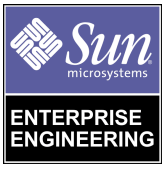
Rapid Recovery Techniques for Solaris™ Operating Environment

By Richard Elling - Enterprise Engineering

Sun BluePrints™ OnLine - January 2000



http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-4392-10
Revision 01, January 2000

Please
Recycle

Rapid Recovery Techniques for Solaris™ Operating Environment

When systems are down, recovery can be time consuming and expensive. Recovering a Solaris™ Operating Environment may require reinstalling the environment or restoring from a tape backup. However, the time to recover a system directly impacts its availability. The Solaris software registry provides an alternative means of recovery for some types of failures. The registry contains a record of all files installed on the system. The registry can be used to audit installed files and recover file attributes. Using software packages to install local changes into the registry also simplifies the configuration management of the Solaris Operating Environment. This article discusses the Solaris software registry, the ease of building packages, and the use of these packages in an automated installation environment.

Introduction

A key component of uptime measurements is how quickly a machine can recover after a failure. For example, a 99.9% annual uptime requirement means that any failure which occurs during the year must be recovered in less than 9 hours, or just a little more than one working shift. This article describes how to know what has changed on a system since the Solaris Operating Environment was installed. This knowledge can speed up recovery times since the system administrator will know exactly what is unique about the system.

All Solaris Operating Environments support local customizations. These can be as simple as nodename changes. Network naming services such as NIS provide an easy way to manage hostnames and user information. The Solaris Operating Environment software registry should be used to manage the local software customizations. Creating packages for installing local customizations is easy to do and is a good practice for operational environments. This article describes how to create such packages.

The Solaris Operating Environment packages can also be verified as installed correctly. This verification capability allows systems administrators to determine which files have changed on the system since installation time.

The Solaris Operating Environment Software Registry

The Solaris Operating Environment provides a software registry for maintaining information about the software packages and files that are installed on the system. This registry plays a critical role in software installation, upgrades, and patches. The registry is the primary location for software installation information. The registry can be used to audit a system and determine what files have been added, removed, or changed since the Solaris Operating Environment was originally installed.

The Solaris Operating Environment itself and the vast majority of Sun software is installed via the `pkgadd` program which properly updates the registry. Packages have been a part of the Solaris Operating Environment since version 2.1 and are part of the SVR4 environment.

The Solaris software registry is located in the `/var/sadm` directory under the `/var` file system. `/var` may vary widely in its disk space consumption because it contains the `/var/tmp` directory where users can place temporary files, as well as `/var/adm` and `/var/log` which collect accounting and logging information. The software registry requires space to grow as software packages are installed. Any patches that are installed without explicitly using the “do not backup the files” option stores the backup files in the `/var/sadm/pkg` directory. Enough space should be available in under `/var` to hold user temporary files and to handle software installation and patch management requirements. For a large server, the software registry may require 50 MB to 100 MB or more, and will grow over time as patches are added.

Patches are often implemented as sparse packages. The `patchadd` process correctly updates the software registry. The `patchadd` process also records the patch README in the registry for future reference.

Logs of all installed packages, files, and patches are kept in the registry. However, a complete set of tools does not exist for searching or modifying the registry. Fortunately, the registry is implemented as normal text or compressed files in a directory structure under the `/var/sadm` directory. The Solaris Operating Environment has a few commands that manage the registry while providing proper consistency controls.

Registry and Package Commands

The Solaris software registry and package commands are:

- `pkgadd(1m)`—transfer software packages to the system
- `pkgrm(1m)`—remove a package from the system
- `installf(1m)`—add a file to the software installation database
- `removef(1m)`—remove a file from software database
- `patchadd(1m)`—apply a patch package to a Solaris 2 or Solaris 7 Operating Environment
- `patchrm(1m)`—remove a Solaris 2 or Solaris 7 Operating Environment patch package and restore previously saved files
- `pkginfo(1)`—display software package information
- `pkginfo(4)`—package characteristics file
- `pkgchk(1m)`—check package installation accuracy
- `prodreg(1)`—GUI viewer for Solaris Product Registry
- `admintool(1m)`—system administration with a graphical user interface

`admintool` has a software installation mode which provides an interface into the registry. `swmtool` is available for backwards compatibility with older Solaris 2 Operating Environment releases and is now a link to `admintool` which starts up in the “add software” mode.

`prodreg` command is new in Solaris 7 5/99. `prodreg` is a Java technology based GUI and provides a more informative interface to the registry than `admintool`.

`pkginfo` is a command line program which provides several options for querying the registry.

`pkgchk` enables you to check specific files. The actual installed file can be checked against the registry. Files can also be checked against the `pkgmap` file in the original package.

Registry Directory Structure

The registry is located in the `/var/sadm` directory. The directory structure was reorganized slightly in Solaris 7 Operating Environment. A README file in `/var/sadm` explains the new directory structure.

Log files from system installation are located under the `/var/sadm/system` directory.

Each installed package has a directory under `/var/sadm/pkg`. The package information and any patch back out data is located under the appropriate package directory.

Patches are recorded under the /var/sadm/patch directory. Each patch has a directory name with the patch number and revision. A copy of the patch README is kept under this directory. This enables an administrator to review the system's patch collection along with the detailed README files. The files saved for patch back out are located under the appropriate /var/sadm/pkg directory.

The database of installed files is the /var/sadm/install/contents file. This is a text file with one line for every file installed on the system. The author's power workstation has more than 43,000 files listed in the contents file.

The pkgchk command can provide a verbose description of the information in the contents file. For example:

```
$ pkgchk -l -p /usr/bin/ls
Pathname: /usr/bin/ls
Type: regular file
Expected mode: 0555
Expected owner: bin
Expected group: bin
Expected file size (bytes): 18120
Expected sum(1) of contents: 53113
Expected last modification: Oct 06 00:43:05 1998
Referenced by the following packages:
    SUNWcsu
Current status: installed
```

Here is another example:

```
$ pkgchk -l -p /etc/hosts
Pathname: /etc/hosts
Type: symbolic link
Source of link: ./inet/hosts
Referenced by the following packages:
    SUNWcsr
Current status: installed
```

Registry Consistency

The registry is important for installing, removing, and patching the Solaris Operating Environment software. A locking mechanism exists to maintain consistency when changing the registry. The pkg commands properly implement the locking mechanism. lockf is not used, so other UNIX® commands which modify files can corrupt the registry. There is no mechanism for restoring the registry, other than restoring from a backup media such as tape. You should never use UNIX text

editors to modify the
`/var/sadm/install/contents` file directly. You can use read-only programs, such as `grep`, with the `contents` file. These programs may prove to be more useful for scripting than the `pkgchk` command.

The registry is not updated when software is installed with `cp`, `tar`, `cpio`, and other file copying programs. The `installf` command can be used to update the registry. However, it is usually easier to create a package and install it using `pkgadd` than it is to modify an installation script to use `installf`.

Packages for Local Customization

It is a good practice to use packages for local customization to systems. This has the advantage of promoting easy-to-implement change control, auditing, system cloning, and rapid recovery. `pkgadd` is preferable to hand editing files or using copy programs (such as `cp`, `tar`, and `cpio`) to copy local customizations.

Steps for Making a Package

The basic steps for creating a package are:

1. Create directory structure and `Makefile`.
2. Create the `pkginfo` file.
3. Organize package contents under the `reloc` directory.
4. Create information files.
5. Create the `pkgproto` template.
6. Run `make` to build the package.
7. Test the package before deployment.

This section gives an example of how to make a package. All of the components you need can be placed under a single directory. For this example, the following directory structure is used:

```
/home/richard/RMEex/
   Makefile
   pkgspool/
   reloc/
   src/
      copyright
      depend
      pkginfo
      pkproto.tmplt
```

The `Makefile` collects the copyright notice, dependency list, and pkginfo files for the new package directly. CODE EXAMPLE 1 shows an example `Makefile` which can be used as is for a variety of packages.

The `all` target is the default target and includes both the `pkgproto` and spooled package.

The `pkgproto` file is made by using the `pkgproto` command. `find` is used to traverse the `reloc` directory and put all of the files located there into the `pkgproto` file.

Once the `pkgproto` file is created, the `pkgmk` command is used to build the installable package. The package is stored in the `pkgspool` directory. The `-o` option instructs `pkgmk` to overwrite any existing package of the same name. Previous versions of the package should be archived for future reference. The `pkgspool` directory contains the completed package. The package is placed in a subdirectory with the name of the package abbreviation as defined in the `pkginfo` file. This package is now ready for installation via `pkgadd`.

CODE EXAMPLE 1 Makefile for a Package

```
# Makefile to make a package
#
# This is a rather simple Makefile designed to be used for
# making simple packages. The intended audience is Systems
# Administrators making packages for installing local
# customizations. A developer making a deliverable software
# package would likely require more sophistication.
#@(#)Makefile 1.3  09/20/99  Richard Elling  Sun Microsystems

# APPLBINDIR contains the binaries. All files and directories
# must have the same ownership and permissions as they are
# expected to have when they are actually installed.
APPLBINDIR=${PWD}/reloc

# APPLSRCDIR contains the source files for the package support.
# These are usually copyright, depend, and any install or remove
# scripts.
APPLSRCDIR=${PWD}/src

# PKGINFODIR contains the pkginfo and pkgproto.tmplt files.
# pkgproto.tmplt is the header for the automatically generated
# pkgproto file needed by the package.
PKGINFODIR=${APPLSRCDIR}
PROTOTMPLT=${PKGINFODIR}/pkgproto.tmplt
PROTOTYPE=${PKGINFODIR}/pkgproto

# SPOOLDEV is the location where the completed package will be
# spooled.
SPOOLDEV=${PWD}/pkgspool

all: pkgproto spoolpkg
pkgproto:
    -@echo "Making prototype file..."
    ( cd ${APPLBINDIR} ;\
    cp ${PROTOTMPLT} ${PROTOTYPE} ;\
    find . -print | pkgproto >> ${PROTOTYPE} )

spoolpkg:
    -@echo "Making package in spool directory..."
    ( cd ${PKGINFODIR} ;\
    pkgmk -o -b ${APPLBINDIR} -d ${SPOOLDEV} -f ${PROTOTYPE} )
```

The `reloc` directory contains the files to be installed. `pkgadd` installs the files exactly as they are in the `reloc` directory. Ownership and permissions (mode) is preserved. Files and directories are handled correctly when they must be owned by root or when they must have special permissions. Hard and symbolic links are also handled correctly.

`setuid` files may represent a security risk while in the `reloc` directory. The package itself does not contain `setuid` files as the ownership and mode are derived from the control files. Thus, the package parent directory should be in a secured environment with the proper mode set such that inappropriate users cannot access the directory.

The `src` directory contains files to be included with the package. In this example, does there are no preinstall or postinstall shell scripts. Such scripts would be placed in the `src` directory as needed.

The `copyright` file is copied intact and stored with the package information in the `/var/sadm/pkg/` directory.

The `depend` file is used at package installation time. It enables the package author to specify other packages that must be installed before installing this package. In the example shown in CODE EXAMPLE 2, the core Solaris Operating Environment must be installed. One would expect core Solaris to be required for all Solaris Operating Environment installations. The Solaris Operating Environment core dependencies are included here for example purposes.

CODE EXAMPLE 2 `depend` File

```
P SUNWcsr  Core Sparc, (Root)
P SUNWcsu  Core Sparc, (Usr)
```

The `pkginfo` file shown in CODE EXAMPLE 3 contains administrative information for the package. There are a number of keywords which can be set. The following parameters are required: `PKG`, `NAME`, `ARCH`, `VERSION`, and `CATEGORY`. The other fields described in the example are very useful and should be included.

`PKG` is the package abbreviation. This abbreviation must consist of nine or fewer characters, it must begin with a letter, and it must not be one of the reserved abbreviations: `install`, `new`, or `all`. By convention, the first four characters are unique to the company (e.g., `SUNW`).

`BASEDIR` is the base directory for installing the relocatable files. It is a good practice to make all packages relocatable so that installation on client systems is straightforward. The `BASEDIR` is the default location for the installed package.

It is a good practice to have separate packages for files to be installed under the / directory and files to be installed under /usr. This allows easy installation onto dataless client environments. Such environments share /usr but have unique / file systems. It is important to include a description and contact information so that other people can track the package back to its source.

CODE EXAMPLE 3 pkginfo File

```
PKG=RMEex
NAME=Richard Elling's package making example
ARCH=sparc
VERSION=1.0
CATEGORY=system
DESC=Richard Elling's package making example
BASEDIR=/
VENDOR=Sun Microsystems Incorporated
HOTLINE=(858)626-3920
EMAIL=Richard.Elling@West.Sun.Com
```

The pkgproto.tmplt, shown in CODE EXAMPLE 4, is used as the header for the pkgproto file. pkgmk includes the pkginfo, copyright, and depend files as information files with the package. The pkgproto command collects the filenames under the reloc directory and builds the remainder of the pkgproto file.

CODE EXAMPLE 4 pkgproto.tmplt File

```
i pkginfo
i copyright
i depend
```

Discipline Required for Package Use

It is too easy and tempting to simply hand edit files on a system. A disciplined approach is needed to ensure that the edits are made to the package and that the package is properly installed.

You can install packages much more quickly than you can edit files by hand. Proposed changes should be tested within a test environment, which should be used to determine the relocatable files for the package. The package can then be built and tested on a non-production machine. During scheduled downtimes, pkgadd can be used to install the package in the production environment. This technique helps to minimize downtime.

The pkgmk command automatically adds a timestamp to the package. This ensures that no two packages are identical. This also helps when auditing installed packages.

Installing the same package again on a system only updates files which have changed. This behavior is very similar for patches. For simple packages it is often easier to update the package and re-install it than it is to develop a patch. However, patches provide a back out mechanism which may prove useful for complicated packages.

AutoInstall

AutoInstall is Sun's mechanism for installing Solaris Operating Environment over the network. System administrators can customize the list of packages to be installed. It is easy to add local packages to the list of packages to be installed.

An AutoInstall environment has the added benefit of providing an alternate boot source. Booting across the network is often much faster than booting via CD-ROM. The ability to boot across the net to single user mode can be a good way to quickly recover a system with a damaged root filesystem.

However, AutoInstall environments also open up opportunities for security holes. Server consoles must be protected from unauthorized access to prevent unauthorized booting. Similarly, the boot network must be protected to prevent unauthorized boot servers from usurping the appropriate boot server. All of these security features are part of a proper datacenter design.

AutoInstall with local packages and a name service can be used to create a datacenter where every server can be installed in minutes. Thus, servers can be treated as field replaceable units (FRUs). In the worst server hardware failures, a completely new server can be installed with local customizations in minutes. Restoring from tape backup may take longer. Trying to track down the CD-ROMs, system administrators notes, or consultants who helped perform the initial installations could take weeks.

Verifying Package Installation

By default, `pkgchk -p pathname` checks `pathname` against the registry file (`/var/sadm/install/contents`) for ownership, mode, and checksum changes. If there is a discrepancy, it is printed to `stderr`.

Files found without entries in the registry are noted with a warning.

```
$ find / -mount -exec pkgchk -p {} \;
...
ERROR: /etc/default/init
file size <219> expected <462> actual
file cksum <18231> expected <38691> actual
WARNING: no information associated with
pathname </etc/default/dhcp>
...
```

In the above example, note that the `/etc/default/init` file contains the default timezone variable, `TZ`, setting. It is expected that this would change after installation for most sites. In this example, note that the `/etc/default/dhcp` file is shown as not in the registry. This file is generated by the `dchpconfig(1m)` command.

It is often interesting to add a `-print` option to the `find` command to see the list of pathnames checked against the registry. This allows you to make sure you are in fact checking the files you wish to be checking.

The `-mount` option to the `find` command restricts the find to the filesystem of the starting path. This allows you to limit the depth at which the search is performed. This is especially useful for excluding filesystems such as `/export/home` which is expected to have many files that are not in the registry.

The `pkgchk` program itself does not consume significant CPU resources. It follows the locking protocol for the registry which necessarily results in overhead. While a single `pkgchk` instance may only consume one to two seconds of wall clock time, the `find` shown in the example above may execute `pkgchk` thousands of times.

Security Audits

The `pkgchk` program can detect changes in file ownership, permissions, or checksums. Thus, it can be used to audit a system to determine if unauthorized `setuid` or trojan horse files have been installed.

A clever attacker can also change the registry contents file to reflect the proper status of the attacker's installed programs. `pkgchk` with the `-m` option can be used to check installed files against the package from which they were installed. This could be the Solaris Operating Environment CD-ROM or another source of secured packages.

A side effect of `pkgchk` is that directories are checked just like files. Since directory sizes and checksums change as files are added to the directory it is quite likely that `pkgchk` will flag many directories as changed.

Recovering File Ownership and Mode

The author has had several experiences where he, a subordinate, or a customer accidentally performed a recursive `chmod` on a system. For example, one case involved the command `chmod -R 777 /usr`. This had the effect of not allowing anyone to login to the system because the `/usr/bin/login` program must be `setuid` root. This also created a significant security problem which would have been potentially damaging if, in fact, anyone could login to take advantage of it. Naturally, a reboot didn't help. The solution was to use the Solaris software registry to fix the file modes. The `-f` option to `pkgchk` causes the pathname to be set to the ownership and mode as described in the registry contents file. Using `pkgchk` to repair the file modes provides a much faster recovery than reinstalling the Solaris Operating Environment.

`chmod` with numerical modes is more dangerous than the alpha modifiers. A numerical mode is set exactly as specified. In most cases, the intent is to add or remove a mode. It is a safer practice to use the alpha modifiers for such changes. See the `man` page on `chmod(1)` for more information.

Conclusion

The Solaris software registry is used for managing the software installed on the system. The registry can be used to verify installed software and determine any changes made to the system. This knowledge can speed recovery since the system administrator knows exactly what is unique about the system.

All Solaris Operating Environment systems have local customizations. These can be as simple as nodename changes. A network naming service, such as NIS, provides an easy way to manage hostnames and user information. The Solaris software registry should be used to manage the local software customizations. Creating packages for installing local customizations is easy, and it is a good practice for operational environments.

References

Additional information on Solaris Operating Environment software packages can be found in *Solaris 7 Application Packaging Developer's Guide* in the AnswerBook. This guide is also available on the Internet at `http://docs.sun.com`

A series of articles describing rapid recovery techniques for Solaris Operating Environment systems is available under the Sun BluePrint OnLine series on the internet at `http://www.sun.com/blueprints/browsesubject.html`.

Author's Bio: Richard Elling

Richard is a Senior Engineer in Enterprise Engineering for the Computer Systems at Sun Microsystems in San Diego, California. Richard had been a field systems engineer at Sun for five years. Richard was the Sun Worldwide Field Systems Engineer of the year in 1996. Prior to Sun, Richard was the Manager of Network Support for the College of Engineering at Auburn University, a design engineer for a startup microelectronics company, and worked for NASA doing electronic design and experiments integration for Space Shuttle missions.