

## Multilib Slackware for x86\_64

This article contains instructions on how to create a *true multilib* Slackware64. A multilib 64bit Linux system is capable of running 64bit as well as 32bit software. The Filesystem Hierarchy Standard dictates the best way to achieve a clean separation between 64bit and 32bit software on a single system. With Slackware64 we chose to adopt this standard, so it has been prepared to look for 64bit libraries in `/lib64` and `/usr/lib64` directories. This is why I call Slackware64 “multilib-ready” - even though 32bit libraries will be looked for in `/lib` and `/usr/lib`, Slackware64 does not ship with any 32bit software. There is one more step to take before Slackware64 can be called “multilib-enabled”.

This is accomplished as follows:

1. First we need to switch to multilib versions of
  - *glibc* (i.e. a *glibc* that supports *running* both 32bit and 64bit binaries), and
  - *gcc* (i.e. able to *compile* 32bit binaries as well as 64bit binaries).
2. Then, system libraries are taken from 32bit Slackware and installed in the 64bit Slackware system which completes the process of creating a 32bit software layer.

Slackware for the x86\_64 architecture (or “*Slackware64*” for short) is a pure 64-bit Operating System, but easily upgradable to multilib. *Out of the box, Slackware64 is only capable of compiling and running 64bit binaries.*

Slackware64 has an advantage over the 64bit “forks” that exist out there. These forks add the 32bit compatibility layer by recompiling a lot of their packages as 32bit binaries. Slackware on the other hand, is a distribution that consists of a 32bit and 64bit version which are being developed in parallel. This means, that you do not have to compile 32-bit packages from scratch in order to add multilib capability to the 64bit system. You simply take them from the 32-bit Slackware package tree. This was one of the reasons for not adding full multilib to Slackware64 - we create the right preconditions but require the user to act if she needs multilib.

In a section further down, I will explain how you can take a 32-bit Slackware package (say, the “*mesa*” package) and re-package its content into a “*mesa-compat32*” package that you can install straight away on Slackware64.

## Advantage of a multilib system

I will give some examples of programs that require multilib support on a 64bit Slackware because they will not start or compile on Slackware64 without the 32bit compatibility layer:

- *Wine*  
Most Windows programs are still 32bit, and in order to run those on Linux with *Wine*, you need a 32bit version of *Wine*.
- *VirtualBox*  
The popular virtual machine software. Although this is (partly) open source it still needs 32-bit compatibility libraries on 64-bit Slackware.
- *Skype*, *Citrix client*, ...  
These programs are proprietary and closed-source. We have to depend on the developer to make 64bit binaries available. So far, that has not happened for these example programs.

Luckily, 64bit support is becoming more and more common. In the past year, Adobe released their Flash

browser plugin in a 64bit version and Sun revealed a 64bit version of their Java browser plugin. This was one of the triggers to start working on Slackware64.

## Obtaining multilib packages

You can download a set of multilib-enabled packages and scripts from my web site:  
<http://slackware.com/~alien/multilib/> .

The packages are accompanied by sources and SlackBuild scripts, plus several README files (this Wiki article is based on one of these READMEs). The *only* required downloads are the binary packages in `<slackware_release_number>` directory below the toplevel directory. The rest is available for educational purposes.

## Enabling multilib support on Slackware64

### The quick 'n' dirty instructions

This section contains the essential instructions to add full multilib capability to your Slackware64 system. If you want to understand the process in more detail, or need information on how to compile 32bit software in Slackware64, read the next sections as well.

- After downloading the packages from my web site (I gave you the URL in the previous section), you upgrade your 64bit Slackware "gcc" and "glibc" packages to my multilib versions. Run the command

```
upgradepkg --reinstall --install-new *.t?z
```

in the directory where you downloaded them.

This command will also install an additional package called "compat32-tools".

- You must have a 32-bit Slackware package tree available. Those who bought an official Slackware 13.0 DVD can simply use that DVD: it is dual-sided and 32bit Slackware is on one of the sides. For the sake of this example I will assume that you have a local 32bit Slackware directory tree available at `"/home/ftp/pub/slackware/slackware-13.0/slackware/"`.

There should be sub-directories called 'a', 'ap', 'd', 'l', 'n', 'x' immediately below this directory. (If you have mounted a Slackware DVD, your directory will probably be `"/media/SlackDVD/slackware/"` but I will not use that in the example commands below).

- Create a new empty directory (let us call it 'compat32') and change into it:

```
mkdir compat32 ; cd compat32
```

- Run the following command to create a set of 32bit compatibility packages, using the directory to the official 32bit Slackware packages as input:

```
massconvert32.sh -i /home/ftp/pub/slackware/slackware-13.0/slackware/
```

- The previous step takes a while. When it ends, proceed to install the 60 MB of freshly converted

32-bit Slackware packages which were created in subdirectories below your *current directory*:

```
installpkg *-compat32/*.t?z
```

- Done! You can now start downloading, installing and running 32bit programs. This was not so hard, was it?

If you use a package manager like *slackpkg* you will have to add all *glibc* and *gcc* package names to its package blacklist. If you do not take this precaution, you run the risk of your package manager accidentally replacing your multilib versions with Slackware's original pure 64-bit versions!

## Detailed instructions

### Upgrading glibc and gcc

The following *glibc/gcc* packages are replacements for - not additions to - standard Slackware packages. You use the “*upgradepkg*” program to upgrade to my multilib versions of *gcc* and *glibc*. You will need these in order to run (*glibc*), and build (*gcc*), 32-bit software on your 64-bit Slackware computer (the package versions shown below are for Slackware 13.0):

### Slackware64 13.0

- The *gcc* compiler suite:
  - *gcc-4.3.3\_multilib-x86\_64-4alien.txz*
  - *gcc-g++-4.3.3\_multilib-x86\_64-4alien.txz*
  - *gcc-gfortran-4.3.3\_multilib-x86\_64-4alien.txz*
  - *gcc-gnat-4.3.3\_multilib-x86\_64-4alien.txz*
  - *gcc-java-4.3.3\_multilib-x86\_64-4alien.txz*
  - *gcc-objc-4.3.3\_multilib-x86\_64-4alien.txz*
- The GNU *libc* libraries:
  - *glibc-2.9\_multilib-x86\_64-3alien.txz*
  - *glibc-i18n-2.9\_multilib-x86\_64-3alien.txz*
  - *glibc-profile-2.9\_multilib-x86\_64-3alien.txz*
  - *glibc-solibs-2.9\_multilib-x86\_64-3alien.txz*
  - *glibc-zoneinfo-2.9\_multilib-noarch-3alien.txz*

### Slackware64 13.1 (current)

- The *gcc* compiler suite:
  - *gcc-4.4.2\_multilib-x86\_64-1alien.txz*
  - *gcc-g++-4.4.2\_multilib-x86\_64-1alien.txz*
  - *gcc-gfortran-4.4.2\_multilib-x86\_64-1alien.txz*
  - *gcc-gnat-4.4.2\_multilib-x86\_64-1alien.txz*
  - *gcc-java-4.4.2\_multilib-x86\_64-1alien.txz*
  - *gcc-objc-4.4.2\_multilib-x86\_64-1alien.txz*
- The GNU *libc* libraries:
  - *glibc-2.11.1\_multilib-x86\_64-1alien.txz*
  - *glibc-i18n-2.11.1\_multilib-x86\_64-1alien.txz*
  - *glibc-profile-2.11.1\_multilib-x86\_64-1alien.txz*

- `glibc-solibs-2.11.1_multilib-x86_64-1alien.txz`
- `glibc-zoneinfo-2.11.1_multilib-noarch-1alien.txz`

There is one additional package that you install using the “installpkg” program:

- The “32-bit toolkit” (scripts that facilitate the creation of 32bit packages)
  - `compat32-tools-1.0-noarch-18.tgz`

Slamd64 has separate 64bit and 32bit gcc/glibc multilib packages.

However, I believe that it is cleaner to keep these essential multilib packages undivided. I followed the concept already used in Slackware64's own *binutils* package, which has 64-bit and 32-bit multilib capability bundled into one package.

### Adding 32-bit Slackware libraries

The upgrade of glibc and gcc which I described in the previous section changes your system from “*multilib-ready*” to “*multilib-enabled*”.

Now, all you need to do is to install 32bit versions of Slackware's system software so that future 32bit programs that you are going to install and/or compile will find all the 32bit libraries they need in order to work.

This is not as simple as grabbing 32bit Slackware packages and installing them in Slackware64:

- In the first place, you will end up with multiple packages carrying the same name (two 'mesa' packages, two 'zlib' packages, etc...) which will be confusing to you as well as to the *slackpkg* package manager.
- And furthermore, if the 32bit package contains binaries (something like `/usr/bin/foo`), they will overwrite their 64bit counterparts when you install the 32bit package on top. It will seriously mess up your system if that happens.

A little bit of extra care is required so that unnecessary/unwanted files are stripped from the 32bit packages before you install them. What you need, is a 32bit package that does not conflict with whatever is already present in 64bit Slackware. Hence the name “32bit compatibility package”.

I decided that it would be a waste of download bandwidth if I created 32bit compatibility versions of Slackware packages myself. After all, you have probably bought the Slackware 13.0 DVD so you already possess both 64bit and 32bit versions of Slackware... or else the 32bit Slackware tree is available for free download of course 😊

Instead, I wrote a few scripts (parts of the script code were written by Fred Emmott of Slamd64 fame) and wrapped these into a “*compat32-tools*” package. Their purpose is to let you extract the content from any 32bit Slackware package and use that to create a new package which you can safely install on your 64bit Slackware.

This “*compat32-tools*” package needs some explanation.

Please read the detailed ‘*README*’ file in the `/usr/doc/compat32-tools-*/` directory, it will help you on your way. These are the three useful scripts which the package installs:

- `/etc/profile.d/32dev.sh`  
This is the same script that comes with Slamd64. It reconfigures your shell environment so that it will be easier for you to compile 32-bit software (by preferring the 32-bit compilers and libraries

over their 64-bit versions)

- *convertpkg-compat32*

This script takes a 32-bit Slackware package and converts it to a '-compat32' package that you can safely install (using "installpkg") on Slackware64, alongside the 64-bit version of the same software package. For instance: suppose you need 32bit libraries that are in the mesa package. You take the mesa package from 32-bit Slackware (x/ mesa - 7.5 - i486 - 1 . t x z) and then run

```
convertpkg-compat32 -i /path/to/ mesa - 7.5 - i486 - 1 . t x z
```

which will create a new package called `mesa - compat32 - 7.5 - x86_64 - 1 . t x z`. This new package (which is created in your `/tmp` directory unless you specified another destination) is basically the old 32bit package, but stripped from non-essential stuff. The changed basename (*mesa* becomes *mesa-compat32*) allows you to install this new package in Slackware64 where it will co-exist with the 64bit *mesa* package, not overwriting any files.

The script leaves temporary files in the directory `"/tmp/package - <prgnam> - compat32"` which you can safely delete.

- *massconvert32.sh*

This script contains an internal list of what I consider the essential subset of 32-bit Slackware packages. It uses the above "*convertpkg-compat32*" script to grab every package that is on this internal list, and convert these into '-compat32' packages.

You need to run this script only once, for example like this (the example assumes that you mounted your 32bit Slackware DVD on `/mnt/dvd`):

```
massconvert32.sh -i /mnt/dvd/slackware -d ~/compat32
```

This action will result in about 60 MB of new packages which you will find inside the newly created directory `~/compat32` (the directory's name is arbitrary of course, I chose it for the sake of this example). These packages comprise the 32bit component of your multilib Slackware64 system. They should be installed using "installpkg", and they give you a pretty complete 32-bit compatibility layer on top of Slackware64:

```
installpkg ~/compat32/*/*.t?z
```

When installing the *compat32* packages you will notice that some will show errors about missing files in `/etc`. This is "by design", and these errors can be ignored. These messages are caused by the fact that files in `/etc` are removed from a "-compat32" package during conversion (except for *pango* and *gtk+2*). I assume that files in `/etc` will already have been installed by the original 64bit packages. An example of these "errors" for the `cups - compat32` package:

```
Executing install script for cups-compat32-1.3.11-x86_64-1.txz.
install/doinst.sh: line 5: [: too many arguments
cat: etc/cups/interfaces: Is a directory
cat: etc/cups/ppd: Is a directory
cat: etc/cups/ssl: Is a directory
cat: etc/cups/*.new: No such file or directory
cat: etc/dbus-1/system.d/cups.conf.new: No such file or directory
chmod: cannot access `etc/rc.d/rc.cups.new': No such file or directory
cat: etc/rc.d/rc.cups.new: No such file or directory
Package cups-compat32-1.3.11-x86_64-1.txz installed.
```

If you were considering to use the `convertpkg-compat32` script to convert a **non-Slackware** package to a `-compat32` package, I must strongly advise against this. The script is written with a single purpose and that is to make 32bit versions of the official Slackware64 binaries/libraries available in a multilib setup. As such, the script will remove a lot of stuff that is present in the original 32bit package - stuff which is expected to have been installed as part of the 64bit version of the package. In almost all cases where you have downloaded a non-Slackware 32bit package and want to make it work on Slackware64, the best way is to find the sources and build a 64bit version of the package. Alternatively, just *install the original* 32bit package instead of trying to “convert it” and then run it from the commandline to find out any missing 32bit libraries you may still have to extract from an official Slackware package.

## Running 32-bit programs

Running a pre-compiled 32-bit program is easy after you've done the above system preparation. Just download, install and run it!

At times, you may run into a program that requires a certain 32-bit Slackware library that you do not yet have available. In that case, find out which 32bit Slackware package contains this missing library. Use the “`convertpkg-compat32`” script to convert that original 32bit Slackware package and install the resulting 32bit “*compatibility*” package on Slackware64.

## Compiling 32-bit programs

In case you need to compile a 32-bit program (wine and grub are two examples of open source programs that are 32-bit only) you first configure your shell environment by running the command:

```
./etc/profile.d/32dev.sh
```

Note the 'dot' in front of the filename - that is actually part of the commandline! Running this command changes or creates several environment variables. The effect of this is, that 32-bit versions of binaries are preferred over 64bit binaries when you compile source code - you will be running a 32bit compilation. The effect will last until you logout from your shell.

In this changed environment, you will be able to use standard SlackBuilds to build 32-bit packages for Slackware64. There are two things to keep in mind:

1. You will have to define the ARCH variable as 'x86\_64' even though you are compiling a 32-bit program!  
This is related to the *triplet* of “\$ARCH-slackware-linux” which is normally used in the “configure”

command. Also, try setting the ARCH to for instance "i486" and you will see that the *CFLAGS* definition for that architecture will result in gcc errors like "*compiler can not create executables*". This is related to the design of a SlackBuild script. Rather than editing the script and change/remove *CFLAGS* definitions, you can set the ARCH to "x86\_64" and save yourself some time. The real work is being done by the 32dev.sh script.

2. You will have to edit the SlackBuild if it wants to use 'lib64/' directories for "\$ARCH = x86\_64". You have to force it to use 'lib/' directories instead. Usually, this is accomplished by finding a definition like:

```
LIBDIRSUFFIX="64"
```

and changing this line to

```
LIBDIRSUFFIX=""
```

## Caveats

After installing the "-compat32" packages, you may have to re-install your binary *Nvidia* or *Ati* video X.Org drivers. These driver packages contain both 64bit and 32bit libraries to be maximally useful on a 64bit multilib OS. If you installed the driver files for both architectures, the "mesa - compat32" package will overwrite some of the 32bit library files.

On the other hand, if you originally *only* installed the 64bit driver libraries for your Nvidia/Ati card, it is recommended after installation of the *multilib* packages, to re-install the binary driver package. This time, choose to install the 32bit driver files as well.

The graphical 32bit applications that you are going to run on your multilib installation will require these 32bit driver libraries. Crashes are likely to occur if you fail to install the correct files.

## Packages converted by massconvert32.sh

This is the list of packages that is converted into "-compat32" versions by the `massconvert32.sh` script. Note that some of these packages are not part of Slackware 13.0, they were added in Slackware 13.1 so they will produce a "\*\*\* FAIL: package 'package\_name' was not found!" message when you run the script. Don't worry about that.

```
# The A/ series:
|bzip2
|cups
|cxxlibs
|dbus
|e2fsprogs
|openssl-solibs
|util-linux-ng

# The AP/ series:
|mpg123
|mysql

# The D/ series:
|libtool

# The L/ series:
|alsa-lib
|alsa-oss
|atk
|audiofile
|cairo
|dbus-glib
|esound
|expat
|freetype
|gamin
|glib2
|gtk+2
|hal
|jasper
|lcms
|libart_lgpl
|libexif
|libgphoto2
|libjpeg
|libidn
|libmng
|libpng
|libtermcap
|libtiff
|libv4l
|libxml2
|libxslt
|ncurses
|pango
|popt
|qt
|readline
|sdl
|seamonkey-solibs
|svgalib
|zlib

# The N/ series:
|curl
|cyrus-sasl
|gnutls
|libgcrypt
|libgpg-error
```

## Translations

- Bruno Russo translated this article to portuguese (brazil): [http://www.brunorusso.eti.br/slackware/doku.php?id=multilib\\_para\\_o\\_slackware\\_x86\\_64](http://www.brunorusso.eti.br/slackware/doku.php?id=multilib_para_o_slackware_x86_64)

## Acknowledgements

- A lot of thanks should go to Fred Emmott, who created Slamd64, the original unofficial 64-bit fork of Slackware. Although Slackware64 was not based on Fred's work, I still learnt most of what I know about setting up the 32-bit part of a multilib Linux from his writings that are found in Slamd64.
- Cross Linux From Scratch.  
The CLFS Wiki (<http://trac.cross-lfs.org/wiki/read#ReadtheCrossLinuxFromScratchBookOnline>) is a 'must-read' if you want to understand how to port Linux to a new architecture. I took several ideas, concepts and patches from them when creating Slackware64 from scratch, and again when I created my multilib gcc/glibc packages from scratch (my README on this multilib-from-scratch is available in the `./source` directory).

Have fun!

Eric