Upgrading Slackware to a New Release

Upgrade or Install from Scratch

Installing Slackware fresh and from scratch is always the best method if you are running a fairly old release of Slackware and want to skip a few releases. Too many intrusive changes to the distro will have occurred if your Slackware is relatively old. It will make a manual upgrade process painful and will not guarantee success.
It is better in such cases to make a backup of your package list ("`ls -lart /var/log/packages`"), a backup of your "`/etc`" directory and (you did this already of course) a backup of your personal data. Formatting your disk and installing from the Slackware boot medium, and an hour later you could already be back in business.

If you want to upgrade to the next Slackware release, you can do this manually by following the instructions in the file "`UPGRADE.TXT`" which you will find in the root of the Slackware DVD/CD1. Advanced instructions and further hints can be found in "`CHANGES_AND_HINTS.TXT`" in the same location. There is also a semi-automated procedure for this type of upgrade, using slackpkg. It will take away a lot of the hard labour from you.
I *always* use slackpkg (with care) to upgrade my Slackware systems from one stable release to the next. I call this process a "*system upgrade*".
You can use the same procedure to migrate to slackware-current, keep a slackware-current system up to date, or upgrade from a slackware-not-so-current to a freshly released stable version.

Considerations about the Kernel

Just running `slackpkg` and hoping for the best is not going to work. Some considerations have to be cared for. One important thing to remember:


Never upgrade your working kernel.

Why is that? Simple - you will be upgrading potentially hundreds of packages and should be prepared for the unlikely event that your computer does not work properly anymore after a system upgrade. One thing you don't want to get hit by is a system which does not boot at all. A new Slackware release may install a kernel that refuses to boot your computer (small chance but nevertheless... be prepared). For that reason, you need to keep your "old" working kernel installed, and keep a section for it in your `/etc/lilo.conf` file. That way, if the new kernel fails to boot, you can fall-back to the old kernel and start investigating what went wrong.
Basically, these are the same precautions you must take when you are compiling a new kernel yourself.

Video driver considerations

If your computer is equipped with a video card powered by an Nvidia or Ati graphics processor and

you have installed these companies' accelerated graphics drivers (closed-source and binary-only), you should not attempt start an X session after upgrading to the next Slackware release.
These drivers depend on kernel version, Mesa version and the X.Org server. You must re-install the binary driver before starting graphical mode. Also, the mesa and xorg-server packages of Slackware overwrite essential files of these closed-source accelerated graphics drivers anyway.

If you want to know how to deal with these binary drivers, we have more detailed instructions in the "Proprietary Graphics Drivers" article on this Wiki.

Slackpkg considerations

If you upgrade Slackware (see below for the procedure), you will upgrade slackpkg as one of the first steps. The `upgradepkg` command will install a file `/etc/slackpkg/mirrors.new`. This is the file which contains the URLs for mirrors carrying the new Slackware release. You will have to compare this to the original version and merge the contents.

*Be sure to have exactly **one line** uncommented which points to a mirror for the desired Slackware release version and architecture.*

System Upgrade using SlackPkg

The following steps should work for all situations:

- Blacklist the following kernel packages in "`/etc/slackpkg/blacklist`":

```
kernel-generic
kernel-generic-smp
kernel-huge
kernel-huge-smp
kernel-modules
kernel-modules-smp
```

  This will prevent an accidental upgrade of your working kernel.

- Blacklist packages from 3rd party repositories by adding appropriate lines for their *repo tags*. Examples for SlackBuilds.org, AlienBOB and multilib:

```
[0-9]+_SBo
[0-9]+alien
[0-9]+compat32
```

- If new kernel(s) have been added to the Slackware release you are upgrading to, then use "`installpkg`" to install those new kernel packages first (do not use "`upgradepkg`" because that will wipe your existing kernel).
  You will have to install at least one kernel (kernel-generic, kernel-generic-smp, kernel-huge, or kernel-huge-smp) and the corresponding kernel modules package (kernel-modules or kernel-

modules-smp).
You can not use `slackpkg` for this step.

- Now that we have the new kernel(s) plus modules in place, we can start upgrading the rest of the packages. First, update the `slackpkg` package database:

```
# slackpkg update
```

- When slackpkg has updated its internal database, the first thing to do is to upgrade `slackpkg` itself to the latest version (including the URL's for the new Slackware release and any package upgrade rules that apply to the new release). If you fail to perform this step, `slackpkg` will upgrade itself in the middle of the *system upgrade*, and abort immediately after that...

```
# slackpkg upgrade slackpkg
# slackpkg new-config
```

That final `new-config` command is there so that you can view the difference betweeen your old and the new `slackpkg` configuration files, in particular `/etc/slackpkg/mirrors` and `/etc/slackpkg/blacklist` are files you have to check. Overwriting `/etc/slackpkg/slackpkg.conf` is usually recommended.

- A new Slackware release usually has a newer version of the GNU C libraries. The new packages are compiled against that new `glibc` version. In order to prevent an upgrade failure, you need to upgrade the `aaa_glibc-solibs` package manually, immediately after upgrading `slackpkg`:

```
# slackpkg upgrade aaa_glibc-solibs
```

Let me give an example of such potential failure: when `slackpkg install-new` installs the `libusb-compat` package, your `gpg` command stops working because it links against libusb.so which will be overwritten with the version from the new `libusb-compat` package. The new library needs the new `glibc` package, `gpg` stops working because of the library linking error, and `slackpkg` will stop the system upgrade because it wants to check every package's gpg signature before upgrading it. Upgrading the `aaa_glibc-solibs` package prevents the library linking errors by providing the correct "`GLIBC`" symbols.

In Slackware 14.2 and earlier, the `aaa_glibc-solibs` package was called `glibc-solibs`

- Let `slackpkg` update the computer to the new Slackware release:

```
# slackpkg install-new
# slackpkg upgrade-all
# slackpkg clean-system
```

  - The first of those three commands (`slackpkg install-new`) will install every package which is marked in the Slackware ChangeLog.txt file with the string "`Added.`" This command will **not** install any other packages which are not yet installed.
    For instance if you did not have KDE installed before, the "`slackpkg install-new`"

command will not add KDE packages to your computer all of a sudden.
    - The second command (`slackpkg upgrade-all`) will compare every official Slackware package which you currently have installed, with the package list on your Slackware mirror. If a different version is available, that version will be (downloaded and) upgraded. [1)]
    - The third commmand (`slackpkg clean-system`) will show you an overview of all packages that you currently have installed and are not part of the Slackware Linux release you are upgrading to. This means, the list will show all packages which has been removed from Slackware. Examples for Slackware 14 are `kdeaccessibility`, `kdebase`, ... Another example of such abandoned package in Slackware 14 is `ntfsprogs`. Actually this package was not *removed* but *renamed*... for Slackware that is equal to a package *removal plus addition*.
    The command will also show you every 3rd party package which you have installed! Use this command wisely: you must de-select every package which you want to keep (i.e. all 3rd party packages) and then click "OK" to let `slackpkg` remove all obsolete packages.

- A massive upgrade like this will have installed several "`.new`" files. Some packages contain configuration files which have been renamed (inside the package) with a "`.new`" extension so that an existing configuration file (containing your customizations) will not be recklessly overwritten during the upgrade. Slackpkg will do a check for the existence of these "`.new`" files at the end of an upgrade or install and prompt you to do something with them.
  You are advised to upgrade to the new versions of configuration files where possible, because they will often bring improvements to your software configuration. Slackpkg allows you to view the differences between old and new files and even, to merge the two files. Alternatively you can decide to keep the "`.new`" file, leaving the old file in place, so that you can investigate the differences at a later time if these are too intrusive.
  You can force a check on "`.new`" files at any time, by running the command

```
  * # slackpkg new-config
```

  and using the easy user interface of slackpkg to merge the changes.

- You should probably decide at this moment to use a generic kernel, especially if you are using LVM or RAID, or installed Slackware to a LUKS-encrypted disk. This is also being recommended in the Slackware README on the DVD/CD. On the other hand - if your system setup is straight-forward and your hardware is fairly new, you could decide to stick with a "huge" kernel.
  *Remember that you **cannot** use an initial ramdisk in combination with a "huge" kernel, but you **have** to create a new initial ramdisk if you are going to use a generic kernel!*
  If you are uncertain at this point how that must be accomplished, then you should make sure that you are booting a "huge" kernel which won't require an initial ramdisk.
  However, the "`mkinitrd_command_generator.sh`" script can help you here. Run this script with the *new* kernel version as a parameter and it will show you an example "`mkinitrd`" command which will work for your particular hardware setup and system configuration:

```
# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 3.2.29
```

  will give this as output (the kernel version 3.2.29 is that of Slackware 14)

```
#
# mkinitrd_command_generator.sh revision 1.45
#
```

```
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:

mkinitrd -c -k 3.2.29 -f ext4 -r /dev/sdb2 -m usb-
storage:pcmcia_core:pcmcia:mmc_core:ssb:modprobe:usbhid:ehci-hcd:ohci-
hcd:mbcache:jbd2:ext4 -u -o /boot/initrd.gz
```

You can copy and paste this command line in your console, and let it create an initial ramdisk for you.

If you were already running a generic kernel and therefore already have an initrd, we strongly advise you to create a **new** initrd with a **new** unique name! For instance, you can copy the above example and modify the name of the initrd file as follows:

```
mkinitrd -c -k 3.2.29 -f ext4 -r /dev/sdb2 -m usb-
storage:pcmcia_core:pcmcia:mmc_core:ssb:modprobe:usbhid:ehci-
hcd:ohci-hcd:mbcache:jbd2:ext4 -u -o /boot/initrd_3.2.29.gz
```

- After you decided what kernel you are going to use and have created an initial ramdisk, you must update your "`/etc/lilo.conf`" file with a section for the new kernel (*don't remove your running kernel!*). The "`mkinitrd_command_generator.sh`" script can help you finding the right block to append to `/etc/lilo.conf`. For instance, the command:

```
# /usr/share/mkinitrd/mkinitrd_command_generator.sh -l /boot/vmlinuz-
generic-3.2.29
```

will result in the following output which you can copy/paste:

```
# Linux bootable partition config begins
# initrd created with 'mkinitrd -c -k 3.2.29 -f ext4 -r /dev/sdb2 -m
usb-storage:pcmcia_core:pcmcia:mmc_core:ssb:modprobe:usbhid:ehci-
hcd:ohci-hcd:mbcache:jbd2:ext4 -u -o /boot/initrd.gz'
image = /boot/vmlinuz-generic-3.2.29
  initrd = /boot/initrd.gz
  root = /dev/sdb2
  label = 3.2.29
  read-only
# Linux bootable partition config ends
```

Note that this command adds a "initrd" line to the kernel section. If you let `mkinitrd` create a unique name for your initial ramdisk, then be sure to apply that name in the above section. The "initrd" line is not needed if you are going to run a "huge" kernel.

- Finally, run the "`lilo`" command to make the change permanent and add the new kernel to

the lilo boot menu. You can just run "`eliloconfig`" if you are using EFI after upgrading Slackware and that will itself install the latest kernel on the EFI partition.
Remember, you should always be able to boot back into a previous kernel in case the new Slackware kernel gives you a hard time.

Bottom-line: you can trust slackpkg to perform a system upgrade safely, but it will need your brains and care.

Multilib considerations

If you are upgrading a multilib 64-bit Slackware computer, there are additional considerations to make.

A multilib installation means that you have replaced Slackware's 64-bit `gcc` and `glibc` packages with multilib versions (ie.e. supporting both 32-bit and 64-bit binaries). Also, you have installed a set of "converted" 32-bit Slackware packages on your 64-bit multilib Slackware. These modifications are all needed to allow you to run and compile 32-bit software.

When upgrading such a system, you must of course upgrade the standard Slackware packages, but separately you must upgrade the multilib-specific packages with new versions which you can obtain from http://slackware.com/~alien/multilib/

- First (if you use slackpkg), you must blacklist all these multilib packages so that they will not be accidentally replaced or removed during a system upgrade. If you don't blacklist them, you will face a lot of manual de-selections in "`slackpkg clean-system`".
  Starting with the release of `compat32-tools` for Slackware 14.0 you can simply add two lines to the `/etc/slackpkg/blacklist` file:

```
[0-9]+alien
[0-9]+compat32
```

  Then you will manually have to download and upgrade the multilib packages. In the below example I will use Slackware 14.0 as the release to which you will be upgrading.

- Download the multilib packages suitable for your new Slackware release from a mirror, like this:

```
# rsync -av
rsync://taper.alienbase.nl/mirrors/people/alien/multilib/14.0/
multilib-14.0/
```

  This command will create a new subdirectory "`multilib-14.0`" in your current directory with all packages inside

- install/upgrade the existing `gcc` and `glibc` packages, and `compat32-tools`:

```
# cd multilib-14.0
# upgradepkg --install-new *.t?z
```

- Upgrade the set of converted 32-bit Slackware packages (often referredto as the “*compat32*” packages):

```
# upgradepkg --install-new slackware64-compat32/*-compat32/*.t?z
```

  Alternatively you can run the “`massconvert32.sh`” script which will have been installed as part of the `compat32-tools` package. Pass it a 32-bit Slackware package directory (or a 32-bit Slackware mirror URL) as parameter and that will create a set of converted “`compat32`” packages which you can then install. You would only have to do this if you suspect that the content of the “`slackware64-compat32`” directory is not up to date.

Java considerations

Slackware used to install a *Java Run-time Engine* prior to the 14.0 release (the JRE binaries were originally Sun's and later distributed by Oracle after it bought Sun).
But Oracle changed the re-distribution license so that Slackware (just like all other distributions) was no longer permitted to ship these Java binaries as part of the distribution. When you perform a *system upgrade* to Slackware 14.0, an old version of the *JRE* will stay behind on your system. This version “6u25” has several critical vulnerabilities and you should remove it manually from your computer as fast as possible, using the command

```
removepkg jre
```

If you need Java then please have a look in the “`/extra/source/java`” directory of the Slackware 14 release. You will find a script there to create a Slackware package from the most recent Java software from Oracle, which you can then install using the “`installpkg`” command. See also our Wiki article “Java in Slackware”

Sources

- Originally written by Eric Hameleers

howtos, slackpkg, author alienbob

1)
Slackware's package manager does not work with the concept of “lower version” and “higher version”. The package tools only look at “*different version*” and thus downgrading packages (reverting to an earlier version) is just as easy as upgrading to a newer version!