Build system usage

See also: Build system setup

Do everything as normal user, **don't use root user or sudo!**

Do not build in a directory that has spaces in its full path. Write all command line commands for build system commands in a terminal window opened in the `<buildsystem root>` directory, e.g. `~/source/` if you wrote the **git clone** command in your home folder (default terminal location).

1. Update the sources.
2. Update and install package feeds.
3. Select a specific branch/revision/tag to use.
4. Configure the firmware image you want to obtain.
5. Start the build. (automatically compile toolchain, cross-compile sources, package packages, and generate an image ready to be flashed).
6. Proceed with the firmware flashing instructions: Factory install or Sysupgrade.

Updating Sources with Git

Sources in development branch change frequently. It is recommended that you work with the latest sources.

```
git pull
```

Updating Feeds

Installing in context of `./scripts/feeds` script means “making package available in `make menuconfig` ” rather than really installing or compiling package.
Also, after you have been developing for a while, and your copy of the repository is getting behind, running `feeds update -a` will pull the latest updates for the feeds.

1. Update feeds:
   ```
   ./scripts/feeds update -a
   ```
2. Make downloaded package/packages available in `make menuconfig` :
   - single package:

```
./scripts/feeds install <PACKAGENAME>
```

- all packages:

```
./scripts/feeds install -a
```

Creating a local feed

1. Add a line to `feeds.conf.default` . `src-link my_packages ~/$buildroot /my_packages` . Replace the `$buildroot` with cloned openwrt sources directory e.g. `~/openwrt`
2. Create the dir: `mkdir -p $buildroot/my_packages/$section/$category /$package_name` . Replace the `$package_name` with the name of your package e.g. `mkdir -p my_packages/net/network/rpcbind` . The section and category can be found in the `Makefile` .
3. Download the `Makefile` from github, download the source package from sources.openwrt.org and place them in the `$package_name` directory.
4. Now run: `./scripts/feeds update -a ; ./scripts/feeds install $package_name`
5. If you are doing this to resolve a dependency you can run `install` one more time and you should notice the dependency has been resolved.

Selecting a specific major revision

Of course this step isn't necessary if you want to build bleeding edge images.

Specify branches

Each branch contains the baseline code for the release version (e.g. 17.01, 18.06, 19.07) and the individual releases (e.g. 17.01.1, 17.01.2, 18.06.1, 19.07.3 etc). Each branch is intended to contain **stable** code with carefully selected fixes and updates backported from the development branch.

To use a branch, you should first clone the Git repository using the **git clone** command shown above. Then move to the branch by using the **git checkout** command.

For OpenWrt 19.07.4:

```
git checkout v19.07.4
```

When changing branches, it is recommended to perform a thorough scrub of your source tree by using the **make distclean** command. This ensures that your source tree does not contain any build artifacts or configuration files from previous build runs.

Specify individual release / tag

See above for an explanation. The same conditions and recommendations apply here.

This may be useful when you want to be able to install packages from the repositories for a long time.

Example for OpenWrt 19.07.4:

```
git fetch --tags
git tag -l
git checkout v19.07.4
```

Source: Checkout a tagged release's source code (https://forum.openwrt.org/t/checkout-a-tagged-releases-source-code-like-v18-06-1/1256)

Compile with the same packages as the official image

Compile openwrt in a way that it gets the same packages as the default official image:

`…/releases/[stable release]/targets/[architecture]/[target]/config.buildinfo`

Example

```
# v19.07.0 and later
wget https://downloads.openwrt.org/releases/19.07.0/targets/ramips/mt7621/config.buildinfo -O .config

# v18.06.8 and before
wget https://downloads.openwrt.org/releases/18.06.1/targets/ramips/mt7621/config.seed -O .config
```

Sourcecompiling-openwrt-exactly-like-the-official-one/23214 (https://forum.openwrt.org/t/compiling-openwrt-exactly-like-the-official-one/23214)

Image Configuration

Typical actions:

1. **`git checkout <branch/revision/tag>`** when you don't want HEAD.
2. **`make menuconfig`** and set target.
3. **`make defconfig`** to set default config for build system and device.
4. **`make kernel_menuconfig`** (optional ⚠ it's highly likely that kernel modules from the repositories won't work when you make changes here).

5. `make menuconfig` and modify set of package.
6. `make download` (download all dependency source files before final make, enables multi-core compilation).
7. `scripts/diffconfig.sh > mydiffconfig` (save your changes in the text file `mydiffconfig`).
8. **`make`** start the build process.

Make menuconfig

The **build system configuration interface** handles the selection of the target platform, packages to be compiled, packages to be included in the firmware file, some kernel options, etc.
Start the build system configuration interface by writing the following command:

```
make menuconfig
```

This will update the dependencies of your existing configuration automatically, and you can now proceed to build your updated images.

You will see a list of options. This list is really the top of a tree. You can select a list item, and descend into its tree.

To search for the package or feature in the tree, you can type the “/” key, and search for a string. This will give you its locations within the tree.

For most packages and features, you have three options: `y` , `m` , `n` which are represented as follows:

- pressing **`y`** sets the **`<*>`** built-in label
  This package will be compiled and included in the firmware image file.
- pressing **`m`** sets the **`<M>`** package label
  This package will be compiled, but **not** included in the firmware image file. (E.g. to be installed with opkg after flashing the firmware image file to the device.)
- pressing **`n`** sets the **`< >`** excluded label
  The source code will not be processed.

When you save your configuration, the file **`<buildroot dir>/.config`** will be created according to your configuration.

When you open `menuconfig` you will need to set the build settings in this order (also shown in this order in `menuconfig` 's interface):

1. Target system (general category of similar devices)
2. Subtarget (subcategory of Target system, grouping similar devices)
3. Target profile (each specific device)
4. Package selection
5. Build system settings
6. Kernel modules

Select your device's **Target system** first, then select the right **Subtarget**, then you can find

your device in the **Target profile**'s list of supported platforms.

Configure using config diff file

Beside `make menuconfig` another way to configure is using a configuration diff file. This file includes only the changes compared to the default configuration. A benefit is that this file can be version-controlled in your downstream project. It's also less affected by upstream updates, because it only contains the changes.

Creating diff file

This file is created using the `./scripts/diffconfig.sh` script.

```
./scripts/diffconfig.sh > diffconfig # write the changes to diffconfi
g
```

Note: Since r2752 LEDE firmware make process automatically creates the configuration diff file `config.seed` file to the target image directory.

Using diff file

These changes can form the basis of a config file (`<buildroot dir>/.config`). By running `make defconfig` these changes will be expanded into a full config.

```
cp diffconfig .config   # write changes to .config
make defconfig   # expand to full config
```

These changes can also be added to the bottom of the config file (`<buildroot dir>/.config`), by running `make defconfig` these changes will override the existing configuration.

```
cat diffconfig >> .config   # append changes to bottom of .config
make defconfig   # apply changes
```

Patches

The build system integrates *quilt (https://en.wikipedia.org/wiki/Quilt (software))* for easy patch management:

Custom files

In case you want to include some custom configuration files, the correct place to put them is:

- **`<buildroot dir>/files/`**

For example, let's say that you want an image with a custom **`/etc/config/firewall`** or a

custom **`etc/sysctl.conf`** , then create this files as:

- **`<buildroot dir>/files/`**`etc/config/firewall`
- **`<buildroot dir>/files/`**`etc/sysctl.conf`

E.g. if your <buildroot dir> is `~/source` and you want some files to be copied into firmware image's `/etc/config` directory, the correct place to put them is `~/source/files/etc/config` .

Defconfig

```
make defconfig
```

will produce a default configuration of the target device and build system, including a check of dependencies and prerequisites for the build environment.

Defconfig will also remove outdated items from `.config` , e.g. references to non-existing packages or config options.

It also checks the dependencies and will add possibly missing necessary dependencies. This can be used to “expand” a short .config recipe (like diffconfig output, possible even pruned further) to a full .config that the make process accepts.

Kernel configuration (optional)

Note that `make kernel_menuconfig` modifies the Kernel configuration templates of the build tree and clearing the build_dir will not revert them. Also you won't be able to install kernel packages from the official repositories when you make changes here.

While you won't typically need to do this, you can do it:

```
make kernel_menuconfig CONFIG_TARGET=subtarget
```

CONFIG_TARGET allows you to select which config you want to edit. possible options: target, subtarget, env.

The changes can be reviewed with

```
git diff target/linux/
```

and reverted with

```
git checkout target/linux/
```

Source Mirrors

The 'Build system settings' include some efficient options for changing package locations which makes it easy to handle a local package set:

1. Local mirror for source packages
2. Download folder

In the case of the first option, you simply enter a full URL () to the HTTP () or FTP () server on which the package sources are hosted. Download folder would in the same way be the path to a local folder on the build system (or network). If you have a web/ftp-server hosting the tarballs, the build system will try this one before trying to download from the location(s) mentioned in the Makefiles . Similar if a local 'download folder', residing on the build system, has been specified.

The 'Kernel modules' option is required if you need specific (non-standard) drivers and so forth – this would typically be things like modules for USB or particular network interface drivers etc.

Download sources and multi core compile

Before running final make it is best to issue make download command first, this step will pre-fetch all source code for all dependencies, this enables you compile with more cpu cores (for example, make -j10, for 4 core, 8 thread cpu works great).

If you try compiling OpenWrt on multiple cores and don't download all source files for all dependency packages it is very likely that your build will fail.

```
make download
```

Building Images

Everything is now ready for building the image(s), which is done with one single command:

```
make
```

Make Tips

`make download` will pre-download all source code for all dependencies, this will enable multi core compilation to succeed, without it is is very likely to fail. `make -j` **N** will speed up compilation by using up to **N** cores or hardware threads to speed up compilation, `make -j9` fully uses 8 cores or hardware threads.

Example of pre-downloading and building the image(s) on a 4 core CPU:

```
make -j5 download world
```

You can use "nproc" command to get available CPU count (https://unix.stackexchange.com/questions/208568/how-to-determine-the-maximum-number-to-pass-to-make-j-option):

```
make -j$(nproc) download world
```

or a better macro with `nproc+1` :

```
make -j$(($(nproc)+1))
```

Building in the background

If you intend to use your system while building, you can have the build process use only idle I/O and CPU capacity like this (4 core, 8 thread CPU):

```
make download
ionice -c 3 nice -n19 make -j9
```

Building single Packages

When developing or packaging software, it is convenient to be able to build only the package in question (e.g. with package `jsonpath`):

```
make package/utils/jsonpath/compile V=s
```

For a rebuild:

```
make package/utils/jsonpath/{clean,compile} V=s
```

It doesn't matter what feed the package is located in, this same syntax works for any installed package.

Spotting build errors

If for some reason the build fails, the easiest way to spot the error is to do:

```
make V=s 2>&1 | tee build.log | grep -i -E "^make.*(error|[12345]...E
ntering dir)"

make V=s 2>&1 | tee build.log | grep -i '[^_-"a-z]error[^_-.a-z]'
(may not work)
```

If **grep** throws an error, use **fgrep** instead.

The above saves a full verbose copy of the build output (with stdout piped to stderr) in `~/source/build.log` and shows errors on the screen (along with a few spurious instances of 'error').

Another example:

```
ionice -c 3 nice -n 20 make -j 2 V=s CONFIG_DEBUG_SECTION_MISMATCH=y
2>&1 | tee build.log
```

The above saves a full verbose copy of the build output (with stdout piped to stderr) in build.log while building using only background resources on a dual core CPU.

Yet another way to focus on the problem without having to wade through tons of output from Make as described above is to check the corresponding log in `logs` folder. i.e. if the build fails at `make[3] -C package/kernel/mac80211 compile` , then you can go to `<buildroot>/logs/package/kernel/mac80211` and view the `compile.txt` found there.

Getting beep notification

Depending on your CPU, the process will take a while, or while longer. If you want an acoustic notification, you could use `echo -e '\a'`:

```
make V=s ; echo -e '\a'
```

Skipping failed packages

If you are building everything (not just the packages to make a flashable image), you will probably want to keep building all packages even if some have compile errors and won't be built.

```
IGNORE_ERRORS=1 make <make options>
```

Locating Images

After a successful build, the freshly built image(s) can be found below the newly created `<buildroot_dir>/bin` directory. The compiled files are additionally classified by the target platform and subtarget, so e.g. a `generic` firmware built for an `ar71xx` device will be located in `<buildroot_dir>/bin/targets/ar71xx/generic` directory (and the package files are below `<buildroot_dir>/bin/packages/mips_24kc`).

E.g. if your <buildroot_dir> is `~/source` , the binaries are in `~/source/bin/targets/ar71xx/generic` and `~/source/bin/packages/mips_24kc` .

See Directory structure for details

Cleaning Up

You might need to clean your *build environment* every now and then. The following `make` -targets are useful for that job:

Clean

```
make clean
```

deletes contents of the directories `/bin` and `/build_dir` .

`make clean` does not remove the toolchain, and it also avoids cleaning architectures/targets

other than the one you have selected in your `.config`

It is a good practice to do `make clean` before a build to ensure that no outdated artefacts have been left from the previous builds. That may not be necessary always, but as a general rule it helps to ensure quality builds.

Dirclean

```
make dirclean
```

deletes contents of the directories `/bin` and `/build_dir` and additionally `/staging_dir` and `/toolchain` (=the cross-compile tools), `/tmp` (e.g data about packages) and `/logs`. 'Dirclean' is your basic “Full clean” operation.

Distclean

```
make distclean
```

nukes everything you have compiled or configured and also deletes all downloaded feeds contents and package sources.

CAUTION: In addition to all else, this will **erase your build configuration (`<buildroot_dir>/.config`)**. Use only if you need a “factory reset” of the build system!

There are numerous other functionalities in the build system, but the above should have covered some of the fundamentals.

Clean less

In more time, you may not want to clean so many objects, then you can use some of the commands below to do it.

Clean linux objects.

```
make target/linux/clean
```

Clean package base-files objects.

```
make package/base-files/clean
```

Clean luci.

```
make package/luci/clean
```

Examples

- https://github.com/mwarning/openwrt-examples (https://github.com/mwarning/openwrt-examples)
- https://forum.openwrt.org/viewtopic.php?pid=129319#p129319 (https://forum.openwrt.org/viewtopic.php?pid=129319#p129319)
- https://forum.openwrt.org/viewtopic.php?id=28267 (https://forum.openwrt.org/viewtopic.php?id=28267)

Troubleshooting

- Beware of unusual environment variables.
- First get more information on the problem using the make option `“make V=sc”` or enable logging.
- Read more about make options: Buildroot Techref.

Missing source code file, due to download problems

First check if the URL () path in the make file contains a trailing slash, then try with it removed (helped several times). Otherwise try to download the source code manually and put it into “dl” directory.

Compilation errors

Try to update the main source and all the feeds (Warning! May result in other problems). Check for a related bug in the bugtracker (depends from the feed the package comes from). Otherwise report the problem there, by mentioning the package, the target data (CPU, image, etc.) and the code revisions (main & package).

WARNING: skipping <package> -- package not selected

Run `make menuconfig` and enable compilation for your package. It should be labeled with `<*>` or `<M>` to work correctly.

Flashable images for my device are not generated

When you execute `make` to build a flashable image for your device, both a sysupgrade and a factory image should be generated for every board that is linked to the device profile that you have selected via `make config` or `make menuconfig` .

If running `make` does *not* yield images for one (or even all) of the boards linked to the device profile that you have selected, than you probably have selected/enabled too many options or packages, and the image was too big to be flashed onto your device.

Notes

- Compiler Optimization Tweaks (https://forum.openwrt.org/viewtopic.php?id=35323)

Last modified: 2021/01/05 08:34 by vgaetera

Except where otherwise noted, content on this wiki is licensed under the following license:
CC Attribution-Share Alike 4.0 International