Compile batocera.linux

The official and recommended way to compile batocera.linux is by using a development container made on purpose with Docker. Although you can do it directly on your Linux machine, if it's a supported Ubuntu Linux.

Note that it's also possible to compile batocera.linux within an LXC container but it's not recommended or officially supported.

If you are not familiar with git, you should first take a look at this page explaining the basics for contributing to Batocera Linux with git.

Install prerequisites

Choose either Docker or Direct Compilation.

Make sure you have a reasonably fast CPU, and at least 8GB of RAM (even 12GB+ if you need to compile MAME). If you have a CPU with 8 cores or more (lucky you!), you might need more than 8GB RAM to compile batocera.linux with all threads in parallel. You also need between 80GB and 130GB of free disk space for each platform you intend to compile for in order to download and compile the final Batocera image.

The x86_64 version is the largest, downloading packages and building them takes up to 130GB.

If compiling inside of a VM, you may need to set its CPU type to host to avoid “missing CPU instruction” errors like #error PCSX2 requires compiling for at least SSE 4.1.

Docker

If you already have a working Docker installation on your Linux system, you just need to install git (if you don't have that already as well), and skip to the preparations section.

Otherwise, the first step is to install Docker on your machine. Each OS and Linux distribution has a particular way to install Docker.

Please note that most developers use Ubuntu Linux to compile batocera.linux. In January 2022, Docker on Windows was not stable enough to compile Batocera.

In March 2020, with the Docker container running as non-root (regular user), you can compile Batocera on MacOS Mojave 10.14 (Darwin 18.0.0).

- Linux Ubuntu/Debian:
  - Installing from the Distribution packages is not recommended. Please refer to the official
To be able to run docker without root privileges don't forget to add your user to the `docker` group:
```
sudo usermod -aG docker your-user
```
then log out and log back in.

- Linux Solus: `sudo eopkg install --y git docker make`
- Arch Linux: `sudo pacman -S docker`
- (for reference) MacOS: see https://docs.docker.com/docker-for-mac/
- (for reference) Windows: see https://store.docker.com/editions/community/docker-ce-desktop-windows

Once Docker is installed, you have to get it up and running (depending on your OS, for Linux Solus or Arch Linux for example, you would start it up with `sudo systemctl start docker.service` and make sure that your user is in the docker group).

Git is used to download the batocera.linux sources, outside of the docker container.

### Direct Compilation

The way to install the necessary packages vary for each Linux distribution. You can find the necessary packages for Ubuntu below.

**Ubuntu 18.04**

May also work with Ubuntu 16.04.

```
sudo apt-get install build-essential git libncurses5-dev libssl-dev mercurial texinfo zip default-jre imagemagick subversion hgsubversion autoconf automake bison scons libglib2.0-dev bc mtools u-boot-tools flex wget cpio dosfstools libtool
```

```
sudo dpkg --add-architecture i386
```

```
sudo apt-get update
```

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

```
pip3 install conan
```

**Ubuntu 20.04**

Includes requirements for some experimental packages.

```
sudo apt-get install build-essential git libncurses5-dev libssl-dev mercurial texinfo zip default-jre imagemagick subversion autoconf automake bison scons libglib2.0-dev bc mtools u-boot-tools flex wget cpio dosfstools libtool gcc-multilib g++-multilib python3-pip
```

```
sudo dpkg --add-architecture i386
```

```
sudo apt-get update
```

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

```
pip3 install conan
```
Preparations

Download the batocera.linux sources

If you are already set up to make a Pull Request to Batocera, run:

```
  cd batocera.linux
  git checkout master
```

and skip the rest of this step.

Choose a work directory (your $HOME?) and clone the batocera.linux source code:

```
git clone https://github.com/batocera-linux/batocera.linux.git
```

Enter the newly created directory.

```
  cd batocera.linux
```

By default, the buildroot submodule will not be cloned. This is required for Batocera, run the following:

```
git submodule init
git submodule update
```

You only need to do this once.

Choose the target you want to build

Depending on the target you want to build Batocera for, you need to select one of the batocera-XXXX_defconfig files, where XXXX has to be replaced with the architecture you are building to. Available ones are:

- rpi1
- rpi2
- rpi3
- rpi4
- rpizero2
- cha (capcom home arcade)
- odroidxu4
- rk3326 (odroid go advance/super and clones)
- rk3288 (miqi, tinkerboard)
- rk3399 (rockpro64 and clones)
- gameforce
- orangepi-pc
- orangepi-zero2
x86_64  
x86 (32 bits, old PCs)  
s812  
s905 (odroid c2)  
s905gen2  
s905gen3  
s912  
s922x (odroid n2)  
tritium-h5

You can find them all listed in the configs directory.

**Compilation**

**Easy compilation**

To make things easier, there is makefile described here that makes the compilation process smoother.

1. **Add your user to the docker group**

If you haven’t already, add your user to the “docker” group so that you can run docker commands:

```
sudo usermod -a -G docker $USER
```

Enter your password when prompted to, then reboot your computer (or if connected via SSH, restart the session).

2. **Install build environment**

```
make batocera-docker-image
```

3. **Customize directories**

By default, Batocera will download dependencies to $(PROJECT_DIR)/dl and build to the $(PROJECT_DIR)/output folder. You can check the current default directories with the following command:

```
make vars
```

To change the directories that Batocera will build to, rename the batocera.mk.template file to batocera.mk and change the DL_DIR ?= and OUTPUT_DIR ?= lines to point to wherever you want to.

4. **Build an image**

Command is different for each target architecture, but share the same suffix like make rockpro64-build or:
make x86_64-build

You can check valid targets architectures with make vars.

Every time a new image is built, the output/<architecture> folder is completely replaced. Images will not stack up over time, so if you need to keep a particular version copy it out of this folder first.

5. Shell

It's also possible to get a shell to the desired build environment. Similar to -build, with make rockpro64-shell or:

make x86_64-shell

6. Build a single package

To build a single package, you can open a shell or use the -pkg target. Examples:

make rk3326-pkg PKG=batocera-splash
make x86_64-shell PKG=libretro-mame

7. ccache

To enable ccache for all builds, add the necessary options to batocera.mk. An example is provided there. See EXTRA_OPTS. No need to edit defconfig.

8. Compile clean builds

This will clear all the data in output/<arch> and build “from scratch”. This is useful for:

- If building an older version of Batocera, and you've already built a future version of it, always do a cleanbuild to ensure that the packages in the build cache do not cause conflicts with older packages producing unintelligible error messages.
- Avoid cherry-picking commits; every commit (in Batocera at least) may be dependent on another previous commit, and the effect daisy chains even with commits that seem otherwise independent. If a single package has been built with a cherry-picked commit, cleanbuild to ensure no future errors.
- After building an older version and wanting to build the most recent version again, do another cleanbuild. Honestly, the only time you can consistently use the regular build command is if you're solely just following master branch and continually building without modifications.
- When facing seemingly random errors that other devs are not facing when building packages, do a cleanbuild. This can also include errors that arise from the compilation process being interrupted.

make x86_64-cleanbuild

9. Build a webserver to upgrade your Batocera
You can easily upgrade your Batocera test machine with the build you have just compiled. Let's imagine your build box's IP address is 10.0.0.2. If your build machine has Python3 installed, you can run a webserver with:

```
make x86_64-webserver
```

Then, you can launch an upgrade from your Batocera test box. If your build box hosting the web server is on 10.0.0.10, you can simply upgrade to your freshly brewed build by entering from your Batocera SSH:

```
batocera-upgrade http://10.0.0.10:8000
```

**Traditional compilation method**

Of course you can still use out of tree builds to compile Batocera. So, for x86_64, you can do the following:

```
make O=$PWD/output/x86_64 BR2_EXTERNAL=$PWD -C $PWD/buildroot batocera-x86_64_defconfig
cd output/x86_64
make
```

Source tree will stay pristine and we be reused for other builds following the same procedure with a different output directory. For example:

```
make O=$PWD/output/rpi3 BR2_EXTERNAL=$PWD -C $PWD/buildroot batocera-rpi3_defconfig
cd output/rpi3
make
```

Then, you can take some time for a coffee (or two, or two hundreds actually). Depending on how powerful your CPU is, how much RAM you have, and how fast your SSD/HDD is, compiling a whole Batocera system can take many hours. Don't forget, it's the full OS with all the emulators that we are compiling here, it's not a small task.

Please also mind that the process will take a significant amount of space, so ensure to have between 50GB (RPi) and 150+GB (x86_64) of free space on that partition.

**Locating the image**

Exit the container if you ran the build inside of it.

The output will be under `output/images/batocera/images/<architecture>` and will be gzipped. Its name contains the release version, the platform, and the build date. For instance: `output/images/batocera/images/x86_64/batocera-x86_64-33-20220130.img.gz`.

You can then flash it with Etcher by following the install instructions, upgrading using the webserver command above, manually upgrading or by using the following commands:
Tips

Building configuration

Build options in batocera.mk can further be tweaked to accommodate your preferences.

> It is important that each option has a leading space character at the start of its line. Do not remove this.

By default, Docker will take all of your computer's resources during compilation. If you'd like to continue using your computer during compilation, it is recommended to lower the `MAKE_JLEVEL` value down to 1.

```
MAKE_JLEVEL := 1
```

You can also limit compilation to a single core by setting `PARALLEL_BUILD` to n. Be warned however, this will cause a dramatic (up to twelve times longer!) increase in compilation time.

```
PARALLEL_BUILD := n
```

By default, buildroot will use a cache to store its most commonly compiled functions. This improves repeated compilation time at the cost of disk space. If disk space is at a premium, you can set a cap to this by appending this to the `EXTRA_OPTS` line:

```
EXTRA_OPTS  :=  BR2_CCACHE_INITIAL_SETUP="--max-size=50G"
```

Download sources

You can download sources from Internet before compiling by running the following command, assuming you started with x86_64:

```
cd output/x86_64
make source
```

Personally, I even run it during 10 minutes and then run make in parallel.

Follow the current compilation
If you want to check on the current status for your x86_64 build:

```
tail -f output/x86_64/build/build-time.log
```

**What isn't compiled yet?**

If you ran make source, you can easily find what's remaining to compile by running:

```
for i in output/x86_64/build//*; do test -d "$i" && test -e "$i"/.stamp_built || echo "$i"; done; for i in output/x86_64/build//*; do test -d "$i" && test -e "$i"/.stamp_built || echo "$i"; done | wc -l
```

**Sharing a single download folder among all builds**

```
docker run -it --rm -v $PWD:/build -v $HOME/dev/batocera/DL:/build/dl batocera-docker
```

**List docker images**

```
docker image ls
```

**Remove docker images**

```
docker rmi <image_name> or docker image rm <image_name>
```

**List docker containers**

```
docker ps
```

**Remove docker containers**

```
docker kill [container name]
```

**Folder structure and other tips**

Follow [this page](https://wiki.batocera.org/) to know more about the structure of the Batocera source code and where your modifications need to be made.

**qt5base compilation error in ubuntu docker**
Add `--security-opt seccomp:unconfined` to your docker command line or update the `libseccomp2` package.

As of February 2020, it looks like this step is not required any longer.

### Compiling Batocera inside of an LXC container

Refer to the [Compiling on LXC containers page](https://wiki.batocera.org/) for info on how to compile Batocera inside of a LXC container.

### Skip compiling a certain package

Sometimes during compilation you may come across a package that won't compile, but you want to check for if a future package is okay or not. You can skip any partially built package by creating its **stamp files** to pretend like it was successfully built anyway.

For a more permanent (and somewhat easier to use) skip, a package can be skipped in compilation by removing its mention from the `Config.in` file for the primary package responsible for it.

For instance, to skip over the PCSX2 standalone emulator and its libretro core from compiling, comment out its respective lines in `package/batocera/core/batocera-system/Config.in`.

```
#select BR2_PACKAGE_PCSX2                        if BR2_PACKAGE_BATOCERA_TARGET_X86_64
#select BR2_PACKAGE_PCSX2_AVX2                   if BR2_PACKAGE_BATOCERA_TARGET_X86_64
#select BR2_PACKAGE_LIBRETRO_PCSX2               if BR2_PACKAGE_BATOCERA_TARGET_X86_64
```

### Troubleshooting

### Docker doesn't have permissions to begin building

If using Docker to compile, you must **add your user** to the docker group before being able to compile.

In case it's not possible to do that for your user, a temporary workaround is to simply run the command with elevated privileges: `sudo make x86_64-build`. This is not recommended for ordinary uses, for obvious reasons.

### Ignoring batocera.mk

If `batocera.mk` is being tracked in your local git repo, the following command may be used to explicitly ignore it:
Compilation stops early when trying to apply the Linux kernel patches

This usually happens when the current Linux kernel in the folder is incomplete or was halted during compilation. To fix it, run the following:

```
make x86_64-shell
rm -rf build/linux-*
```

This can also be used to “repair” packages that were interrupted during their compilation, such as you know if the power got cut out by a thunderstorm or something.

Compilation stops at a random time

This may happen at any point during compilation, but usually is related to a specific emulator. Hidden .stamp files are placed into each package's output/build/<package name>-<package version> folder, indicating which step was reached before the failure. Available stamps include download, extract, patch, configure, built, staging_install, target_install. Not all packages use all stamps. Once you've recognized and solved the issue, the entire package can be set to be rebuilt by deleting the respective build folder.

For example, if bsnes was the emulator causing the compilation issues, the built and target_installed stamps can be applied like so:

```
touch ~/.batocera.linux/output/x86_64/build/libretro-bsnes-hd-0fd18e0f5767284fd373aebd75b00b5bab0d44a9/*.stamp_built,.stamp_target_installed
```

If that goes to its end, then it can be recompiled with the following actions:

```
rm -rf ~/.batocera.linux/output/x86_64/build/libretro-bsnes-hd-0fd18e0f5767284fd373aebd75b00b5bab0d44a9
make libretro-bsnes-hd
```

This can be used to determine if it is a local issue with that package or a global issue with Batocera.

Compilation stops at a random time and even after using stamps I can't determine why or how to fix it

It is possible that the downloaded code has been corrupted, or outdated. This can be remedied by performing a clean build.

```
make <target>-cleanbuild
```
A package no longer exists at its previous URL address/I get a 404 error

This can be temporarily worked around by finding the culprit package and downloading it manually to the dl/<package name> folder. For instance, if xa-2.3.11.tar.gz were returning HTTP request sent, awaiting response... 404 Not Found, then the xa-2.3.11.tar.gz could be manually downloaded and placed into batocera.linux/dl/xa:

```
batocera.linux/
├── dl/
│   ├── xa/
│       └── xa-2.3.11.tar.gz
```

After compilation a bunch of .po files get modified, I didn't touch them!

This can happen if someone merged the master branch without first compiling, meaning the PO files get outdated. They are updated once compilation is run. It's usually safe to ignore these, if your addition has no new strings then feel free to discard all batocera-es-system.po and batocera-es-system.pot files. Or include them, they'll just get updated once master gets compiled again anyway.

However, one caveat, if master has been merged without being compiled, new strings were added and a translator was really on point and has already provided a translated line for it, then your updated PO file would overwrite their translated line. In this specific situation, it's no longer “feel free to discard” but “you must discard” in the paragraph above.

My build completes successfully, but the changes I've made aren't appearing

If there were no version changes to the package, and the folder for it exists in output/<arch>/build/<package>, then Buildroot will skip over rebuilding it, using the older (unmodified) version. Simply remove that folder to force it to rebuild the package.

For instance, if you've added a new system which involved edits to es_systems.yml, then you would remove the build/batocera-configgen folder.