line from the desktop file

1.10.21 Changes in v5.11.0

- Smarter app killing using `pkill -f`
- Fix deprecated configuration options showing as a schema error

1.10.22 Changes in v5.10.0

- Added configuration option `root_dir`
- Always ignore .git/.bzr directories when building pure, rust, or go apps

1.10.23 Changes in v5.9.1

- Fixed missing schema file

1.10.24 Changes in v5.9.0

- New schema validation for clickable.json
- Publish to the OpenStore with a changelog message

1.10.25 Changes in v5.8.1

- Fixed a bug in `make_args`

1.10.26 Changes in v5.8.0

- New configuration option for automatically including ppas in the build environment: *dependencies_ppa*.
- Changed *libraries* format from a list to a dictionary (the old format is still supported for now)
- The default `cargo_home` is now set to `~/.cargo`

1.10.27 Changes in v5.7.0

- Introduced two new dependency options to separate *build* <*clickable-json-dependencies_build*> and *target* <*clickable-json-dependencies_target*> dependencies

1.10.28 Changes in v5.6.1

- Fixed build lib
- Made cordova build respect the –debug-build argument

1.10.29 Changes in v5.6.0

- Fixed Cordova build
- Added `--debug-build` support for QMake and CMake templates

1.10.30 Changes in v5.5.1

- New `--config` argument to specify a different path to the clickable.json file
- New configuration called `clickable_minimum_required` to specify a minimum version of Clickable
- New `make_args` configuration for passing arguments to make

1.10.31 Changes in v5.5.0

- build-libs now only uses the same arch as specified in clickable.json or in the cli args
- Added the option to build/clean only one lib
- Added support for GOPATH being a list of paths
- Exits with an error with an invalid command

1.10.32 Changes in v5.4.0

- Added support for Rust apps
- Added support for distros using SELinux

1.10.33 Changes in v5.3.3

- More fixes for building libraries
- Set the home directory to /home/phablet in desktop mode

1.10.34 Changes in v5.3.2

- Fixed issue building libraries
- Create arch specific directories in .clickable
- Fixed –dirty breaking when using a custom default set of commands

1.10.35 Changes in v5.3.1

- Fixed dependencies in library prebuild

1.10.36 Changes in v5.3.0

- *Added options for compiling libraries*

1.10.37 Changes in v5.2.0

- Fixed bug in build template auto detection
- Added new dirty build option

1.10.38 Changes in v5.1.1

- Fixed bug in “shell” command

1.10.39 Changes in v5.1.0

- Added app template for QML/C++ with a main.cpp

1.10.40 Changes in v5.0.2

- Fixed publish command not exiting with an error code when there is an error

1.10.41 Changes in v5.0.1

- Fixed typo in cache path
- Improved Cordova support

1.10.42 Changes in v5.0.0

- **New features**
  - Xenial by default (use `--vivid` to compile for 15.04)
  - Major code refactor
  - **More environment variables**
    * `CLICKABLE_ARCH` - Overrides the clickable.json’s `arch`

        * `CLICKABLE_TEMPLATE` - Overrides the clickable.json's `template`
        * `CLICKABLE_DIR` - Overrides the clickable.json's `dir`
        * `CLICKABLE_LXD` - Overrides the clickable.json's `lxd`
        * `CLICKABLE_DEFAULT` - Overrides the clickable.json's `default`
        * `CLICKABLE_MAKE_JOBS` - Overrides the clickable.json's `make_jobs`
        * `GOPATH` - Overrides the clickable.json's `gopath`
        * `CLICKABLE_DOCKER_IMAGE` - Overrides the clickable.json's `docker_image`
        * `CLICKABLE_BUILD_ARGS` - Overrides the clickable.json's `build_args`
        * `OPENSTORE_API_KEY` - Your api key for publishing to the OpenStore
        * `CLICKABLE_CONTAINER_MODE` - Same as `--container-mode`
        * `CLICKABLE_SERIAL_NUMBER` - Same as `--serial-number`
        * `CLICKABLE_SSH` - Same as `--ssh`
        * `CLICKABLE_OUTPUT` - Override the output directory for the resulting click file
        * `CLICKABLE_NVIDIA` - Same as `--nvidia`
        * `CLICKABLE_VIVID` - Same as `--vivid`
- **Removed**
    - Chroot support has been removed, docker containers are recommended going forward
- **clickable.json**
    - **Removed**
        * `package` - automatically grabbed from the manifest.json
        * `app` - automatically grabbed from the manifest.json
        * `sdk` - Replaced by docker_image and the `--vivid` argument
        * `premake` - Use `prebuild`
        * `ssh` - Use the `--ssh` argument
- **Commands**
    - **New**
        * `log` - Dumps the full log file from the app
        * `desktop` - Replaces `--desktop` to run the app in desktop mode
    - **Changed**
        * `init` - Changed to `create` (`init` will still work)
        * `update-docker` - Changed to `update`
    - **Removed**
        * `kill` - Changed to be part of the `launch` command
        * `setup-docker` - Automatically detected and run when using docker
        * `display-on` - Not very useful
- **Command line arguments**

- **New**
  - `--vivid` - Compile the app for 15.04
  - `--docker-image` - Compile the app using a specific docker image
- **Changed**
  - `--serial-number` - Replaces `--device-serial-number`
  - `--ssh` - Replaces `--ip`
- **Removed**
  - `--desktop` - Use the new `desktop` command
  - `--xenial` - Xenial is now the default
  - `--sdk` - Use `--vivid` or `--docker-image`
  - `--device` - Use `shell`
  - `--template` - Use the `CLICKABLE_TEMPLATE` env var
  - `--click` - Specify the path to the click after the `install` command: `clickable install /path/to/click`
  - `--app` - Specify the app name after the `launch` command: `clickable launch app.name`
  - `--name` - Specify the app template after the `create` command: `clickable create pure-qml-cmake`

CHAPTER 2

Install Via Pip (Recommended)

- Install docker, adb, and pip3
- Run (may need sudo): `pip3 install git+https://gitlab.com/clickable/clickable.git`

CHAPTER 3

Install Via PPA (Ubuntu)

- Add the PPA to your system: `sudo add-apt-repository ppa:bhdouglass/clickable`
- Update your package list: `sudo apt-get update`
- Install clickable: `sudo apt-get install clickable`

CHAPTER 4

Install Via AUR (Arch Linux)

- Using your favorite AUR helper, install the clickable-git package
- Example: `pacaur -S clickable-git`

CHAPTER 5

Getting Started

Read the getting started guide to get started developing with clickable.

CHAPTER 6

Code Editor Integrations

Use clickable with the Atom Editor by installing atom-build-clickable made by Stefano.

CHAPTER 7

Issues and Feature Requests

If you run into any problems using clickable or have any feature requests you can find clickable on GitLab.