PKGBUILDER

Antonio Gabriel Muñoz Conejo

Sábado, 14 de Noviembre de 2004

Resumen

Instalación de **PkgBuilder**, gestión y administración del arbol de ports y creación de scripts de construcción de paquetes.

Índice

1. Motivos:

Los motivos por los que nace **PkgBuilder** son varios pero el principal es la escasez de paquetes oficiales que existen en **Slackware**, multitud de paquetes que no se incluyen en la distribución oficial de **Slackware**, tan básicos como pueden ser *gkrellm*, *wine*, o *postfix*. Buscando solucionar esto intenté migrar a otras distribuciones de **GNU/Linux** o incluso migrar a otros sistemas operativos libres como **NetBSD** pero no encontré nada que se ajustara a lo que buscaba. Lo que buscaba era un sistema que me permitiera crear paquetes para **Slackware** a partir de los fuentes sin tenerme que preocupar de nada salvo de ejecutar unos sencillos comandos, que resolviera dependencias automaticamente y que compilara los paquetes optimizados para mi sistema. Asi que decidí ponerme manos a la obra.

2. Descripción general:

PkgBuilder es un sistema para automatizar la creación de paquetes similar a los *ports* de los sistemas **BSD**. La intención inicial es que funcione en cualquier distribución y/o cualquier sistema de paquetes, pero por ahora solo soporta **Slackware**, que es la distribución que uso.

Su funcionamiento es muy simple: se descarga el código fuente desde la web oficial del desarrollador, después verifica el checksum de los archivos descargados, descomprime los archivos y les aplica los parches correspondientes, luego se realiza la compilación del paquete y la creación del tgz.

Todo esto está gobernado por una serie de sencillos scripts escritos en shell script. Por cada paquete existe un script para cada versión del paquete y en él se configuran las variables necesarias para la compilación y creación del paquete y se implementan una serie de métodos, uno por cada acción. Dentro de cada uno de esos métodos está la implementación de la acción para ese paquete.

3. Prerrequisitos:

3.1. Sistema operativo:

El sistema se basa sobre **Slackware 10.0**, instalando sólo una serie de paquetes básicos sobre los que construir el resto del sistema. Para la mayoría de paquetes que contiene **PkgBuilder** existe un paquete oficial compatible en **Slackware**, asi que se puede optar por instalar el paquete oficial en lugar del paquete de **PkgBuilder**, aunque no todos, de hecho, algunos son totalmente incompatibles, como puede ser el paquete *oggutils* y los paquetes *libao*, *libogg* y *libvorbis* de **PkgBuilder**.

El sistema deberá tener tamaño libre suficiente para almacenar los archivos de código fuente descargados, los paquetes binarios creados y también tamaño suficiente para la compilación de cada paquete. Los archivos descargados se pueden eliminar sin problemas y los paquetes binarios generados también para liberar espacio en disco.

3.2. Paquetes basicos:

Esta es la lista de paquetes de **Slackware 10.0** necesarios para el sistema base. Algunos de ellos son opcionales como *jfsutils* o *xfsprogs*, estos serán necesarios si el sistema se va instalar en una partición con el sistema de ficheros jfs o xfs, respectivamente.

- a)

aaa_base 10.0.0

aaa_elflibs 9.2.0

bash 3.0

bin 9.2.0

bzip2 1.0.2

coreutils 5.2.1

cxxlibs 5.0.6

dcron 2.3.3

devs 2.3.1

e2fsprogs 1.35

elvis 2.2_0

etc 5.1

findutils 4.1.7

gawk 3.1.4

glibc-solibs 2.3.3

glibc-zoneinfo 2.3.3

grep 2.5

gzip 1.3.3

hdparm 5.7

hotplug 2004_09_23

infozip 5.51

jfsutils 1.1.6 (opcional, necesario si utilizamos particiones jfs)

kdb 1.12

kernel-ide 2.4.27

kernel-modules 2.4.27

less 382

lilo 22.5.9

module-init-tools 3.0

openssl-solibs 0.9.7d

pci-utils 2.1.11

pcmcia-cs 3.2.8 (opcional, necesario en portatiles y ordenadores con interfaz pcmcia)

pkgtools 10.1.0

procps 3.2.3

reiserfsprogs 3.6.18 (opcional, necesario si utilizamos particiones reiserfs)

sed 4.0.9

shadow 4.0.3

sysklogd 1.4.1

syslinux 2.11

sysvinit 2.84

tar 1.14

udev 042 (opcional, recomendable si se va a utilizar un kernel de la serie 2.6)

usbutils 0.11

util-linux 2.12h

utempter 1.1.1

xfsprogs 2.6.13 (opcional, necesario si utilizamos particiones xfs)

- ap)

diffutils 2.8.1

groff 1.17.2

man 1.5m2

man-pages 1.64

texinfo 4.7

- d)

autoconf 2.59

automake 1.9.2

bin86 0.16.15

binutils 2.15.92.0.2

bison 1.35

byacc 1.9

flex 2.5.4a

gcc 3.3.4

gcc-g++ 3.3.4

gdb 6.2.1

kernel-headers 2.4.27

libtool 1.5.10

m4 1.4.2

make 3.80

perl 5.8.5

- l)

glibc 2.3.3

glibc-i18n 2.3.3 (opcional)

ncurses 5.4

zlib 1.2.2

- n)

openssl 0.9.7d

tcpip 0.17

wget 1.9.1

wireless-tools 26 (opcional, necesario si queremos utilizar dispositivos inalambricos)

También se pueden utilizar paquetes equivalentes de la rama current de **Slackware**.

4. Instalación:

4.1. Instalación del sistema base:

4.1.1. Instalación desde el cdrom de Slackware:

La instalación del sistema base es totalmente similar a una instalación usual de **Slackware**. Se seleccionarán para su instalación unicamente los paquetes indicados en la lista de paquetes básicos.

4.1.2. Instalación desde una instalación GNU/Linux previa:

Para este tipo de instalación es necesario tener instalado previamente un sistema GNU/Linux completo.

- **Creación y formateo de las particiones:**

Primero debemos crear las particiones donde vamos a instalar el sistema en el que vamos a utilizar **PkgBuilder**. Podemos utilizar cualquier programa para ello como el clasico **fdisk**. Luego debemos formatear las particiones utilizando el sistema de ficheros que más nos guste. Y por fin, las montamos bajo **/mnt/pkgbuilder**, por ejemplo.

```
# mkdir /mnt/pkgbuilder
# mkreiserfs /dev/sda7
# mount /dev/sda7 /mnt/pkgbuilder
```

- **Instalación de los paquetes:**

Después ya podemos instalar los paquetes que previamente hemos debido descargar y los instalamos con el comando:

```
# installpkg -root /mnt/pkgbuilder -menu *.tgz
```

Esto hará que los paquetes se instalen bajo el directorio **/mnt/pkgbuilder** sobre las particiones que hemos montado allí. La opción *-menu* hará que se muestre un dialogo preguntando si se quiere instalar ese paquete.

- **Hacer chroot:**

Ahora para entrar dentro del nuevo sistema debemos hacer chroot sobre el directorio en el que hemos instalado los paquetes con esta secuencia de comandos:

```
# chroot /mnt/pkgbuilder ldconfig
# mount -t proc proc /mnt/pkgbuilder/proc
# mount -t devpts devpts /mnt/pkgbuilder/dev/pts
# chroot /mnt/pkgbuilder
# . /etc/profile
```

Y ya estamos dentro.

- **Compilar el kernel:**

Si necesitamos un kernel hecho a nuestra medida ahora es el momento de compilarlo. Por supuesto, siempre podemos utilizar el que trae por defecto **Slackware**.

- **Configurar el sistema:**

Ahora es el momento de configurar el sistema. Para ello, una vez dentro del sistema chroot, ejecutamos **pkgtool** y seleccionamos la opción *Setup*. Nos aparecerá una lista de scripts de configuración de los que seleccionaremos los que nos interesen.

install-kernel (opcional) si queremos instalar un kernel diferente al que viene de serie.

make-bootdisk (opcional) si queremos crear un disco de arranque.

modem-device (opcional) si tenemos un modem y queremos configurarlo.

hotplug (opcional) si queremos que hotplug se ejecute al inicio.

liloconfig (obligatorio) desde aquí configuraremos lilo para poder arrancar luego desde el nuevo sistema.

netconfig (obligatorio) configuramos la red.

services (opcional) para activar o desactivar servicios al inicio.

setconsolefont (opcional) si queremos cambiar la fuente de la consola.

timeconfig (obligatorio) para configurar la zona horaria.

- **Crear el archivo fstab**:

Esto no es mas que un ejemplo:

```
/dev/sda2 swap swap defaults 0 0
/dev/sda6 / reiserfs defaults 1 1
/dev/sda1 /tmp reiserfs defaults 1 2
/dev/sda8 /home reiserfs defaults 1 2
/dev/hdc /mnt/cdrw iso9660 noauto,ro,user 0 0
/dev/hdd /mnt/cdrom iso9660 noauto,ro,user 0 0
/dev/fd0 /mnt/floppy auto noauto,user 0 0
devpts /dev/pts devpts gid=5,mode=620 0 0
proc /proc proc defaults 0 0
usbfs /proc/bus/usb usbfs defaults 0 0
none /sys sysfs defaults 0 0
```

Y ya lo tenemos instalado. Solo nos queda arrancar desde él e instalar **PkgBuilder**.

4.2. Instalación de PkgBuilder:

Para instalar **PkgBuilder** es necesario primero conseguir una copia del arbol de ports. Se puede descargar un **tarball** de la página oficial o se puede descargar directamente a traves de **Subversion**. Yo recomiendo instalarlo a traves de **Subversion** ya que esto permitirá ir actualizando **PkgBuilder** fácilmente.

4.2.1. Instalación desde tar ball:

Simplemente será necesario descomprimir el **tarball** en el lugar elejido.

4.2.2. Instalación desde Subversion:

La instalación a traves de Subversion requiere instalar una serie de paquetes extra. Estos paquetes son **neon** y **subversion**. Se pueden encontrar estos paquetes precompilados en el ftp anonimo de **PkgBuilder** o también se pueden crear utilizando **PkgBuilder** desde el tar ball disponible en la web del proyecto.

El comando que descargar la última version de PkgBuilder es muy sencillo:

```
$ svn checkout svn://svn.berlios.de/pkgbuilder/trunk/pkgbuilder
```

Y ya tendremos nuestra copia del arbol de ports.

4.2.3. Configuración del archivo build.rc

Después de obtener una copia de **PkgBuilder** hay que configurar el archivo *build.rc*. Existe un archivo con un ejemplo del archivo de configuración, se trata de *build.rc-sample*.

```
   # Copyright 2004 Antonio G. Muñoz, tomby (AT) users.berlios.de
   # Distributed under the terms of the GNU General Public License
v2
   PKGBUILDER_HOME="/var/pkgbuilder"
   VERSION="20041114"
   # Packages db directory
   PACKAGES_LOGDIR="/var/log/packages"
   # PkgBuilder mirror
   PKGBUILDER_MIRROR="ftp://ftp.berlios.de/pub/pkgbuilder"
   # Local network source code mirror
   MIRROR_URL="ftp://localhost/pub/pkgbuilder"
   # CDROM with source code files
   CDROM_DIR="/mnt/cdrom"
   # Fetch options
   FETCH_TRIES="5"
   FETCH_DIR="/var/cache/pkgbuilder/sources"
   FETCH_OPTIONS="--tries=$FETCH_TRIES --directory-prefix=$FETCH_DIR"
   FETCH_FTP_OPTIONS="--passive-ftp"
   # Binary packages directory
   BINARIES_DIR="/var/cache/pkgbuilder/binaries"
   # Temporal directory
   TMP="/var/tmp"
   # USE flags
   USE_AUDIO="alsa oss arts esd oggvorbis speex lame mad flac fame
mikmod"
   USE_VIDEO="X opengl divx mpeg win32codecs sdl aalib svga fbcon
directfb"
   USE_NET="samba ssl slp maildir"
   USE_PRINT="cups pnm2ppa"
   USE_XLIBS="motif gtk gtk2 qt kde gnome"
   USE_LIBS="ncurses readline sasl bidi jpeg png tiff gif slang
fam"
   USE_CPU="mmx sse 3dnow"
   USE_DEV="python tcltk perl java ruby scheme"
   USE_MISC="nls doc gpm javascript mysql xml sane"
   USE="$USE_AUDIO $USE_VIDEO $USE_NET $USE_PRINT
   $USE_XLIBS $USE_LIBS $USE_CPU $USE_DEV $USE_MISC"
   # I18N configuration
   I18N="es"
   # CPU and ARCHITECTURE flags configuration
   ARCH="i486"
   CPU="i686"
   CFLAGS="-O2 -march=$ARCH -mcpu=$CPU"
   CXXFLAGS="$CFLAGS"
   CPPFLAGS="$CFLAGS"
```

- **PKGBUILDER_HOME** configura el directorio donde se ha instalado **PkgBuilder**.

- **PACKAGES_LOGDIR** configura el directorio donde se encuentra la base de datos de paquetes instalados. Usualmente toma el valor de /var/log/packages.
- **PKGBUILDER_MIRROR** configura la url del ftp oficial de **Pkg-Builder**.
- **MIRROR_URL** permite configurar un servidor local para que actue como cache.
- **CDROM_DIR** permite configurar el directorio donde tomar los archivos con el codigo fuente desde una unidad de CDROM.
- **FETCH_DIR** configura el directorio donde se almacenaran los archivos descargados.
- **FETCH_OPTIONS** permite configurar los parametros a utilizar por wget a la hora de descargar archivos.
- **FETCH_FTP_OPTIONS** permite configurar los parametros a utilizar por wget cuando la url se trata de un servidor FTP.
- **BINARIES_DIR** permite indicar el directorio donde se van a generar los paquetes binarios.
- **USE** permite configurar una serie de flags para activar o desactivar funcionalidades a la hora de compilar los paquetes. Es similimar a la variable USE de Gentoo GNU/Linux. Se ha dividido en varios grupos para facilitar la lectura de las opciones.
- **I18N** permite configurar la opcion de internacionalización indicando los lenguajes deseadoa.
- **ARCH** permite configurar la arquitectura de la máquina actual.
- **CPU** permite configurar el procesador de la máquina actual.
- **CFLAGS** permite configurar los flags de compilación indicando las opciones de optimización.

4.2.4. Configuración del soporte de compilación paralela con distcc:

Por defecto el soporte de compilación paralela está deshabilitado y es necesario configurarlo explicitamente. Para ello existe una serie de variables que es necesario configurar. También es necesario instalar y configurar correctamente **distcc** y/o **ccache** para que funcione junto con **distcc**. El soporte de compilación paralela aún es experimental y es posible que la compilación de algunos paquetes falle.

También se puede utilizar la compilación paraléla en el caso tener varios procesadores, sin necesidad de utilizar **distcc**.

```
   # Parallel compilation. Use with care, some packages compilation
fails.
   COMPILATION="parallel"
   DISTCC_HOSTS="host1 host2 host3 hostn"
   CCACHE_PREFIX="distcc"
   MAKE_OPTIONS="-j4"
```

- **COMPILATION** especifica la modalidad de compilación.
- **DISTCC_HOST** especifica el nombre de las diferentes máquina a usar para la compilación paralela en el caso de utilizar **dictcc**.
- **CCACHE_PREFIX** es necesario configurarla si queremos utilizar **distcc** junto con **ccache**.
- **MAKE_OPTIONS** especifica las opciones para **make**. Usualmente tomara el valor **-jn**, siendo **n** el número de procesos simultaneos que queremos que se ejecuten.

5. Empieza la fiesta:

5.1. El script build.sh:

El script *build.sh* permite ejecutar diversas acciones sobre los archivos build, entre ellas, descargar el codigo fuente compilarlo y generar el paquete tgz correspondiente.

```
# ./build.sh xap/aterm/aterm-0.4.2.build fetch patch configure
```

Este ejemplo ejecutará las acciones fetch, patch y configure para el archivo build aterm-0.4.2.build del paquete aterm.

Esta es la lista completa de acciones que podemos ejecutar:

info: muestra información del paquete

fetch: descarga los archivos necesarios para la compilación y creación del paquete.

unpack: descomprime los archivos necesarios.

verify: verifica el checksum de los paquetes descargados.

patch: parchea los archivos necesarios para la compilación.

configure: ejecuta el script configure del paquete.

build: ejecuta make.

install: ejecuta make install.

postinstall: ejecuta todas la acciones necesarias después de la instalación del paquete.

buildpkg: genera el paquete tgz correspondiente listo para su instalación en el sistema. El paquete sólo se generará si se ejecuta como superusuario.

installpkg: instala el paquete previamente generado por la acción buildpkg. Solo funciona cuando se ejecuta como superusuario.

upgradepkg: actualiza el paquete previamente generado. Solo funciona cuando se ejecuta como superusuario.

cleanup: realiza la limpieza de los archivos utilizados para la compilación y creación del paquete.

auto: ejecuta automáticamente esta secuencia acciones: fetch, verify, unpack, patch, configure, build, install, postinstall, buildpkg.

5.2. El script install.sh:

El script *install.sh* instala un paquete resolviendo dependencias instalandolas si es necesario de forma recursiva.

```
$ ./install.sh xap/epiphany
```

Este comando instala el paquete epiphany resolviendo las dependencias de este paquete.

Opciones del script:

-d Ejecuta el script en modo *dummy* o tonto, resolviendo las dependencias pero sin instalar nada en el sistema.

-v Ejecuta el script incrementando los mensajes de salida utilizado para depurar la gestión de dependencias.

-p Ejecuta el script en modo *frompackage*, en lugar de instalar el paquete compilandolo, lo instala a partir de un paquete precompilado localizado en el directorio $TMP.

5.3. El script update.sh:

El script update.sh actualiza todo el sistema buscando nuevas versiones de los paquetes en el arbol de ports.

-d Ejecuta el script en modo *dummy* o tonto, muestra los nuevos paquetes disponibles pero sin realizar sin ninguna acción.

Previamente a utilizar el script *update.sh* es necesario ejecutar el script *build_ db.sh* que genera el archivo **PACKAGES** necesario para ejectuar el script *update.sh*.

6. Creación de scripts build:

6.1. Anatomia de un script build:

Un script **build** define una serie de variables y métodos que dirigen la generación de un paquete. Además un script **build** puede heredar de otro ciertas caracteristicas que se repiten en múltiples paquetes.

6.1.1. Variables:

Existen una serie de variables comunes a todos los scripts build que definen una serie de parámetros cruciales para la construcción de los paquetes. Aqui definimos la mayoría de ellos, es posible que se nos pase alguno, además cada paquete puede definir toda una serie extra de variables.

- **PKG_NAME**: Define el nombre del paquete y su valor esta implicito, no es necesario declararlo.
- **PKG_VERSION**: Define la version del paquete y su valor esta implicito, no es necesario declararlo.
- **PKG_BUILD**: Define el numero de la revision del paquete actual. Esto es necesario ya que aunque no se haya cambiado de version es necesario actualizar el paquete.
- **PKG_ARCH**: Define la arquitectura del paquete. Por defecto toma el valor de la variable **ARCH** definida en el archivo *build.rc*.
- **PKG_SRC**: Define el directorio donde residen los fuentes del paquete después de ser descomprimidos.
- **PKG_UNPACK_DIR**: Define un directorio previo a **PKG_SRC** donde se descomprimiran los archivos. No suele ser necesario casi nunca ya que con definir **PKG_SRC** suele ser suficiente.
- **PKG_DEST**: Define el directorio de destino para la instalación del paquete.
- **PKG_DOC**: Define el directorio donde se instalaran los archivos de documentación.
- **PKG_FILENAME**: Define la lista de nombres de archivos necesarios para la construcción del paquete.
- **PKG_URL**: Define la lista de URL's desde donde descargar los archivos necesarios para la construcción del paquete.
- **PKG_LICENSE**: Define la licencia del paquete.
- **PKG_USE**: Resume la lista de flags USE utilizados para la construcción del paquete. Estos flags determinan que dependencias utilizar y que opciones de construcción se van a emplear, etcetera.
- **PKG_DEPENDS**: Define la lista de dependencias necesarias para la construcción del paquete. El sistema utiliza esta variable para resolver las dependencias. El formato con el que se especifican las dependencias es.
- **PKG_HOMESITE**: Define la web principal del paquete donde encontrar información extra sobre el paquete y nuevas versiones.
- **PKG_DOC_FILES**: Lista de archivos que se copiaran dentro del directorio PKG_DOC.
- **PKG_PREFIX**: Define el directorio prefijo. Este suele ser /**usr**.
- **PKG_CONFIGURE_OPTIONS**: Define la opciones que se pasaran al script **./configure**.
- **PKG_BUILD_OPTIONS**: Define las opciones que se pasaran a make a la hora de construir el paquete.

- **PKG_BUILD_TARGET**: Define los targets o objetivos make que se utilizaran a hora de construir el paquete.
- **PKG_INSTALL_OPTIONS**: Define las opciones que se pasaran a make a la hora de instalar el paquete.
- **PKG_INSTALL_TARGET**: Define los targets o objetivos make que se utilizaran a hora de instalar el paquete.
- **PKG_NOSTRIP**: Indica si es necesario o no pasar **strip** a los archivos binarios generados.
- **PKG_CONFIG_FILES**: Defina la lista de archivos de configuración que deben tener un tratamiento especial.

6.1.2. Acciones:

Todos los scripts **build** deben implementar todas las acciones, o sino ellos, sí alguno de los scripts de los que heredan. El nombre de las funciones que implementan las acciones es **do_** <***nombre_accion***> en todos los casos. Esta es la lista de funciones:

- do_info
- do_fetch
- do_verify
- do_unpack
- do_patch
- do_configure
- do_build
- do_install
- do_postinstall
- do_buildpkg
- do_installpkg
- do_upgradepkg
- do_cleanup

6.2. Herencia de scripts:

Como se ha explicado anteriormente, cada script build debe implementar todas las acciones, pero si realmente se tuviera que implementar todas las funciones en cada script build, se llegaría a la situación de tener el mismo código repetido en todos los scripts. Esto haría al sistema inmantenible ya que si quisieramos modificar el comportamiento de una acción, o añadir una nueva, habría que modificar todos los scripts ya creados.

Para solucionar esto se ha implementado una herencia sencilla entre scripts build. Cada script build puede tener un padre del que hereda todas las acciones y variables definidas en él. Además el script padre puede heredar de otro y así sucesivamente, formando una jerarquía. Así hasta llegar al script base.build que implementa una serie de acciones globales y del que heredan todos los scripts build.

A la hora de ejecutar una acción primero se busca si se ha declarado la función **do_<accion>**. Si esta no está definido se busca si el metodo está declarado en el padre a través de la función **<padre>_do_<accion>**, así sucesivamente subiendo por la jerarquia. Si no está declarada tampoco en la jerarquía se buscará si esta acción está implementada en el script *base.build* con el nombre **pkg_do_<accion>**. Si no se encuentra ninguna de estas funciones se asume que la acción no ha sido implementada porque no es necesaria, sin mas.

Con esto conseguimos un sistema sencillo a base de shell scripts que nos permite mantener de forma sencilla el sistema.

6.3. Funciones genericas:

- **version**: imprime la versión de **PkgBuilder**.
- **include**: incluye el script build del directorio common que se le pasa como parámetro
- **inherit**: realiza las acciones necesarias para que el script build que se le pase como parámetro sea el padre del script actual.
- **use**: comprueba si está definido el flag use en la lista de la variable USE.
- **use_dep**: comprueba si está definido el flag use en la lista de la variable USE, si es así además imprime la cadena que se le pasa como segundo parámetro.
- **use_enable**: comprueba si está definido el flag use en la lista de la variable USE, además imprime la cadena *–enable-<flag>* o *–disable-<flag>*.
- **use_with**: comprueba si está definido el flag use en la lista de la variable USE, además imprime la cadena *–with-<flag>* o *–without-<flag>*.
- **call_action**: llama la acción correspondiente buscandola tal y como se describe en el punto 6.2.
- **execute_action**: ejecuta la acción que se le pasa como parámetro.
- **fetch**: descarga el archivo que se pasa como parámetro.

- **unpack**: descomprime el archivo que se pasa como parámetro.
- **apply_patch**: aplica el parche correspondiente buscando el nombre del archivos en varios directorios.
- **verify**: verifica el checksum de los archivos descargados.
- **gzip_man**: comprimir las páginas man.
- **gzip_info**: comprime las páginas info.
- **strip_all**: realiza el script sobre los binarios generados.
- **is_installed**: comprueba si el paquete está instalado.
- **compare_versions**: compara dos versiones determinado si una es mayor, menor o son iguales.
- **latest_version**: imprime el valor de la última versión disponible del paquete indicado.
- **installed_version**: imprime la versión instalada del paquete indicado.
- **installed_build**: imprime la reversión instalada del paquete indicado.
- **extract_meta**: extrae el valor del metapaquete.
- **extract_name**: extrae el valor del nombre del paquete
- **extract_version**: extra la valor de la versión del paquete
- **extract_only_version**: extrae el valor de la versión sin el valor de la version extra del paquete.
- **extract_extra_version**: extra el valor de la versión extra del paquete.
- **result_msg**: imprime el resultado.

6.4. Funciones de paquetes:

- **pkg_installdoc**: instala los archivos definidos en la variable **PKG_DOC_FILES** en el directorio **PKG_DOC**.
- **pkg_stripall**: realiza el strip sobre los binarios instalados en **PKG_DEST**.
- **pkg_gzipmaninfo**: comprime las páginas de manual instaladas en **PKG_DEST$PKG_PREFIX/** y las páginas info instaladas en **PKG_DEST$PKG_PREFIX/info**.
- **pkg_configfiles**: realiza la gestion de los archivos definidos en la variable **PKG_CONFIG_FILES**.
- **pkg_installfiles**: instala los archivos **slack-desc** y **doinst.sh** necesarios para la creación del paquete.
- **pkg_fetchfiles**: descarga la lista de archivos definidos en la variable **PKG_URL**.
- **pkg_fetchcvs**: descarga desde el cvs.

- **pkg_unpack**: descomprime la lista de archivos definidos en la variable **PKG_FILE_NAME**.
- **pkg_configure**: ejecuta el script **./configure** pasandole las variables necesarias definidas en el script **build**.
- **pkg_build**: ejecuta el **make** pasandole las variables necesarias definidas en el script **build**.
- **pkg_install**: ejecuta el **make install** pasandole las variables necesarias definidas en el script **build**.
- **pkg_virtual**: añade la creación de un paquete virtual.
- **pkg_localeclean**: limpia los locales instalados.
- **pkg_postinstall**: ejectuta todo lo necesario para después de la instalación.

6.5. Cómo escribir un script build: