NPC on Solaris HOWTO

Author: Solarpack Project Team
Revision: 1.16
Date: 2003-04-22

This document gives instructions on how to install the **NetBSD package collection tools** (NPCT) on a Solaris machine.

The NPC is further documented on its home site, at NetBSD.

Contents

1 Prerequisites

To install the NPCT on top of a Solaris system, you'll need to make sure you have the following Sun packages installed:

package name	description
SUNWcs[drul]	Core Solaris
SUNWscp[ru]	Source Compatibility
SUNWxcu4	XCU4 Utilities
SUNWesu	Extended System Utilities
SUNWgzip	The GNU Zip (gzip) compression utility

Copy/paste the following commands to quickly check this:

```
# pkginfo SUNWcsd SUNWcsr SUNWcsu SUNWcsl
# pkginfo SUNWscpr SUNWscpu
# pkginfo SUNWxcu4 SUNWesu SUNWgzip
```

If you also want to install GUI applications, you'll probably also need to install the OpenWindows environment packages – too long to be listed here, but they'll probably be installed in every default Solaris configuration.

2 Quick start

NPCT installs in a "sandbox", located under `/usr/pkg` (which means you'll require root access to the machine you're installing on).

Download Solarpack's latest NPC binary kit, available in a `.tar.Z` tarball form, and unpack the tarball at the root of your filesystem.

> **Note**
>
> If the binary kit for your specific software/hardware configuration is not present, it is worth trying to run a kit designed for a later Solaris release (and to tell us how it went). We hope to increase the number of binary kits available in the future. If no binary kit works for you, you will have to compile it from source; see the developer HOWTO.

You need to set a few environment variables for proper operation:

`PKG_TMPDIR`**:** set this to a temporary directory. This directory will be used as a staging area when binary packages are unpacked, before actually installing them:

```
# PKG_TMPDIR=/usr/pkg/tmp; export PKG_TMPDIR
```

Note that `PKG_TMPDIR` defaults to `/var/tmp`, which you'll possibly find convenient.

`PKG_PATH`**:** this is a colon-separated list of where `pkg_add` will look for binary packages.

If you're going to use the Solarpack's project binary repository, set it with (e.g.) the following Bourne shell script snippet:

```
# OS_VERSION=‘uname -r‘          # 5.8 for Solaris 8, etc.
# OS_PLATFORM=‘uname -p‘         # either ’sparc’ or ’i386’
# REMOTE_DIR="pub/sourceforge/solarpack/direct_download"
# REMOTE_DIR="${REMOTE_DIR}/SunOS-${OS_VERSION}/${OS_PLATFORM}/All"
# SERVER="<SERVER>"
# PKG_PATH="ftp://${SERVER}.dl.sf.net/${REMOTE_DIR}"
# export PKG_PATH
```

Sample `PKG_PATH` value:

```
ftp://umn.dl.sf.net/pub/sourceforge/solarpack\
        /direct_download/SunOS-5.8/sparc/All
```

Please note that the `URL‘‘s in ‘‘PKG_PATH` should **not** end with a / character.

Replace <SERVER> with the download server closest to you. The values (mirrors) that are currently possible for <SERVER> are:

- `umn` (Minneapolis, MN, USA)
- `unc` (Chapel Hill, NC, USA)

Of course, you can add all of these servers in case one of them is offline. Note that the other SourceForge mirrors are *not* usable, since they don't have an anonymous FTP service.

If you're going to build packages from source, make sure the first item of `PKG_PATH` is your local package repository; normally it's `/usr/solarpkgsrc/packages/All`. Other items can include `http` and `ftp` resources, but recursive dependency resolution will only be possible with `ftp` servers.

`PATH`: prepend `/usr/pkg/bin`, which will contain the executables of the packages you'll install. It should also contain `/usr/pkg/X/bin` (for X11-based packages). For the super-user `root` account, you may also want to add `/usr/pkg/sbin` to your path because it contains all the management tools like `pkg_*`:

```
# PATH=/usr/pkg/bin:/usr/pkg/sbin:${PATH}
```

`LD_LIBRARY_PATH`: prefix it by `/usr/pkg/lib` and `/usr/pkg/X/lib`:

```
# LD_LIBRARY_PATH=/usr/pkg/lib:/usr/pkg/X/lib:${LD_LIBRARY_PATH}
```

or:

```
# LD_LIBRARY_PATH=/usr/pkg/lib:/usr/pkg/X/lib
# export LD_LIBRARY_PATH
```

Now that the environment is set, you'll learn how to install binary packages in the following section.

3 Managing binary packages

Until the development of our package manager, zounds, enters beta status, the tools that can be used are the low-level tools provided with Zoularis. These tools have often a similar name as the native SVr4 packaging tools provided by Solaris; but making the difference is easy thanks to the underscore infixed in their name.

The basic three tools used to manage binary packages are pkg_info(1) pkg_add(1), and pkg_delete(1). pkg_add and pkg_delete can only be used by the root user – remember to set the environmental variables for this user as specified before.

For more information, and uncommon uses, take a look at the manpages.

Common uses:

`pkg_add <package>`: looks for a package in one of the locations specified by `PKG_PATH`; then downloads, unpacks and installs `<package>`, which can either be a package name (e.g. bison), a local file (`/space/packages/bison-1.35.tgz`) or an FTP URL. Dependencies are fetched, unpacked and installed first, of course.

Notes:

- you can also use HTTP URLs, but dependencies won't be resolved, as the contents of a directory can't be listed through HTTP.
- with plain package names as well as with FTP URLs, you don't have to specify the full package name (with version number and `.tgz` extension); the latest available version will then be used.

`pkg_delete [-R] <package>`: uninstalls a package. With the `-R` option, dependencies are also removed unless required by another package.

`pkg_info [<package>]`: lists summary information about all the installed packages If a package name is specified, shows details about a package.

`<package>` can either be a package name in short form (e.g. `teTeX`) or a path to a binary package file (e.g. `./lang/gcc-2.95.3nb2.tgz`), or even a glob pattern on package names.

`pkg_info -FI <file>`: shows which package `<file>` (absolute path) belongs to.

`pkg_info -n <package>`: shows which package(s) `<package>` depends upon.

`pkg_info -R <package>`: shows which package(s) depend upon `<package>`.

4 Notes

4.1 Standard C++ library

Many packages are written in C++, and use the standard C++ library. This usually means they need to find the `libstdc++.so` library at runtime, and this library is provided with the `gcc` package. Even if you're only installing binaries and are not going to compile anything, you should install NPCT's `gcc`: C++-written packages don't have a dependency against `gcc` (because it's installed in the base NetBSD distribution), so they won't work if you don't install `gcc` as well.

4.2 Manual pages

Sun's man browser toolchain lacks some capability to vew certain GNU or BSD manual pages; occasionally they'll appear as an ugly blob of text.

Here's a workaround. You'll need to have the `groff` package installed. Let's say you want to see the man page for foobar. First, type:

```
# man -d foobar
```

It should output gobs of text, and finally something like:

```
unformatted = /usr/pkg/man/man1/foobar.1
```

Then, the command:

```
# gnroff -mandoc /usr/pkg/man/man1/foobar.1 | less -is
```

does the trick, and you'll (hopefully) get a beautiful manual page...

4.3 Non-opensource packages

Some packages (`acroread`, `navigator`) are not available from binary repositories because of licencing restrictions. This packages can still be built and installed from source (instructions follow), and then be made available on your site's repository.

Solaris and Sun are trademarks of Sun Microsystems, inc