Creating pkgadd Software Packages under Solaris

The software pointed to by my Sun Freeware Web page is archived in a format that can be read and installed by the **pkgadd** program that comes with Solaris 2.5, 2.6, 7 and 8. I gave instructions on how to install our pkgadd software on a another page, but I did not discuss how we created the packages. A number of people have contacted me and said "How did you do that?!" The answer is "... with some struggle and hair pulling."

Michael Short at Berkeley first educated me on the basic steps to create a pkgadd format software package. Here I will add to his steps and try to make them as clear as I can. No doubt, there are more sophisticated or general ways to do this, but I will not attempt anything harder here. Comments are welcome.

There now exists a detailed discussion of the packaging process in the Answerbook 2 on the Solaris CD or on the Application Packaging Developer's Guide . See also updates to these basic processes here or on the FAQ as I learn them.

The Steps

Important: You will need root access on your machine for some of this.

- **Select your software**

  Find the source code to the package you want to compile. Read all the installation and other instructions carefully. In particular, look at the Makefile and understand which executables are created and how they are linked to other executables and libraries. Also figure out exactly what parts of the compiled code, libraries, include files, data, etc. are required to run the program. Some programs require other programs in order to run. You will have to find that software and package it also. Find the documentation, README, Copyright, manual, and any other files that might be of use to your end users.

- **Read the manuals and man pages**

  As difficult and unpleasant as this is to do, read the man pages for at least the following commands:

  pkgadd, pkginfo, pkgmk, pkgparam, pkgproto, pkgtrans, pkgrm

  There are a lot of options and details that you might need to know beyond what I will describe here.

- **Set up the directory structure**

  Typically, I set up a source directory like /opt/SOURCES to contain the source. In many cases, the default installation directory is /usr/local. For illustration, we will install a ficticious program called `prog` in the directory /usr/local. Your directory names and programs may be different of course. Each package needs a name like SCprog. The SC gives some indication of the author or organization that created the program.

- **Compile the program and install into /usr/local or elsewhere**

  You now must get your program to compile and run. Go to the directory containing the source and read the instructions. Most programs have detailed installation instructions either in a README, INSTALL, or similar file. Once you have the program compiled you will need to run something like "make install". This will put the files in /usr/local or elsewhere depending on what the Makefile says to do.

  It is often the case that you have a /usr/local on your system and when you do a "make install" the new files will get mixed up with other files already there. I usually have a structure like:

  /opt/SOURCES/local.full

  which contains all my usual working programs and a few links like:

  ln -s /opt/SOURCES/local.full /opt/SOURCES/local

  ln -s /opt/SOURCES/local /usr/local

  Under normal compiling I have links like these, But when I want "make install" to put files in place where I need to package them, I do:

  cd /opt/SOURCES
  rm local
  mkdir local

  Then I do the "make install". This puts files in /usr/local/bin, etc. I also mkdir directories like /usr/local/doc/prog to keep all the source documentation. Once I have all the files in /usr/local (which is really /opt/SOURCES/local), I do the packaging discussed next.

- **Create the prototype and pkginfo files**

  Go into the /usr/local directory with

  ```
  unix# cd /usr/local
  ```

  and run the command

  ```
  unix# find . -print | pkgproto > prototype
  ```

  This will produce the `prototype` file in /usr/local.

  Now take your editor and edit out the line that has the prototype file name in it. Then add a line like

  ```
  i pkginfo=./pkginfo
  ```

  on the first line.

  Finally, convert all the user and group ownerships from whatever they are to bin and bin. An example file looks like

  ```
  i pkginfo=./pkginfo
  d none bin 0755 bin bin
  ```

```
f none bin/prog 0755 bin bin
d none doc 0755 bin bin
f none doc/doc1 0644 bin bin
d none lib 0755 bin bin
f none lib/lib1 0644 bin bin
d none man 0755 bin bin
d none man/man1 0644 bin bin
f none man/man1/prog.1 0444 bin bin
```

It is very important that you change the ownerships. The program might not work when installed if owned by another user. Not changing these properly is one of the most common problems we have encountered.

Now in /usr/local create a file `pkginfo` with contents for your package like

```
PKG="SCprog"
NAME="prog"
ARCH="sparc"
VERSION="1.00"
CATEGORY="application"
VENDOR="Christensen and Associates, Inc."
EMAIL="steve@smc.vnet.net"
PSTAMP="Steve Christensen"
BASEDIR="/usr/local"
CLASSES="none"
```

These values are fairly obvious, but they mean

PKG = the name you have chosen for the package directory
NAME = the program name
ARCH = the operating system version
VERSION = the version number for your program
CATEGORY = the program is an application
VENDOR = whoever wrote the software
EMAIL = an email contact
PSTAMP = the person who did the port perhaps
BASEDIR = the /usr/local directory where the files install
CLASSES = just put none here

- **Run pkgmk**

  Now while in /usr/local, run

  ```
  unix# pkgmk -r `pwd`
  ```

  This places a file in /var/spool/pkg called SCprog.

- **Run pkgtrans**

  Now do

  ```
  unix# cd /var/spool/pkg
  ```

  and then

  ```
  unix# pkgtrans -s `pwd` /tmp/prog-1.00
  ```

You will be asked to select which package you want to make. Select you package name (like SCprog) by number.

This now creates a file called `prog-1.00` in /tmp.

- **gzip prog-1.00 and obtain prog-1.00.gz.**

  Now run

  ```
  unix# gzip prog-1.00
  ```

  in /tmp to produce the gzipped version **`prog-1.00.gz`** that you can move to where ever you want to store packages. This completes the packaging process. It is relatively easy to write scripts to do this.

- **Test the packaging**

  I usually test the `prog-1.00.gz` file by doing a new install with it. I back up what I had in /usr/local somewhere and then delete the directory. I store my gzipped packages in a directory called /opt/SOURCES/PKG. In /opt/SOURCES/PKG, I run

  ```
  unix# gunzip prog-1.00.gz
  ```

  to get back prog-1.00. Then running as root user, I do

  ```
  unix# pkgadd -d prog-1.00
  ```

  and follow the instructions to create the SCprog. I can then put /usr/local into my UNIX path or make the executables and man pages available in whatever way I typically choose.

  Once I have tested the package, I then relink

  cd /opt/SOURCES
  mv local local.prog
  ln -s /opt/SOURCES/local.full /opt/SOURCES/local

  which moves the packages files to another directory and then relinks my working /usr/local.

- **Postinstall scripts**

  It can be that not all the files you need to install go into a single directory like /usr/local. This is a brief outline of how to put files other places.

  Suppose I have files that need to go into /etc rather than /usr/local. What I do is create a directory in /usr/local like /usr/local/etc. I put the files there that need to go eventually into /etc. Then I create a file called **postinstall** into /usr/local. In that file I put one line

  mv /usr/local/etc/* /etc

  which contains the command to do what I want. (You might want to put checks in the postinstall file to first test if the files you are moving do not overwrite files that are already there.) A series of commands (compatible with /bin/sh) can be put there. These commands are executed after the files are put in /usr/local by pkgadd. In order for postinstall to be used, put a

line like:

i postinstall=./postinstall

into prototype after the first i pkginfo line. No other lines containing postinstall can be in the prototype file.

Recently, Diethard Ohrt from Austria sent the following suggestions regarding the use of postinstall, which may be of some value to you:

```
Date: Mon, 12 Jul 1999 15:32:15 +0200
From: Diethard Ohrt <Diethard.Ohrt@siemens.at>
To: steve@smc.vnet.net
Subject: pkgmk -- prototype

I'd like to add a remark to your section "Creating Packages" --
especially the subsection about postinstall, regarding files to be
installed "elsewhere", i.e. NOT under /usr/local in your example.

If you do it your way, the problem will be:
"pkgadd" creates the files you are mentioning in /usr/local/etc/; then
"postinstall" moves them to /etc, but the installation process itself
does NOT control or notice that. So when you do a "pkgrm", you will
probably get some warning messages about files that have been removed
from /usr/local/etc. But, more important, you have to remove your files
from /etc YOURSELF, e.g. via a "preremove" or "postremove" file. This
additionally makes up another problem: You have to know yourself, which
files you have to remove from /etc ...

So, why not modify your "prototype" in a way that it installs the files
itself into the proper directories? This has the big advantage that the
files are removed by "pkgrm".

Suppose you have some files etc/file_1, etc/file_2, ... that have to be
installed in /etc; the appropriate "prototype" line would be:
f none /etc/file_1=etc/file_1 0644 bin bin
f none /etc/file_2=etc/file_2 0644 bin bin
...
All other "prototype" lines may be left unchanged.


HTH,
Diethard
      _/_/    _/    _/  _/_/_/    _/_/_/_/              _  __------ \
   _/    _/  _/    _/  _/    _/     _/                 | --          \
  _/    _/  _/_/_/_/  _/_/_/       _/     Diethard     |_              \
 _/    _/  _/    _/  _/   _/      _/        Ohrt        | ___-\   o   |
  _/_/    _/    _/  _/    _/     _/                     /_/     |      |
                                                              |__---|
Diethard.Ohrt@siemens.at       Steiermark - das gruene Herz Oesterreichs
SIEMENS AG / PSE TMN G3              Styria - the green heart of Austria
A-8054 Graz     Austria                Styrie - le coeur vert d'Autriche
=============================================================================

Another method for making packages some from Jasper Aukes.

Date: Tue, 15 Feb 2000 12:09:54 +0100 (MET)
From: Jasper Aukes  <J.Aukes@azu.nl>
Subject: make_package
To: steve@smc.vnet.net

Hi Steve,
```

I've create the following script that could help people creating packages. It creates the prototype file, builds the pkginfo file after asking some questions about the software and finally builds the package using pkgmk and pkgtrans. Afterwards it gzips the package, ready to be stored on your package-server.

[You can download this file rather than copy and pasting by clicking make_package.

```
#!/net/bin/perl
#
# Automated processes to create SUN packages
# You can run this script after you did a 'make install' in the chrooted
# environment. Run it from the <whatever>/packagename-1.0/usr/local/ directory
#
# JA: 06-01-2000        Initial release
# JA: 25-01-2000        Beautified a little

$find = "/usr/bin/find";
$pkgproto = "/usr/bin/pkgproto";
$pkgmk = "/usr/bin/pkgmk";
$pkgtrans = "/usr/bin/pkgtrans";
$temp = "/tmp/prototype$$";
$prototype = "prototype";
$pkginfo = "pkginfo";

# Sanitycheck

$pwd = `pwd`;
if ($pwd =~ '\/usr\/local') {
        $pwd = $`;
}
die  "Wrong location, please cd to <PKGBASE>/usr/local/ and run again.\n" if ($pw

system  ("$find . -print | $pkgproto > $temp");
open (PREPROTO,"< $temp") || die "Unable to read prototype information ($!)\n";
open (PROTO,"> $prototype") || die "Unable to write file prototype ($!)\n";
print PROTO "i pkginfo=./$pkginfo\n";
while (<PREPROTO>) {
        # Read the prototype information from /tmp/prototype$$
        chomp;
        $thisline = $_;
        if ($thisline =~ " prototype ") {
          # We don't need that line
        } elsif ($thisline =~ "^[fd] ") {
          # Change the ownership for files and directories
          ($dir, $none, $file, $mode, $user, $group) = split / /,$thisline;
          print PROTO "$dir $none $file $mode bin bin\n";
        } else {
          # Symlinks and other stuff should be printed as well ofcourse
          print PROTO "$thisline\n";
        }
}
close PROTO;
close PREPROTO;

# Clean up
unlink $temp || warn "Unable to remove tempfile ($!)\n";

# Now we can start building the package
#
```

```
# First get some info

$thispackage = `basename $pwd`;
if ($thispackage =~ '-') {
        $default{"name"} = $`;
        $default{"version"} = $';
        chomp $default{"version"};
} else {
        $default{"name"} = $thispackage;
        chomp $default{"name"};
        $default{"version"} = "1.0";
}
$default{"pkg"} = "UMC" . substr($default{"name"},0,4);
$default{"arch"} = `uname -m`;
chomp $default{"arch"};
$default{"category"} = "application";
$default{"vendor"} = "GNU";
$default{"email"} = "info@\gnu.org";
$login = getlogin();
($user, $passwd, $uid, $gid, $quota, $default{"pstamp"}, $userInfo, $userHome, $l
$default{"pstamp"} = "Jasper Aukes" if ($default{"pstamp"} eq "");
$os = `uname -r`;
$os =~ '\.';
$os = "sol$'";
chomp $os;
$default{"basedir"} = "/usr/local";

# Check for correctness of guessed values by userinput

%questions = (
  pkg => "Please give the name for this package",
  name => "Now enter the real name for this package",
  arch => "What architecture did you build the package on?",
  version => "Enter the version number of the package",
  category => "What category does this package belong to?",
  vendor => "Who is the vendor of this package?",
  email => "Enter the email adress for contact",
  pstamp => "Enter your own name",
  basedir => "What is the basedir this package will install into?",
  packagename => "How should i call the packagefile?",
);

@vars = qw(pkg name arch version category vendor email pstamp basedir packagename
foreach $varname (@vars) {
        $default{"$varname"} = "$name-$version-$os-$arch-local" if ($varname eq "p
        getvar($varname);
}
$classes = "none";

# Create the pkginfo file

print "\nNow creating $pkginfo file\n";
open (PKGINFO,"> $pkginfo") || die "Unable to open $pkginfo for writingi ($!)\n";
print PKGINFO "PKG=\"$pkg\"\n";
print PKGINFO "NAME=\"$name\"\n";
print PKGINFO "ARCH=\"$arch\"\n";
print PKGINFO "VERSION=\"$version\"\n";
print PKGINFO "CATEGORY=\"$category\"\n";
print PKGINFO "VENDOR=\"$vendor\"\n";
print PKGINFO "EMAIL=\"$email\"\n";
print PKGINFO "PSTAMP=\"$pstamp\"\n";
print PKGINFO "BASEDIR=\"$basedir\"\n";
print PKGINFO "CLASSES=\"$classes\"\n";
close PKGINFO;
```

```
print "Done.\n";

# Build and zip the package

print "Building package\n";
system ("$pkgmk -r `pwd`");
system ("(cd /var/spool/pkg;$pkgtrans -s `pwd` /tmp/$packagename)");
system ("gzip /tmp/$packagename");
print "Done. (/tmp/$packagename.gz)\n";

# The subroutines

sub getvar {
        my $questionname = "@_";
        print "$questions{$questionname} [$default{\"$questionname\"}]: ";
        my $answer = <STDIN$gt;;
        chomp $answer;
        $$questionname = $answer;
        $$questionname = $default{$questionname} if ($$questionname eq "");
}
```

I also created a chrooted environment creator, to enable people to just run 'make install' in a clean environment. I think i mailed you about that some time ago, but perhaps something went wrong.

The chrooted environment avoids problems with /usr/local/ in an environment where you just can't miss that directory (no way to quickly unmount it and remount it afterwards)

Please tell me if you want that story (and script) again.

When i have to create a package these days, i do the following:

```
cd /tmp
tar zxvf example-1.4.tar.gz
cd example-1.4
./configure
make                                    # Software is compiled here
mkdir /tmp/example-1.4/tmp
mv * /tmp/example-1.4/tmp
setup_chroot /tmp/example-1.4
cd ..
chroot example-1.4 /bin/sh              # Get a chrooted environment (as root)
make install                            # Install in the chrooted environment
exit                                    # Back to normal mode
cd /tmp/example-1.4/usr/local
make_package                            # Run the above script (as user)
```

You got to have writepermission in /var/spool/pkg as user for this.

Hope this is usefull. It helps me in building packages very much.

Jasper

--
Jasper Aukes | Unix system-administrator
| Academic Medical Centre Utrecht
Phone: +31 30 250 9283 | Bolognalaan 4, Utrecht
Fax: +31 30 254 2028 | POBox 85500, 3508 GA Utrecht, NL

Date: Wed, 16 Feb 2000 10:29:13 +0100 (MET)
From: Jasper Aukes <J.Aukes@azu.nl&ft;
Subject: Re: make_package

To: steve@smc.vnet.net

Hello Steve,

 Thanks for this.   I have added it to my pkgadd page.

I see. Please note this script only works if a 'make install' has been
run that installed the compiled package in a _clean_ usr/local
directory from a <whatever>/packagename-1.0/ directory.
You will need my 'setup_chroot' script as well. I'll include it here:

```
#!/bin/sh
## Original script by Dug Song <dugsong@UMICH.EDU>, used for a chrooted
## postfix environment, adapted by Jasper Aukes <J.Aukes@azu.nl> to be used for
## chrooted package creation.
#
# Location: /root/bin/setup_chroot
#
# Usage:
#
# First, create a chrooted dir with needed files (some may be left out (tcsh
# etc), some might be missing on your system, some might be located elsewhere.
# This script is just a dirty hack and by NO means intelligent. I should
# improve it and write it in Perl when i have time.
#
# /root/bin/setup_chroot /tmp/PAKNAME
#
# Then, tar zxvf your package into: /tmp/PAKNAME/tmp/
#
# cd /tmp/PAKNAME/tmp, configure and compile your package
# DO NOT 'make install' just yet...
#
# Now, cd to /tmp and run:
#
# chroot PAKNAME /bin/sh        # Or /bin/tcsh if you can and if you prefer it
#
# Check if you're really in the chrooted environment (f.e. cat /etc/passwd, it
# shouldn't be there) :-)
#
# Now cd to your /tmp/package-version##         # fill in the right name
# and run a make install
#
# Exit your chrooted shell and start building the SUN package from the
# directory /tmp/PAKNAME/usr/local/     # Or /usr, or even /
#
# A nice page to see how this stage could see a happy ending is to be found at
# http://sunfreeware.com/pkgadd.html by Steven M. Christensen

PATH=/usr/bin:/sbin:/usr/sbin

# Create chroot'd area under Solaris 2.5.1 for postfix.
#
# Dug Song <dugsong@UMICH.EDU>

if [ $# -ne 1 ]; then
  echo "Usage: `basename $0` <directory>, e.g.: /var/spool/postfix" ; exit 1
fi

CHROOT=$1

# If CHROOT does not exist but parent does, create CHROOT
if [ ! -d ${CHROOT} ]; then
  # lack of -p below is intentional
```

```
  mkdir ${CHROOT}
fi
if [ ! -d ${CHROOT} -o "${CHROOT}" = "/" -o "${CHROOT}" = "/usr" ]; then
  echo "$0: bad chroot directory ${CHROOT}"
  exit 2
fi
for dir in etc/default etc/inet dev usr/bin usr/lib usr/share/lib/zoneinfo \
    usr/local net \
    tmp ; do
  if [ ! -d ${CHROOT}/${dir} ]; then mkdir -p ${CHROOT}/${dir} ; fi
done
ln -s usr/bin ${CHROOT}/bin
ln -s usr/bin ${CHROOT}/net/bin

# Set the right permissions
chmod -R 755 ${CHROOT}

# Copy some terminfo files
for term in v x ; do
  if [ ! -d ${CHROOT}/usr/share/lib/terminfo/${term} ]; then \
    mkdir -p ${CHROOT}/usr/share/lib/terminfo/${term} ; fi
  cp /usr/share/lib/terminfo/${term}/* ${CHROOT}/usr/share/lib/terminfo/${term}
  chmod 644 ${CHROOT}/usr/share/lib/terminfo/${term}/*
done

# AFS support.
if [ "`echo $CHROOT | cut -c1-4`" = "/afs" ]; then
  echo '\tCreating memory resident /dev...'
  mount -F tmpfs -o size=10 swap ${CHROOT}/dev
fi

# Setup /etc files.
cp /etc/nsswitch.conf ${CHROOT}/etc
cp /etc/netconfig /etc/resolv.conf ${CHROOT}/etc
cp /etc/default/init ${CHROOT}/etc/default
cp /etc/inet/services ${CHROOT}/etc/inet/services
ln -s /etc/inet/services ${CHROOT}/etc/services
cp /usr/share/lib/termcap ${CHROOT}/usr/share/lib
ln -s ${CHROOT}/usr/share/lib/termcap ${CHROOT}/etc/termcap
find ${CHROOT}/etc -type f -exec chmod 444 {} \;

# Most of the following are needed for basic operation, except
# for libnsl.so, nss_nis.so, libsocket.so, and straddr.so which are
# needed to resolve NIS names.
cp /usr/lib/ld.so /usr/lib/ld.so.1 ${CHROOT}/usr/lib
for lib in libc libdl libintl libmp libnsl libsocket libw libkstat \
    libcurses libkvm libelf libgen nss_nis nss_nisplus nss_dns nss_files; do
  cp /usr/lib/${lib}.so.1 ${CHROOT}/usr/lib
  rm -f ${CHROOT}/usr/lib/${lib}.so
  ln -s ./${lib}.so.1 ${CHROOT}/usr/lib/${lib}.so
done

for lib in straddr libmp; do
  cp /usr/lib/${lib}.so.2 ${CHROOT}/usr/lib
  rm -f ${CHROOT}/usr/lib/${lib}.so
  ln -s ./${lib}.so.2 ${CHROOT}/usr/lib/${lib}.so
done

chmod 555 ${CHROOT}/usr/lib/*

# Copy timezone database.
(cd ${CHROOT}/usr/share/lib/zoneinfo
  (cd /usr/share/lib/zoneinfo; find . -print | cpio -o) | cpio -imdu
  find . -print | xargs chmod 555
```

```
)

# Make device nodes. We need ticotsord, ticlts and udp to resolve NIS names.
for device in zero tcp udp ticotsord ticlts; do
  line=`ls -lL /dev/${device} | sed -e 's/,//'`
  major=`echo $line | awk '{print $5}'`
  minor=`echo $line | awk '{print $6}'`
  rm -f ${CHROOT}/dev/${device}
  mknod ${CHROOT}/dev/${device} c ${major} ${minor}
done
chmod 666 ${CHROOT}/dev/*

# Now copy some usefull binaries
for bin in expr ls dirname cp chmod rm mv sed mkdir grep find cat \
  true basename ln chown false cmp chgrp ; do
  cp /usr/bin/${bin} ${CHROOT}/usr/bin
done
cp /usr/ccs/bin/strip ${CHROOT}/usr/bin
cp /bin/sh ${CHROOT}/usr/bin
for bin in install tar tcsh make ; do
  cp /usr/local/bin/${bin} ${CHROOT}/usr/bin
done
chmod 755 ${CHROOT}/usr/bin/*

exit 0
```

```
Greetings,

Jasper
--
Jasper Aukes           | Unix system-administrator
                       | Academic Medical Centre Utrecht
Phone: +31 30 250 9283 | Bolognalaan 4, Utrecht
Fax:   +31 30 254 2028 | POBox 85500, 3508 GA  Utrecht, NL
```

If you have any problems with these techniques, please study the man pages for the pkg programs above or contact your local UNIX guru. Please report any successes or failures in your understanding of these instructions.

Good luck and happy packaging.

Steve Christensen

Send comments or questions to steve@smc.vnet.net