Notes on an in-place migration from Gentoo/i686 to multilib Gentoo/x86_64

Johan Hattne • *e*-mail: auzzie@yodel.net • Revised: 2011-03-04

This document attempts to summarise one of several possible ways to migrate from a 32-bit system to a 64-bit ditto. It is nowhere claimed that this is how it is supposed to be done—because it is not. Hence, this text is *not* a how-to; it may, however, be a how-*not*-to. The sane way is undoubtedly a fresh install. This insane way was chosen because it seemed more fun. All applicable disclaimers apply: the procedure outlined here gives the reader ample opportunity to break every aspect of a fully functional system. Anyone not confident in their ability to recover from taking any bad advice given here should not start following it. In the **words of bsdmaster**: "you've been warned".

Introduction

The starting point is an up-to-date i686 Gentoo system, *i.e.* neither `emerge -1DNu @world` nor `revdep-rebuild -i` does anything and `uname -m` outputs `i686`. It is assumed that the kernel does not need any external modules to boot the system and that any architecture-dependent binary data, such as database files, are dumped to some portable format before starting the migration.

Before going any further, a backup directory is created. The backups taken during this exercise will certainly not allow for a full system recovery, and are thus not a substitute for a proper backup. The backup directory should reside on the root partition, because it will contain copies of essential system libraries which may be required during boot before any other partitions have become available. Since it is hard to predict exactly how the system will be broken in case something goes wrong, the reader is largely expected to know how to use contents of this directory if a recovery becomes necessary.

```
# mkdir /migration-backup
# chmod 1777 /migration-backup
```

Because some steps can be performed by a regular, unprivileged user, the permissions of the backup directory grant write access to everybody. Note that in this text `$` denotes the prompt of any unprivileged user on the system. Certain shells may use `%` or `>` for that purpose. The user prompt is distinct from `#`, which denotes the prompt of the superuser.

Build a 64-bit cross compiler

The first preparatory step in the transition to a 64-bit system is the installation of a cross compiler—here, a `gcc` that runs on an i686 system but creates executable code for an x86_64 system. These steps are adapted from Ortwin Glück's blog post **Migration from x86 to x86_64**. Indeed, Ortwin's post, as well as an unplanned hardware upgrade, was the inspiration for this project.

```
# cp -a /etc/make.conf /migration-backup
# emerge -1 crossdev
# crossdev -S -s3 --target x86_64-pc-linux-gnu
```

This will install a 64-bit development environment—a C compiler, the kernel headers, and a standard C library—to build and link x86_64 objects on an i686 system. A C++ compiler is not required. A standard local overlay may need to be set up prior to installing the cross-compiler.

```
# mkdir /usr/local/portage
# echo PORTDIR_OVERLAY=/usr/local/portage >> /etc/make.conf
```

The sources necessary to build `sys-devel/binutils` and `sys-devel/gcc` will be needed later, but should now be available in `/usr/portage/distfiles`. If not, they can be fetched separately.

```
# emerge -f sys-devel/binutils sys-devel/gcc
```

Build and boot a 64-bit kernel

If directory permissions allow, a 64-bit kernel can be configured and built by an unprivileged user. In the commands below, mind the trailing dash after `gnu` in the definition of `CROSS_COMPILE`.

```
$ cd /usr/src/linux
$ cp -a .config /migration-backup/kernel32.config
$ make ARCH=x86_64 oldconfig
$ make ARCH=x86_64 CROSS_COMPILE=x86_64-pc-linux-gnu-
```

Make sure to say yes to `IA32 Emulation`, otherwise the new kernel will not be able to run anything from the current 32-bit userland. Then install and boot the new kernel. The exact commands depend on the configuration of the bootloader and the layout of the filesystem. Here, the following commands were used:

```
# make ARCH=x86_64 CROSS_COMPILE=x86_64-pc-linux-gnu-
modules_install
# mount /boot
# cp -a /boot/kernel /boot/kernel.old
# cp -a /boot/System.map /boot/System.map.old
# cp -a arch/x86/boot/bzImage /boot/kernel
# cp -a System.map /boot/System.map
# umount /boot
# reboot
```

When the system is back up, `uname -m` should output `x86_64`.

Prepare for a 64-bit userland

In order to start replacing the 32-bit userland with its 64-bit equivalent, some preparations are in order. First, edit `/etc/make.conf` and make the following changes:

- Ensure `CFLAGS` and `CXXFLAGS` are conservative, *e.g.* `"-O2 -pipe"`.
- Change `CHOST` to `x86_64-pc-linux-gnu`.
- If present, set `ACCEPT_KEYWORDS` to `amd64` or `~amd64`.
- Add `-sandbox` to the `FEATURES` variable. The currently installed `sys-apps/sandbox` package does not understand 64-bit binaries, so it will not work until it can be rebuilt with a multilib compiler.
- Comment out any `LINGUAS` and `USE` definitions. The purpose of this change is merely to reduce the number of dependencies while rebuilding the userland later on.

Enable a 64-bit multilib profile. Note that the location of the most appropriate profile may differ in more recent versions of the portage tree.

```
 # cd /etc
 # mv make.profile /migration-backup
 # ln -s ../usr/portage/profiles/default/linux/amd64/10.0
make.profile
```

Back up the current 32-bit libraries, and instruct the dynamic linker to consider the backup locations when resolving symbols at runtime.

```
 $ cp -ar /lib /migration-backup/lib32
 $ cp -ar /usr/lib /migration-backup/usr_lib32
 # echo
LDPATH=/migration-backup/lib32:/migration-backup/usr_lib32 >
/etc/env.d/99migration
 # env-update
```

Create directories for the 64-bit libraries and populate them with the 64-bit runtime linker and the fundamental libraries from the cross-development environment.

```
 # mkdir /lib32 /lib64 /usr/lib32 /usr/lib64
 # cp -ar /usr/x86_64-pc-linux-gnu/lib64/* /lib64
 # cp -ar /usr/x86_64-pc-linux-gnu/usr/lib64/* /usr/lib64
```

Because the currently installed system header files contain 32-bit specific information, replace them with the corresponding files from the cross-development environment and updated kernel headers.

```
 # cp -ar /usr/include /migration-backup/include32
 # cp -ar /usr/x86_64-pc-linux-gnu/usr/include/* /usr/include
 # emerge -1 sys-kernel/linux-headers
```

Rebuilding the native toolchain

From here on 32-bit packages are going to be replaced by their corresponding 64-bit versions. This may cause all kinds of havoc, and mistakes can lead to the

machine becoming inaccessible. It is advisable to do all that follows in a single session.

Build a minimal native C compiler

The finicky bit is to replace the cross-development environment with native binutils, `gcc` and `libc`. To that end, any libraries depended on by the toolchain need to be present in their 64-bit versions.

```
# emerge -1 zlib
# USE="-cxx -gpm" emerge -1 ncurses
# USE="nocxx" emerge -1 gmp
# CPPFLAGS=-I/usr/include LDFLAGS=-L/usr/lib64 emerge -1 mpfr
```

The `USE`-flags are chosen in order to eliminate any further dependencies. The preprocessor- and linker-flags are customised for the build of `dev-libs/mpfr`, because the current cross-compiler does not search `/usr/include` or `/usr/lib64` by default. Note that this is volatile information: newer versions of the dependencies may sport different `USE`-flags, and newer versions of GCC may have different dependencies altogether. Indeed, `sys-devel/gcc-4.4.5` was stabilised while this text was written up.

While the 64-bit toolchain dependencies are compiling, the build directory for GCC can be prepared. Note again that the version numbers may need to be adjusted.

```
$ mkdir /migration-backup/toolchain-build
$ cd /migration-backup/toolchain-build
$ tar -xjf /usr/portage/distfiles/binutils-2.20.1.tar.bz2
$ tar -xjf /usr/portage/distfiles/gcc-4.4.4.tar.bz2
$ ln -s -t gcc-4.4.4 ../binutils-2.20.1/bfd
../binutils-2.20.1/binutils ../binutils-2.20.1/gas
../binutils-2.20.1/ld ../binutils-2.20.1/opcodes
$ mkdir objdir
```

Once the 64-bit dependencies are in place, the cross-compiler is used to build a minimal, native development environment in the directory previously prepared. For further information on configuring and compiling the GNU compiler collection, see the **Installing GCC** document.

```
$ cd objdir
$ ../gcc-4.4.4/configure --prefix=/migration-backup/toolchain
--with-gmp-include=/usr/include --with-gmp-lib=/usr/lib64
--with-mpfr-include=/usr/include --with-mpfr-lib=/usr/lib64
--disable-bootstrap --enable-languages=c --disable-nls
CC=x86_64-pc-linux-gnu-gcc
$ make
$ make install
```

These steps can be performed by an unprivileged user.

Build a proper native C compiler

Unlike in the standard situation, where the cross-compiler does not interfere with the native build system, the cross-compiler has here been (ab)used as a native compiler. To avoid conflicts when installing the proper native toolchain, it is necessary to remove the cross-development environment and, optionally, the `sys-devel/crossdev` package, before proceeding.

```
# crossdev -C x86_64-pc-linux-gnu
# emerge -c crossdev
```

The `/usr/x86_64-pc-linux-gnu` directory should be recursively removed. Then, install `sys-devel/binutils` and `sys-devel/gcc` through portage using the minimal, native compiler

```
# CC=/migration-backup/toolchain/bin/x86_64-unknown-linux-gnu-gcc
USE=-nls emerge -1 binutils
# CC=/migration-backup/toolchain/bin/x86_64-unknown-linux-gnu-gcc
USE=-nls emerge -1 gcc
```

Now, only `sys-libs/glibc` is missing for the the new 64-bit development environment to be complete.

From lib to multilib

The next step is to rename `/lib` to `/lib32` and to create a symbolic link to `/lib64` in its place. Until now, there *should* not have been any need to worry about the 32-bit executables—as long as they can find their libraries they *should* remain functional. Copies of any 32-bit libraries that were removed from their original locations by the previous steps *should* be present in `/migration-backup` and the runtime linker *should* find them there. However, all 32-bit dynamic executables will be broken as soon as `/lib` does not contain a working 32-bit runtime linker. Hence, in order complete the switcheroo of the `/lib*` directories—which critically depends on functional `mv(1)` and `ln(1)` commands—a 64-bit version of `sys-apps/coreutils` is necessary.

```
# USE="-acl -nls" emerge -1 coreutils
```

The next few steps may introduce massive—albeit temporary—breakage: swap the `/lib` and `/usr/lib` directories, preserve the 32-bit runtime linker and standard C library, adjust the GNU ld-script in `/usr/lib32/libc.so`, and migrate the kernel modules. In principle, the contents of `/lib` and `/usr/lib32` may just be moved over to `/lib32` and `/usr/lib32` respectively, but the imminent reinstallation of `sys-libs/glibc` and `@world` will replace them anyway.

```
# cp -a /usr/lib/libc[-._]* /usr/lib32
# sed -i -e "s:/lib/:/lib32/:g" /usr/lib32/libc.so
# mv /lib/ld-* /lib/libc[-._]* /lib32
# mv /lib/modules /lib64
# mv /lib /lib.old
# cd /
# ln -s lib64 lib
# ln -s -t lib64 ../lib32/ld-linux.so.2
```

A multilib standard C library

Finally, reinstall the standard C library. The `COLLISION_IGNORE` variable is set because the several files belonging to `sys-libs/glibc` were manually moved into place outside portage's control.

```
# COLLISION_IGNORE="/lib32 /lib64 /usr/include /usr/lib32
/usr/lib64" USE=-nls emerge -1 glibc
```

Just like during an ordinary change of the `CHOST` variable, the environment may need to be cleaned up, see the **Changing the CHOST variable** document for more information.

```
# mv /usr/i686-pc-linux-gnu /migration-backup
# rm -f /etc/env.d/05gcc-i686-pc-linux-gnu
/etc/env.d/binutils/config-i686-pc-linux-gnu
/etc/env.d/gcc/config-i686-pc-linux-gnu
# env-update && source /etc/profile
```

Eventually `grep -r i686 /etc/env.d` should not find any matches, but there *should* be files for the updated `CHOST`, *i.e.* `x86_64`.

Rebuild a 64-bit system

Optional: A 64-bit `login(1)`

The system will remain largely broken until everything is rebuilt. Notably, PAM will not work, because it is searching for 32-bit modules modules in the `/lib/security` directory. While that means that the machine has become completely inaccessible, one may work around that by installing `sys-libs/pam`, `sys-apps/shadow` and their dependencies before proceeding.

```
# USE=-nls emerge -1 flex
# USE="-berkdb -cracklib -nls" emerge -1 pam
# USE="-cracklib -nls" emerge -1 shadow
```

Now, the new `login` program will make use of the new PAM modules in `/lib64/security`. Access through `ssh` may require additional work.

Rebuild `@world`

Now only remains to rebuild the entire system in 64-bit mode. Even though straightforward, some packages will probably fail because their dependencies are of the wrong architecture. One approach is to run

```
# emerge -1e --keep-going @world
```

and then manually rebuilding skipped packages and packages that have residues left in `/var/tmp/portage`. `sys-apps/portage` may fail during the `postrm` and `postinst` phase, which means that the contents of `/usr/lib/portage` were not properly removed. That can be done manually by

```
# mv /usr/lib/portage /tmp/migration-backup
```

Endgame

Edit `/etc/make.conf`, reverting some of the changes introduced earlier.

- Set `CFLAGS` and `CXXFLAGS` to something appropriate.
- Remove `-sandbox` from the `FEATURES` variable.
- Remove the definition of `PORTDIR_OVERLAY` if appropriate.
- Uncomment any commented `LINGUAS` and `USE` definitions.

The contents of the `/lib.old` directory should now be replicated in `/lib32` and `/lib64`. The `/usr/lib` directory should be mostly empty. Once confident that the library directories `/lib32`, `/lib64`, `/usr/lib32`, and `/usr/lib64` and /usr/i686-pc-linux-gnu are complete, the redundant directories can be removed.

```
# mv /lib.old /migration-backup
# cd /usr
# mv lib/gcc lib64/gcc
# mv lib /migration-backup/usr_lib
# ln -s lib64 lib
```

Remove the backup directories from the dynamic linker's search path,

```
# rm /etc/env.d/99migration
# env-update && source /etc/profile
```

restore any backed-up databases, and verify link consistency,

```
# revdep-rebuild -i
```

Since the `USE`-flags have changed, an `emerge -1DNu @world` is now in order. However, because the system has gone through extensive changes and the `CFLAGS` may have changed substantially, an `emerge -1e @world` may actually be preferred in order to verify that all is well. Once convinced that it is so, remove the `/migration-backup` directory, and that will be the end of this text.