

---

# Clickable Documentation

*Release 6.14.1*

**Brian Douglass**

**Jun 03, 2020**



---

## Contents

---

<b>1</b>	<b>Using Clickable</b>	<b>3</b>
<b>2</b>	<b>Install Via Pip (Recommended)</b>	<b>33</b>
<b>3</b>	<b>Install Via PPA (Ubuntu)</b>	<b>35</b>
<b>4</b>	<b>Install Via AUR (Arch Linux)</b>	<b>37</b>
<b>5</b>	<b>Getting Started</b>	<b>39</b>
<b>6</b>	<b>Code Editor Integrations</b>	<b>41</b>
<b>7</b>	<b>Issues and Feature Requests</b>	<b>43</b>



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](#).



## 1.1 Install

### 1.1.1 Install Via Pip (Recommended)

- Install docker, adb, git, python3 and pip3 (in Ubuntu: `sudo apt install docker.io adb git python3 python3-pip`)
- Run: `pip3 install --user --upgrade clickable-ut`
- Add pip scripts to your PATH: `export PATH=$PATH:~/local/bin'` >> `~/.bashrc` and open a new terminal for the setting to take effect
- Alternatively, to install nightly builds: `pip3 install --user git+https://gitlab.com/clickable/clickable.git@dev`

### 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 After install

- Let Clickable setup docker (it could ask for the sudo password): `clickable setup`
- Log out or restart to apply changes

## 1.3 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.

### 1.3.1 Getting Logs

To get logs from your 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.

### 1.3.2 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.

### 1.3.3 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.

### 1.3.4 Ubuntu Touch SDK Api Docs

For more information about the Ubuntu Touch QML or HTML SDK check out the [docs over at UBports](#).

### 1.3.5 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).

### 1.3.6 Handling Dependencies

For more information about compiling, using and deploying app dependencies, check out the [docs over at UBports](#).



### 1.3.7 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).

## 1.4 Usage

### 1.4.1 Getting Started

At this point it is assumed that you have completed the [installation process](#)

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` which is outlined in more detail on the previous [getting started](#) page.

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.

Note: The first time you run `clickable` in your app directory, behind the scenes it will download a new Docker container which is about 1GB in size - so plan your time and data transfer environment accordingly. This will only happen the first time you build your app for a specific architecture and when you run `clickable update`.

Running the default sub-commands will:

- 1) Clean the build directory (by default `./build/<arch_triplet>/app`)
- 2) Build 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

Note: ensure your device is in [developer mode](#) for the app to be installed when using `adb` or [enable ssh](#) when using `ssh`.

### 1.4.2 Configuration

It is recommend to 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. If there is none Clickable will ask if it should attempt to detect the type of app and choose a fitting [builder](#) with default configuration.

### 1.4.3 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. Make sure to [enable ssh](#) on your device for this to work.

### 1.4.4 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`.

## 1.5 Debugging

### 1.5.1 Desktop Mode

The easiest way to do GDB Debugging via Clickable is 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>'`.

To analyze errors in memory access run `clickable desktop --valgrind`.

### 1.5.2 On Device

Two terminals are required to do debugging on the device, one to start the `gdbserver` and the other one to start `gdb`. In the first terminal run `clickable gdbserver` and in the second one `clickable gdb`. This method is limited to apps that are started via their own binary file.

The `clickable gdbserver` command provides the server at `localhost:3333`. In theory one could connect to that one from within any IDE. But to actually make it work, one needs to provide the corresponding libc6 debug symbols. Otherwise the App won't start due to a segfault.

## 1.6 Commands

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

### 1.6.1 `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`.

## 1.6.2 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.

## 1.6.3 clickable desktop --nvidia

`clickable` checks automatically if `nvidia-drivers` are installed and turns on `nvidia` mode. If `prime-select` is installed, it is queried to check whether the `nvidia-driver` is actually in use. The `--nvidia` flag lets you manually enforce `nvidia` mode. The `--no-nvidia` flag in contrast lets you disable automatic detection.

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.

## 1.6.4 clickable ide <custom\_command>

The desktop mode allows to run arbitrary programs in the docker container, such as an IDE. Add the dependency to the `clickable.json` and append the command, e.g. by running `clickable ide qtcreator`.

## 1.6.5 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.

## 1.6.6 clickable shell

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

## 1.6.7 clickable clean-libs

Cleans out all library build dirs.

### 1.6.8 `clickable build-libs`

Builds the dependency libraries specified in the `clickable.json`.

### 1.6.9 `clickable clean`

Cleans out the build dir.

### 1.6.10 `clickable build`

Builds the project using the specified builder, 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).

Set the manifest architecture field to `@CLICK_ARCH@` to have Clickable replace it with the appropriate value.

### 1.6.11 `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`.

### 1.6.12 `clickable clean-build`

Cleans out the build dir before building the project as outlined in the `clickable build` docs.

### 1.6.13 `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.

### 1.6.14 `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`.

### 1.6.15 `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

### 1.6.16 `clickable launch`

Launches the app on a device.

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

Launches the specified app on a device.

### 1.6.17 `clickable logs`

Follow the apps log file on the device.

### 1.6.18 `clickable log`

Dumps the apps log file on the device.

### 1.6.19 `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.

### 1.6.20 `clickable run "some command"`

Runs an arbitrary command in the clickable container.

### 1.6.21 `clickable update`

Update the docker container for use with clickable.

### 1.6.22 `clickable no-lock`

Turns off the device's display timeout.

### 1.6.23 `clickable writable-image`

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

### 1.6.24 `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.

### 1.6.25 `clickable <custom command>`

Runs a custom command specified in the "scripts" config

### 1.6.26 `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.

### 1.6.27 `clickable <any command> --verbose`

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

### 1.6.28 `clickable <any command> --ssh <ip address>`

Run a command with a device over ssh rather than the default adb.

## 1.7 clickable.json Format

Example:

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

### 1.7.1 Placeholders & Environment Variables

Placeholders are values provided by Clickable that can be used in some configuration fields as `${PLACEHOLDER}` (`$PLACEHOLDER` is deprecated syntax). All placeholders are provided as environment variables during build, additionally. For custom environment variables see *libraries*.

The following table lists all available placeholders.

Placeholder	Output
ARCH_TRIPLET	Target architecture triplet (arm-linux-gnueabi, aarch64-linux-gnu, x86_64-linux-gnu or all)
ROOT	Value of <code>root_dir</code>
BUILD_DIR	Value of <code>build_dir</code>
SRC_DIR	Value of <code>src_dir</code>
INSTALL_DIR	Value of <code>install_dir</code>
CLICK_LD_LIBRARY_PATH	<code>INSTALL_DIR/lib/\${ARCH_TRIPLET}</code> or <code>INSTALL_DIR/lib</code> for architecture independent apps (will be in <code>LD_LIBRARY_PATH</code> on runtime)
CLICK_QML2_IMPORT_PATH	<code>INSTALL_DIR/lib/\${ARCH_TRIPLET}</code> or <code>INSTALL_DIR/qml</code> for architecture independent apps, which is not in <code>QML2_IMPORT_PATH</code> at runtime (otherwise will be in <code>QML2_IMPORT_PATH</code> on runtime)
CLICK_PATH	<code>INSTALL_DIR/lib/\${ARCH_TRIPLET}/bin</code> or <code>INSTALL_DIR</code> for architecture independent apps (will be in <code>PATH</code> on runtime)
<lib>_LIB_BUILD_DIR	Value of <code>build_dir</code> from library with name <lib> (see <i>libraries</i> ), where the library name consists solely of capital letters (e.g. from lib name <code>my-libC++</code> this env var would be <code>MY_LIBC__LIB_BUILD_DIR</code> )
<lib>_LIB_INSTALL_DIR	Value of <code>install_dir</code> from library with name <lib> (e.g. <code>OPENCV_LIB_INSTALL_DIR</code> )
<lib>_LIB_SRC_DIR	Value of <code>src_dir</code> from library with name <lib> (e.g. <code>OPENCV_LIB_SRC_DIR</code> )

Parameters accepting placeholders: `root_dir`, `build_dir`, `src_dir`, `install_dir`, `app_lib_dir`, `app_bin_dir`, `app_qml_dir`, `gopath`, `cargo_home`, `scripts`, `build`, `build_args`, `make_args`, `postmake`, `postbuild`, `prebuild`, `install_lib`, `install_qml`, `install_bin`, `install_data` and `env_vars`.

This is an ordered list. Parameters that are used as placeholders themselves accept only predecessors. Ex: `${ROOT}` can be used in `src_dir`, but not vice-versa.

Example:

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

## 1.7.2 clickable\_minimum\_required

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

## 1.7.3 restrict\_arch

Optional, specifies an exclusive architecture that this configuration is compatible with. This prevents the app from being build for other architectures and may also prevent the desktop mode.

To specify the architecture for building use the cli argument instead (ex: `--arch arm64`).

## 1.7.4 arch

Deprecated, use `restricted_arch` instead.

## 1.7.5 builder

Optional, see *builders* for the full list of options.

## 1.7.6 template

Deprecated, use *builder* instead.

## 1.7.7 prebuild

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

## 1.7.8 build

A custom command to run from the build dir. Required if using the `custom` builder, ignored otherwise.

## 1.7.9 postmake

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

### 1.7.10 postbuild

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

### 1.7.11 env\_vars

Optional, environment variables to be set in the build container. Ex:

```
"env_vars": {  
  "TARGET_SYSTEM": "UbuntuTouch"  
}
```

When passing `--debug` to Clickable, `DEBUG_BUILD=1` is set as an environment variable, additionally.

### 1.7.12 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.

### 1.7.13 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 overridden by the according environment variable.

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

### 1.7.14 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.

### 1.7.15 launch

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

### 1.7.16 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.

### 1.7.17 src\_dir

Optional, a custom source directory. Defaults to `#{ROOT}`



### 1.7.18 install\_dir

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

### 1.7.19 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. Paths must not contain " characters. Supports wildcards as this actually calls `ls "<path>"` in a bash.

### 1.7.20 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/Qt/labs/calendar"
]
```

QML modules will be installed to the correct directory based on the name of the module. In the above example it will be installed to `lib/${ARCH_TRIPLET}/Qt/labs/calendar` because the module specified in the `qmldir` file is `Qt.labs.calendar`. Can be specified as a string or a list of strings. Paths must not contain " characters. Supports wildcards as this actually calls `ls "<path>"` in a bash.

### 1.7.21 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. Paths must not contain " characters. Supports wildcards as this actually calls `ls "<path>"` in a bash.

### 1.7.22 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}"
},
```

Can be specified as a string or a list of strings. Paths must not contain " characters. Supports wildcards as this actually calls `ls "<src>"` in a bash. `${INSTALL_DIR}` is added as prefix if path is not relative to the install dir.

### 1.7.23 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.

### 1.7.24 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`.

### 1.7.25 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.

### 1.7.26 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`.

### 1.7.27 dependencies\_host

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

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

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

### 1.7.28 dependencies\_build

Deprecated, use `dependencies_host` instead.

### 1.7.29 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 depends on.

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

### 1.7.30 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.

### 1.7.31 image\_setup

Optional, dictionary containing setup configuration for the docker image used. The image is based on the default image provided by Clickable.

#### run

Optional, a list of commands to run on image setup (each added as *RUN <cmd>* to the corresponding Dockerfile). Example:

```
"image_setup": {
  "run": [
    "rustup default nightly",
    "rustup install 1.39.0"
  ]
}
```

### 1.7.32 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](#).

### 1.7.33 ignore

Optional, a list of files to ignore when building with pure builder Example:

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

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

### 1.7.34 GOPATH

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

### 1.7.35 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`.

### 1.7.36 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`.

### 1.7.37 test

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

### 1.7.38 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": {
    "builder": "cmake",
    "make_jobs": "1",
    "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`, `env_vars`, `build_args`, `docker_image`, `dependencies_host`, `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 <code>libraries</code> 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.

#### builder

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.

## 1.7.39 Removed keywords

The following keywords are no longer supported:

- `dependencies` (use `dependencies_target` and `dependencies_host` 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`

## 1.8 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.

### 1.8.1 CLICKABLE\_ARCH

Restricts build commands (`build`, `build-libs`, `desktop`) to the specified architecture. Architecture agnostic builds (`all`) are not affected. Useful in container mode.

### 1.8.2 CLICKABLE\_BUILDER

Overrides the `clickable.json`'s *builder*.

### 1.8.3 CLICKABLE\_BUILD\_DIR

Overrides the `clickable.json`'s *dir*.

#### 1.8.4 CLICKABLE\_DEFAULT

Overrides the clickable.json's *default*.

#### 1.8.5 CLICKABLE\_MAKE\_JOBS

Overrides the clickable.json's *make\_jobs*.

#### 1.8.6 GOPATH

Overrides the clickable.json's *gopath*.

#### 1.8.7 CARGO\_HOME

Overrides the clickable.json's *cargo\_home*.

#### 1.8.8 CLICKABLE\_DOCKER\_IMAGE

Overrides the clickable.json's *docker\_image*.

#### 1.8.9 CLICKABLE\_BUILD\_ARGS

Overrides the clickable.json's *build\_args*.

#### 1.8.10 CLICKABLE\_MAKE\_ARGS

Overrides the clickable.json's *make\_args*.

#### 1.8.11 OPENSTORE\_API\_KEY

Your api key for *publishing to the OpenStore*.

#### 1.8.12 CLICKABLE\_CONTAINER\_MODE

Same as *-container-mode*.

#### 1.8.13 CLICKABLE\_SERIAL\_NUMBER

Same as *-serial-number*.

#### 1.8.14 CLICKABLE\_SSH

Same as *-ssh*.

### 1.8.15 CLICKABLE\_OUTPUT

Override the output directory for the resulting click file

### 1.8.16 CLICKABLE\_NVIDIA

Same as `-nvidia`.

### 1.8.17 CLICKABLE\_NO\_NVIDIA

Same as `-no-nvidia`.

### 1.8.18 CLICKABLE\_DIRTY

Overrides the `clickable.json`'s `dirty`.

### 1.8.19 CLICKABLE\_DEBUG\_BUILD

Same as `--debug`

### 1.8.20 CLICKABLE\_TEST

Overrides the `clickable.json`'s `test`.

### 1.8.21 CLICKABLE\_DARK\_MODE

Same as `--dark-mode`

## 1.9 App Templates

Find the source code for the templates on [GitLab](#).

### 1.9.1 QML Only

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

### 1.9.2 C++ (Plugin)

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

### 1.9.3 Python

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

## 1.9.4 HTML

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

## 1.9.5 Go

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

## 1.9.6 C++ (Binary)

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

## 1.9.7 Rust

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

## 1.10 Builders

Builders have been called Build Templates in the early days of Clickable.

### 1.10.1 pure-qml-qmake

A purely qml qmake project.

### 1.10.2 qmake

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

### 1.10.3 pure-qml-cmake

A purely qml cmake project

### 1.10.4 cmake

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

### 1.10.5 custom

A custom build command will be used.

### 1.10.6 cordova

A project that builds using cordova



### 1.10.7 pure

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

### 1.10.8 precompiled

A project that does not need to be compiled. All files in the project root will be copied into the click. There may be precompiled binaries or libraries included in apps build with this builder. Specifying the *restrict\_arch* in the `clickable.json` file can be useful with this builder.

### 1.10.9 python

Deprecated, use “precompiled” instead.

### 1.10.10 go

A project that uses go version 1.6.

### 1.10.11 rust

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

## 1.11 Continuous Integration

### 1.11.1 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.

### 1.11.2 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.

## 1.12 Changelog

### 1.12.1 Changes in v6.14.1

- Fixed issue when using the Atom editor extension
- Merged the C++ templates into one and included qrc compiling
- Minor bug fixes

### 1.12.2 Changes in v6.14.0

- Added new setup command to help during initial setup of Clickable
- Prevent building in home directory that isn't a click app

### 1.12.3 Changes in v6.13.1

- Fixed issue with an error showing the wrong message
- Fixed multiple ppas in `dependencies_ppa`

### 1.12.4 Changes in v6.13.0

- Fixed packaging issues and published to pypi
- Fixed the builder auto detect showing up when it wasn't needed
- Added better errors when the current user is not part of the docker group
- Remove apps before installing them to avoid apparmor issues
- Various bug fixes
- Added optional git tag versioning in cmake based templates

### 1.12.5 Changes in v6.12.2

- Fixed bug checking docker image version
- Renamed build template to builder
- Fixed the publish command

### 1.12.6 Changes in v6.12.1

- Bug fixes
- Display nicer error messages when a template fails to be created
- Fixed auto detecting the build template

### 1.12.7 Changes in v6.12.0

- Added new feature for debugging via *valgrind*
- Added new *ide* command to allow running arbitrary graphical apps like qtcreator
- Code improvements
- Added versioning to the docker images to allow Clickable to depend on certain features in the image

### 1.12.8 Changes in v6.11.2

- Fixed the `review` and `clean-build` commands not working

### 1.12.9 Changes in v6.11.1

- Fixed the `run` command not working

### 1.12.10 Changes in v6.11.0

- Added *on device debugging with gdb*.
- Deprecated chaining commands (this will be removed in the next major release)
- Fixed the build home directory for libraries
- Added error when trying to use docker images on unsupported host architectures
- Use the host architecture as the default when building in container mode
- Enable localhost access and pseudo-tty in run command
- When using CMake a Release build will be created unless `--debug` is specified
- Added new library placeholders
- Added new `clean-build` command
- Fixed issues with `clickable create` on older versions of Ubuntu
- Various minor bug fixes and code improvements

### 1.12.11 Changes in v6.10.1

- Fixed issues installing dependencies when in container mode

### 1.12.12 Changes in v6.10.0

- Fix containers being rebuilt when switching between desktop mode and building for amd64
- Enabled compiling rust apps into arm64
- Make `install_data` paths relative to the install dir
- Fixed the `clickable create` command when using an older version of git

### 1.12.13 Changes in v6.9.1

- Fixed broken lib builds

### 1.12.14 Changes in v6.9.0

- Placeholders are now allowed in env vars
- Changed placeholder syntax to `${PLACEHOLDER}`, the old syntax is now deprecated
- Replaced `dependencies_host` with `dependencies_build` to avoid confusion about the name, `dependencies_build` is now deprecated
- Normalized env var names
- Added new `precompiled` build template to replace the now deprecated `python` build template

- Fixed issues using the `install_*` configuration options
- `install_qml` will now install qml modules to the correct nested path
- A per project home directory gets mounted during the build process
- Cleaned up arch handling and improved conflict detection

### 1.12.15 Changes in v6.8.2

- Fixed broken architecture agnostic builds

### 1.12.16 Changes in v6.8.1

- Fixed new architecture errors breaking architecture agnostic builds

### 1.12.17 Changes in v6.8.0

- Fixed the `ARCH` placeholder breaking `ARCH_TRIPLET` placeholder
- Added new `env_vars` configuration for passing custom env vars to the build process
- Fixed errors on systems where `/etc/timezone` does not exist
- Added errors to detect conflicting architecture settings
- Improved multi arch support

### 1.12.18 Changes in v6.7.2

- Fixed architecture mismatch error for architecture agnostic templates

### 1.12.19 Changes in v6.7.0

- New error when there is no space left on the disk
- New error when the manifest's architecture does not match the build architecture
- New option to use `@CLICK_ARCH@` as the architecture in the manifest to allow Clickable to automatically set the architecture

### 1.12.20 Changes in v6.6.0

- Fixed issue in with timezone detection
- Added better detection for nvidia mode and added a new `--no-nvidia` argument

### 1.12.21 Changes in v6.5.0

- New bash completion, more info [here](#)
- Fixed crash when running in container mode
- Added `image_setup` configuration to run arbitrary commands to setup the docker image
- Added arm64 support for qmake builds

### 1.12.22 Changes in v6.4.0

- Use the system timezone when in desktop mode

### 1.12.23 Changes in v6.3.2

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

### 1.12.24 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 exists in the PATH
- Added a prompt for autodetecting the template type
- Improved Clickable's logging

### 1.12.25 Changes in v6.2.1

- Fixed env vars in libs

### 1.12.26 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

### 1.12.27 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

### 1.12.28 Changes in v6.0.3

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

### 1.12.29 Changes in v6.0.2

- Fixed container mode

### 1.12.30 Changes in v6.0.1

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

### 1.12.31 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

### 1.12.32 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

### 1.12.33 Changes in v5.14.0

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

### 1.12.34 Changes in v5.13.3

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

### 1.12.35 Changes in v5.13.2

- Fixed libraries not building after latest update

### 1.12.36 Changes in v5.13.1

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

### 1.12.37 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

### 1.12.38 Changes in v5.12.3

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

### 1.12.39 Changes in v5.12.2

- Fixed `scripts` breaking Clickable

### 1.12.40 Changes in v5.12.1

- Fixed issues with build dir

### 1.12.41 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

### 1.12.42 Changes in v5.11.0

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

### 1.12.43 Changes in v5.10.0

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

### 1.12.44 Changes in v5.9.1

- Fixed missing schema file

### 1.12.45 Changes in v5.9.0

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

### 1.12.46 Changes in v5.8.1

- Fixed a bug in `make_args`



### 1.12.47 Changes in v5.8.0

- New configuration option for automatically including ppa's 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`

### 1.12.48 Changes in v5.7.0

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

### 1.12.49 Changes in v5.6.1

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

### 1.12.50 Changes in v5.6.0

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

### 1.12.51 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`

### 1.12.52 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

### 1.12.53 Changes in v5.4.0

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

### 1.12.54 Changes in v5.3.3

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

### 1.12.55 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

### 1.12.56 Changes in v5.3.1

- Fixed dependencies in library prebuild

### 1.12.57 Changes in v5.3.0

- *Added options for compiling libraries*

### 1.12.58 Changes in v5.2.0

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

### 1.12.59 Changes in v5.1.1

- Fixed bug in “shell” command

### 1.12.60 Changes in v5.1.0

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

### 1.12.61 Changes in v5.0.2

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

### 1.12.62 Changes in v5.0.1

- Fixed typo in cache path
- Improved Cordova support

### 1.12.63 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`

---

### Install Via Pip (Recommended)

---

- Install docker, adb, git, python3 and pip3 (in Ubuntu: `sudo apt install docker.io adb git python3 python3-pip`)
- Run: `pip3 install --user --upgrade clickable-ut`
- Add pip scripts to your PATH: `export PATH=$PATH:~/local/bin'` >> `~/bashrc` and open a new terminal for the setting to take effect
- Alternatively, to install nightly builds: `pip3 install --user git+https://gitlab.com/clickable/clickable.git@dev`



## 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](#).