

## NAME

**makeinstall** – capture file system changes with snapshot and rsync

## SYNOPSIS

```
makeinstall fs:=[+-]<file systems to make snapshot of> \  
    snaproot:=<snapshot root> \  
    bs:=<backing store> \  
    step=<n>
```

## INTRODUCTION

Generally speaking, a software application is just a set of files on the file systems. Some start one or more processes, like sendmail, apache, mysql, waiting for connection; some just sit there waiting to be called, like Perl, PHP, Python, and shells.

The major part of the Unix System Administration is to manage these sets of files – add, delete, and modify – either directly, like “vi httpd.conf”, or most of the time indirectly by running front end programs, such as “make install”, pkgadd, pkgm, rpm, useradd.

The first step in understanding and managing an application is to find its fileset. Traditional Unix makes this difficult as files belong to different packages are intermixed. Once you installed a software with “make install”, it is difficult to know where the files went, and how to uninstall it cleanly. The package manager excels in this regard, however, it won’t help you if there is no precreated package, and you need to create one yourself.

This article presents a simple idea and implementation using Solaris UFS snapshots and rsync to capture the file system changes, and to optionally reverse the changes. This technique can be used to make software package, to run a command in “test mode”, and to find out what is happening behind the scene.

The script “makeinstall” was originally written as a wrapper for “make install” identifying installed files for packaging (hence the name). But for maximum flexibility and capability, the script is divided it into 5 callable steps and can be used to catch the changes to the file systems made by “make install” or any other command.

## HOW MAKEINSTALL WORKS

Considering the simple case of replacing an existing file, a prudent administrator would use the following steps:

```
cp    foo foo.old # save the old copy  
cp    foo.new foo # install the new copy  
diff  foo.new foo # find what has changed  
cp    foo.old foo # optionally to reverse the change  
rm    foo.old    # remove the old copy
```

The 5 steps in “makeinstall” script are similar to above steps, but are on the file system level. The 5 steps are:

```
Step 1: save the old copies of file systems with snapshots  
Step 2: make the changes  
Step 3: compare the old and new file systems with rsync  
Step 4: optionally reverse the changes  
Step 5: delete the snapshots
```

The alternative ways to achieve the same include gaining intimate knowledge of the software package; using time stamps to find the newly installed or changed files; working on chroot environment.

Compared with the alternative ways, “makeinstall” uses brute force, and therefore is simpler and more reliable – as reliable as snapshot and rsync. The alternative ways can still be used to validate the results.

## BASIC SCRIPT USAGE

The makeinstall script is called with 4 named arguments processed with my\_getopts (Wang 2003):

```

fs:=[+-]<list of filesystems>           Default: /:/usr:/var:/opt
snaproot:=<snapshot top mount point>   Default: /snapshot
bs:=<backing-store>                   Default: $(< <snaproot>/backing-store)
step=<step number>                     Default: 0 (do nothing)

```

The `fs` option specifies what file systems to make snapshots of. The default is the subset of “/”, “/usr”, “/var”, “/opt” that exists. For example, if “/opt” is not a separate file system, it will not be in the list. You can use `fs:=+<file system list>` to specify additional file systems; and use `fs:=-<file system list>` to remove the file systems from the default list; and `fs:=<file system list>` to specify alternative file systems. The `<file system list>` is a list of file systems separated by colons.

The `snaproot` option specifies the snapshot root directory where the snapshot file systems are mounted under. The default is `/snapshot`. The snapshot for a file system will be mounted as their original name under this `snaproot`, except “/” is replaced by “/root” for clarity.

The `bs` option specifies the common backing store for all the snapshot file systems. Backing store is used to backup the old data when there is a change in the original file system. The backing store should not be in any of the file systems to make snapshots of.

The `step` option specifies which step to run. For example, “1” is to run step 1, and “2” is to run step 2, and so on. “34” is to run step 3 and 4. “9” is to run all the steps. “0”, which is the default, is to run no steps.

Example:

```
makeinstall fs:=+/apps4 snaproot:=/snapshot bs:=/apps2 step=1
```

## CODE REVIEW

The step 1 is to make a backup of all file systems possibly involved. This is like “`cp foo foo.old`”, but the temporary backup is done using `snapshot`.

This part is done using a `do_snapshot` function, the `fssnap` command hidden inside this function is Solaris specific:

```

function do_snapshot {
    typeset fs=${1} sr=${2} bs=${3} mp dv ix
    for ix in ${fs//:/ }; do
        mp=${sr}${ix/%"/"/"/root"}
        mkdir -p $mp
        mount -F ufs -o ro $(fssnap -F ufs -o bs=${bs},unlink $ix) $mp
        fssnap -i -o blockdevname $ix | awk "{print \$NF}" | read dv &&
        df -Pk $mp | grep "^$dv" || return 1
    done
}

```

This function takes three arguments: a list of file systems separated by colons to make snapshots of, the snapshot root directory for the snapshot mount points, and the backing store directory.

For each of the file system in the list, the snapshot is created, and mounted under the snapshot root directory under their original names except “/” file system is mounted as “root”.

For example,

```
do_snapshot /:/usr /snapshot /backing-store
```

will create snapshot for “/” and “/usr”, and mounted them as “/snapshot/root”, and “/snapshot/usr”.

The line

```
mp=${sr}${ix/%"/"/"/root"}
```

is a shorter but cryptic way to say:

```

if [[ $ix == "/" ]]
then mp=${sr}/root
else mp=${sr}${ix}
fi

```

The last two lines in the for loop is to check if the snapshot is successfully created and file system mounted. If the check fails, the function will immediately return with non-zero status. And this in turn will cause the program to terminate. This is because “errexit” (set -e) is set in the makeinstall script. For the nature of this program, we do not want to proceed whenever there is an error. We do not want to proceed with step 2 if step 1 fails.

Although this is a standalone function, the function itself is not exposed to the user. The user passes arguments to the makeinstall script, the script processes the arguments and calls the function.

Step 2 is a place holder. It simply reminds you to type “make install” or whatever command, for which you want to see the changes it makes on the file systems.

Step 3 is to find what changes have been made. This is done using rsync in “dry-run” mode (-n or --dry-run option), without doing the actual synchronization:

```
rsync -vanx --delete <source>/ <target>/
```

As of this writing, there are two issues in the current version of rsync (2.6.3) in dry-run mode:

- It does not produce a list of files with changes in owner / group, permission, or time stamp (“Rsync Bug 1764”).
- It does not report empty directories to be transferred to target (“Rsync Bug 1433”).

However this does not affect the majority of the application of makeinstall script, as changes in most cases do not fall into these categories.

Step 4 provides the option to reverse the changes. Rsync is run first in dry-run mode from the snapshot file system to the live file system. Then it ask your confirmation for the actual run. This is like “cp foo.old foo”.

We can think of two situations for which this step is needed. One is that the changes you just made are not desired. For example, it overwrites your existing files. The other situation is for packaging. You just want to find the list of files installed by “make install”, but do not actually install it.

Step 5 deletes the snapshots. This is a critical step, as the previously saved state of file systems will be gone. Unless you explicitly invoke this step, confirmation is prompted. The “undo\_snapshot” function used in this step is the reverse of the “do\_snapshot” function:

```

function undo_snapshot {
typeset fs=${1} sr=${2} mp ix
for ix in ${fs//:/ }; do
mp=${sr}${ix/%"/"/"/root"}
umount $mp
fssnap -F ufs -d $ix
! fssnap -i -o blockdevname $ix | awk "{print \$NF}" | read &&
! df -Pk $mp | grep "^/dev/fssnap/[0-9]\{1,\}" || return 1
done
}

```

The last two lines in the for loop to check if the snapshot is actually deleted and snapshot file system unmounted. Function fails when either of the conditions is not satisfied.

## CASE STUDIES

## CASE 1. Install Perl module

Frank is a system administrator for a major bank in US. One day he got a request from a developer to install the Perl module “Date::Manip”. Frank was not familiar with this Perl module or Perl module in general, but he did research and found the command to install the module was:

```
perl -MCPAN -e 'install Date::Manip'
```

Frank knows makeinstall script. To be safe, he tried the installation on his own workstation running Solaris with UFS file systems.

First he ran the step 1 of makeinstall to create snapshots for /, /usr, /var, and /opt:

```
# ./makeinstall bs:=/apps4 step=1
/dev/fssnap/3 493688 77196 367124 18% /snapshot/root
/dev/fssnap/2 2055705 1181127 812907 60% /snapshot/usr
/dev/fssnap/1 2055705 48364 1945670 3% /snapshot/var
/dev/fssnap/0 1915416 14166 1843788 1% /snapshot/opt
```

Then he ran

```
perl -MCPAN -e 'install Date::Manip'
```

to install the module.

After this, he ran the step 3 to find what files was installed or changed on his workstation:

```
# ./makeinstall bs:=/apps4 step=3
```

This produced a list of files:

```
/opt/csw/lib/perl/5.8.2/perllocal.pod
/opt/csw/lib/perl/site_perl/auto/Date/Manip/.packlist
/opt/csw/share/man/man3/Date::Manip.3perl
/opt/csw/share/perl/site_perl/Date/Manip.pm
/opt/csw/share/perl/site_perl/Date/Manip.pod
```

He looked at the files and figured out that the perllocal.pod is a shared file with entries for all locally installed modules. And the .packlist contains the list of files in this module.

Upon finding this, he was enlightened. He distributed the last 4 files to the developer’s workstation which was running VxFS and was not configured for CPAN access, and manually modified perllocal.pod.

Since the installation worked fine, he skipped the step 4 which would allow him to reverse the changes, but proceeded to step 5 to delete the snapshots:

```
# ./makeinstall bs:=/apps4 step=5
Deleted snapshot 3.
Deleted snapshot 2.
Deleted snapshot 1.
Deleted snapshot 0.
```

## CASE 2. Making sendmail packages

Andrew is a sendmail administrator. He currently has the Sun’s sendmail packages SUNWsndmr and SUNWsndmu installed, but he wants to build the sendmail package from the current source.

Andrew knows about makeinstall. So he ran the first step to create snapshots for all the operating system file systems. Then he ran “make install” to install the sendmail. He continued to run step 3 to produce a list of newly installed files. He was glad he used makeinstall, as the files were installed all over the place, mixed with other operating system files:

```

/etc/mail/helpfile
/etc/mail/statistics
/usr/bin/hoststat -> /usr/lib/sendmail
/usr/bin/mailq -> /usr/lib/sendmail
/usr/bin/newaliases -> /usr/lib/sendmail
/usr/bin/purgestat -> /usr/lib/sendmail
/usr/bin/vacation
/usr/lib/sendmail
/usr/lib/smrsh
/usr/sbin/editmap
/usr/sbin/mailstats
/usr/sbin/makemap
/usr/sbin/praliases
/usr/share/man/cat1/mailq.1
/usr/share/man/cat1/newaliases.1
/usr/share/man/cat1/vacation.1
/usr/share/man/cat5/aliases.5
/usr/share/man/cat8/editmap.8
/usr/share/man/cat8/mailstats.8
/usr/share/man/cat8/makemap.8
/usr/share/man/cat8/praliases.8
/usr/share/man/cat8/sendmail.8
/usr/share/man/cat8/smrsh.8

```

He saved the files. As his intention was to create a new package to replace Sun's packages, he reversed the changes by running step 4. He verified the integrity of the Sun's original packages with `pkgchk`, and then deleted the snapshots.

### CASE 3. Install CSW openssh package

Janet wants to replace Sun's ssh package with newer version. She knows and likes the blastwave distribution of Solaris software packages (<http://www.blastwave.org/>). Being a careful administrator, Janet downloaded the open ssh package, and examined the file list contained in this package using "`pkgchk -l`". She was glad that the package does not conflict with the Sun's package.

Janet knows about `makeinstall`, and wish to compare the files using both methods. Upon comparison of both lists, Janet found that the differences:

- `makeinstall` indicated that the files `/etc/passwd` and `/etc/shadow` were updated. She looked at the files and found that a "sshd nonpriv userid" sshd was created. Upon further investigation, she found a `useradd` command in `preinstall` script. Files created and modified in `preinstall` and `postinstall` scripts are not in the `pkginfo` list, but are captured by the catch-all `makeinstall` script with brute force.

What the userid was used for? Janet googled around and found all about privilege separation. Upon discovering this, she was enlightened.

- `makeinstall` also reported that

```
/opt/csw/etc/sshd_config
```

was created but was not in the `pkginfo` list. Janet found this file was generated in `postinstall` script:

```
cp -p $CONFDIR/$CONF_FILE.CSW $CONFDIR/$CONF_FILE
```

- `makeinstall` indicates the following files are installed or modified:

```
/var/sadm/install/contents
/var/sadm/pkg/CSWossh/install/copyright
/var/sadm/pkg/CSWossh/install/depend
/var/sadm/pkg/CSWossh/install/preremove
/var/sadm/pkg/CSWossh/pkginfo
```

Again this is because “makeinstall” reports what actually happened. Janet was glad to find how Solaris software packaging works.

By the way, the makeinstall also catches that after removing the package, the userid “sshd” created in the installation remained. The package removal should restore the system to the state as if the package was not installed at all, whenever possible.

#### **CASE 4. Run Oracle’s root.sh**

Ben is system administrator. He was asked to run “root.sh” as super user as part of Oracle software installation.

What does root.sh do? Is that safe to run? He looked at root.sh and had an idea what it would do, but decided to use makeinstall to find the actual changes.

Since Oracle software is installed on /myapp-ds1, which is not part of the operating system file systems, he specified an additional file system on the makeinstall command line:

```
makeinstall step=1 fs:=+/myapp-ds1 bs:=/apps2 snaproot:=/snapshot
```

Then he proceeded to step 2 to run root.sh. This is shown as follows:

```
# ./root.sh
Running Oracle9 root.sh script...

The following environment variables are set as:
  ORACLE_OWNER=oracle
  ORACLE_HOME=/myapp-ds1/ora01/app/oracle/product/9.2.0

Enter the full pathname of the local bin directory: [/usr/local/bin]:
  Copying dbhome to /usr/local/bin ...
  Copying oraenv to /usr/local/bin ...
  Copying coraenv to /usr/local/bin ...

Adding entry to /var/opt/oracle/oratab file...
Entries will be added to the /var/opt/oracle/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root.sh script.
Now product-specific root actions will be performed.
```

Then he ran step 3 to find what changes on the operating system file systems, and on Oracle’s own file system. He summarized the changes as follows:

(a) Three Oracle provided utilities were copied to a local binary directory:

```
/usr/local/bin/coraenv
/usr/local/bin/dbhome
/usr/local/bin/oraenv
```

(b) “oratab” was created:

```
/var/opt/oracle/oratab
```

(c) Three files were installed in /opt area:

```
/opt/ORCLfmap/bin/fmputl
/opt/ORCLfmap/bin/fmputlhp
/opt/ORCLfmap/etc/filemap.ora
```

(d) File permissions were changed for many Oracle files, for example:

```
/myapp-ds1/ora01/app/oracle/product/9.2.0/bin/*
```

(e) setuid root for the following two files:

```
/myapp-ds1/ora01/app/oracle/product/9.2.0/bin/dbsnmp  
/myapp-ds1/ora01/app/oracle/product/9.2.0/bin/oradism
```

(d) and (e) were discovered with the patched rsync code due to the rsync bug 1764 mentioned earlier (“Rsync Bug 1764”).

Ben supports a fail-over cluster with the Oracle software and database installed on storage shared among the nodes. Upon the node failure, the Oracle file systems are unmounted from the failed node, and mounted on another node within the cluster. The Oracle instance and Sqlnet listener are started on the new node.

Will the installed locally files affect the fail over capability? Will the setuid files present a security issue? Ben brought the issues to the database group. After the discussion, he was enlightened.

## CONCLUSION

“makeinstall” uses the “when in doubt, use brute force” technique to find the file system changes after running a command. The program can be used for software packaging, test run, and educational purpose. “makeinstall” is written for UFS on Solaris. However the idea can be extended to other systems.

The entire program is listed in Listing 1. Future version of this program, if any, can be found at:

```
<http://www.unixlabplus.com/unix-prog/makeinstall/>.
```

## SEE ALSO

"Rsync Bug 1433." Samba Team. Retrieved 13 Sep 2004.  
<[https://bugzilla.samba.org/show\\_bug.cgi?id=1433](https://bugzilla.samba.org/show_bug.cgi?id=1433)>.

"Rsync Bug 1764." Samba Team. Retrieved 12 Jan 2005.  
<[https://bugzilla.samba.org/show\\_bug.cgi?id=1764](https://bugzilla.samba.org/show_bug.cgi?id=1764)>.

"Rsync Web Pages." The rsync team. Retrieved 9 Sep 2004.  
<<http://rsync.samba.org/>>.

Wang, Michael. "Processing Command-line Arguments with my\_getopts." Shell Corner. Ed. Ed Schaefer. January 2003. Unix Review. Retrieved 19 November 2003  
<<http://www.unixreview.com/documents/s=7781/unil042138723500/>>.

## AUTHORS

Julie Wang works for Independence Air, <<http://www.flyi.com/>>. She manages Oracle databases, Unix operating systems, Lawson enterprise systems, and whatever to help the company flying. She can be reached at Julie.Wang@flyi.com.

Michael Wang is Julie’s husband. He can be reached at xw73@columbia.edu.

## COPYLEFT

This is free software. You may copy or redistribute it under GPL. However, if you modify it, please share your modifications with the authors for possible inclusion in the original program.

## VERSION HISTORY

2005-02-09, Julie Wang and Michael Wang.

\* Rewritten for Sys Admin,  
<<http://www.sysadminmag.com/articles/2005/0504/>>.

2002-08-18, Michael Wang, <xw73@columbia.edu>.

\* Workaround the ksh88 bug "pwd -P => //opt/ulocal (note //)".

\* Add "var=y|n" option.

\* Workaround the Solaris 9 fssnap by adding \_awk "{print \\${NF}"}\_".

\* Fix problem that "/usr/local" has no symlink: : \${T:=197001010101}.

\* Use "PATH=\$OPATH make install": make may use apxs (PHP).

\* "grep -Ev "rsync|unpredictable|lines" => [[ -r /usr/local/"\$line" ]].

2002-07-29, Michael Wang, <xw73@columbia.edu>.  
\* rewrite based on snapshot and rsync.

2002-01-22, Michael Wang, <xw73@columbia.edu>.  
\* Bug fix: save old link which could be updated during install.

2002-01-10, Michael Wang, <xw73@columbia.edu>.  
\* check for existence of /usr/local and /usr/local.4pkg at same time.

05/22/2001, mwan@mindspring.com  
\* check for existence of /usr/local and /usr/local.4pkg  
\* add rm option

05/06/2001, mwan@mindspring.com  
\* used my\_getopts, lock\_unlock functions.  
\* handles the symlink correctly.  
\* used GNU tar --no-recursion option.  
\* added touch, and makelocal options.  
\* can use exported env variable PKGDEST.  
\* added traps.

03/12/2001, mwan@mindspring.com  
\* minimal approach does not work, install requires existence of some  
utilities in /usr/local to run. Combine new version (file based) with

12/15/2000, mwan@mindspring.com  
\* Eliminate requirements in /usr/local.4pkg.  
\* More portable approach.  
\* Prerequisites:  
\* 1. Existence of /usr/local.4pkg/4pkg  
\* 2. Optionally PARFILE, PKGDEST, PKGNAME env variables  
old version (time based).

01/19/2000, mwan@mindspring.com  
IPO.