# rdist

BY CY SCHUBERT

## What is RDIST?

To quote the man page, "rdist is a program to maintain identical copies of files over multiple hosts." rdist is a general-purpose tool that can be utilized for multiple purposes, such as maintaining consistent copies of files across the network, like rsync and unison do, or as a distributed configuration management tool like cfengine, ansible, puppet, salt, or other configuration management tool.

## Why RDIST?

To understand why *rdist* understanding a little bit of its history will give us a better idea why it was created.

Written in 1983-1984 by Ralph Campbell at UCB, *rdist* first appeared in 4.3BSD (in 1986) and was one of the first applications to address the issue of distributed software management. During the late 1980s and 1990s it was distributed with almost every commercial UNIX. It became the standard for remote platform administration at the time.

*rdist* predates other software of its kind. It also predates rsync. Rsync is a backup tool used to backup or clone directory trees while *rdist* is generally used as a network file distribution application. Each is tailored to its design purpose.

With all these options, why use *rdist*?

- *rdist* is lighter weight than any of the subsequent configuration distribution applications such as ansible.
- *rdist* integrates easily into shell scripts and Makefiles.
- rsync is not able to distribute to multiple hosts in parallel like *rdist* can. Nor can rsync synchronize files using a configuration file, like *rdist* does. Rsync and *rdist* are designed for different purposes, rsync for backup and cloning of files while *rdist* works better as a configuration management tool.

On the flip side, why would one want to use another tool instead? As *rdist* is lightweight when compared to tools like cfengine and ansible, its ability to configure remote nodes on the network is limited to distributing files and performing simple post-distribution tasks. Whereas a heavier weight tool can perform pre-distribution tasks, this can be addressed through simple shell scripting or a Makefile.) As a personal example of this, I manage my ipfilter firewall rules using a configuration tool (called ipfmeta) to generate the firewall configuration files from a rules file and an objects file using a Makefile. The Makefile uses rdist to distribute the generated files to remote firewalls as defined in rdist's Distfile. One can think of a Distfile similar in relation to *rdist* as a Makefile is to make. Unlike rsync, *rdist* distributes files using rules coded in its Distfile.

## How Does RDIST Work?

Just as make(1) parses its Makefile to build applications, *rdist* parses its Distfile to describe what files or directories are to be distributed and any post-distribution tasks to be performed. Initially, *rdist* used the insecure rcmd(3) interface for network communication. rcmd() would make a connection to a remote rshd(8). When the connection is made it spawns an rdistd(8) remote file distribution server to perform the distribution functions on the remote server. This is similar to how ssh's sftp-server provides remote function to sftp.

The Berkeley "r" commands, such as rsh are insecure. Today's implementation of *rdist* can use ssh as a transport instead of rsh. With ssh one can use ssh keys or GSSAPI (kerberos) authentication. Unlike ansible where connection is made under your own account and privilege escalation is done using "become," *rdist* must connect directly to root on the destination server. To facilitate this PermitRootLogin can be set to prohibit-password in sshd_config, forcing the use of either ssh keys or Kerberos tickets.

*rdist* does no authentication itself. It relies on the transport for this. Compared to ansible, it also relies on the ssh transport for authentication and it relies on su(1), sudo(1) or ksu(1) for privilege escalation.

*rdist* can be used to manage application files in a service account such as mysql, oracle or other application account. Replace root with the desired account name.

rdistd(8) must be in the user's search path ($PATH) on the target server.

*rdist* negotiates a protocol version. The systuils/rdist6 port/package uses the RDIST version 6 protocol while sysutils/rdist7 (alpha) uses the RDIST version 7 protocol.

## Installing RDIST

To install *rdist*, simply,

```
pkg install rdist6
```

or

```
pkg install rdist7
```

Or using ports, using rdist7 as an example,

```
cd /usr/ports/sysutils/rdist7
make install clean
```

## Using RDIST

As noted previously, *rdist* uses a configuration file similar to how host make uses its configuration file. We must build our Distfile.

There are three types of Distfile statement.

### The Distfile

Like make, *rdist* looks for a file named Distfile or distfile. Just as we can override the name of the Makefile make uses, we can override the name of the *rdist* Distfile.

Distfiles contain a sequence of entries that specify the files to be distributed (copied), to which nodes those files are to be copied, and the post-copy operations that are to be performed following the distribution of the files.

### Variables

One or more items can be assigned to a variable using the following format.

```
<variable name> '=' <name list>
```

For example,

```
HOSTS = ( matisse root@arpa )
```

This defines the strings matisse and root@arpa to the variable HOSTS.
Another example assigns three directory names to the variable called FILES.

```
FILES = ( /bin /lib /usr/bin /usr/games )
```

## Distributing Files to Other Hosts

The second statement type tells *rdist* to distribute files to other hosts. Its format is,

```
[ label: ] <source list> '->' <destination list> <command list>
```

<source list> is the name of a file or a variable. <destination list> is a list of hosts to which the files will be copied. While <command list> are a list of rdist instructions to be applied to applied to the copy operation.

The optional label identifies the statement for partial updates when the label is referenced from the command line.

For example, from my firewall Distfile,

```
install-ipf: ipf.conf -> ${HOSTS}
      install /etc/ipf.conf ;
      special "chown root:wheel /etc/ipf.conf; chmod 0400 /etc/ipf,conf" ;
```

This tells *rdist* to install ipf.conf to the nodes listed in the HOSTS variable. The install command line tells *rdist* the file is to be installed to /etc/ipf.conf.

The special command line tells *rdist* to run chown and chmod following the copy operation.

The install-ipf label can be addressed on the *rdist* command line, limiting the operation to just that operation, i.e. rdist install-ipf.

The command list includes keywords such as install, except, special and cmdspecial.

| | |
|---|---|
| install | Identifies where to install target files. |
| notify | List email addresses to be notified upon completion of the copy operation. |
| except | An exception pattern of files not to be copied. |
| except_pat | Same as except but using a regexp pattern. |
| special | Shell commands to be executed after each file is copied. |
| cmdspecial | Shell commands to be executed after all files in a rule have been copied. |

A simple example follows. It copies my working copy of this article to a directory in my FreeBSD working directory tree.

```
HOSTS = ( localhost )

FILES = ( /t/tmp/rdist.odt )

${FILES} -> ${HOSTS}
        install /home/cy/freebsd/rdist/rdist.odt ;
```

Here we are copying the file /t/tmp/rdist.odt to /home/cy/freebsd/rdist/rdist.odt on my laptop. Of course, a simple cp(1) command would suffice, but this simple example gives us a taste of how to copy single files. Also note that the destination is a file by the same name. If the destination was a directory, i.e., /home/cy/freebsd/rdist, it would remove all the files and subdirectories in the target directory, replacing it with a single rdist.odt file. Be careful when specifying target files or directories. This would be like,

```
 rsync -aHW -delete /t/tmp /home/cy/freebsd/rdist
```

Unanticipated results can make for a bad day.

The rdist(1) man page provides a better example:

```
             HOSTS = ( matisse root@arpa)

              FILES = ( /bin /lib /usr/bin /usr/games
                       /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

              EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
                       sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

              ${FILES} -> ${HOSTS}
                       install -oremove,chknfs ;
                       except /usr/lib/${EXLIB} ;
                       except /usr/games/lib ;
                       special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

              srcs:
              /usr/src/bin -> arpa
                       except_pat ( \\.o\$ /SCCS\$ ) ;

              IMAGEN = (ips dviimp catdvi)

              imagen:
              /usr/local/${IMAGEN} -> arpa
                       install /usr/local/lib ;
                       notify ralph ;

              ${FILES} :: stamp.cory
                       notify root@cory ;
```

In the example above, files listed in the FILES variable will be copied from the localhost to the machines listed in the HOSTS variable. Except for files listed in the EXLIB variable, /usr/games/lib and a pattern. After each file is copied, sendmail with a -bz option is run.

Typically, special is used to run shell commands, but in the example above, /usr/lib/sendmail is executed (as if it were a shell), passing the quoted arguments to sendmail.

Three files in /usr/local will be copied to /usr/local/lib on the target systems, with an email to ralph when the copy has been completed.

A time stamp file is touched when the job completes, sending an email to root@cory. Time stamp files are used to avoid gratuitous copies. For example, if any of the listed files is newer than the time stamp file, the file is copied. (Conversely, ansible uses a checksum.)

## Gotchas

As mentioned, things can go wrong if one is not careful. Like rsync, *rdist* does not verify the source file is the same type of object (file or directory) as the target. It is easy to replace a destination file with a directory or replace a destination directory with a file. Like rsync, it can render a system unusable. Be careful and test in a sandbox or jail.

## Summary

*rdist* is an excellent tool when used in conjunction with scripts, makefiles, or other tooling in scenarios when no one tool can do everything, combined with other tools as I do to manage my ipfilter firewalls, ipfmeta, make Makefiles, *rdist* Distfiles, and git rdist integrates nicely to create a lightweight application. In the case of integration with heavier weight tools like ansible or cfengine which don't integrate with scripts and Makefiles, *rdist* fills that unique niche. *rdist* follows the original UNIX philosophy of a single tool for a single purpose that can be integrated with other tools to create new tools and applications.

## Bibliography

- https://man.freebsd.org/cgi/man.cgi?query=44bsd-rdist&sektion=1&apropos=0&manpath=FreeBSD+14.0-RELEASE+and+Ports
- https://www.magnicomp.com/download/rdist/overhaul.pdf
- https://www.cs.umb.edu/~ckelly/teaching/common/project/linux/sys_admin/p7_rdist.pdf

**CY SCHUBERT** is a FreeBSD src and ports committer. His career began over fifty years ago, writing and maintaining electrical engineering applications written in Fortran. His experience includes IBM MVS (mainframe) systems programming, writing extensions to the MVS kernel and Job Entry Subsystem 2 (JES/2). His career took a turn down the UNIX path thirty-five years ago moving to SunOS, Solaris, Tru64, NCR AT&T, DG/UX, HP-UX, SGI, Linux and FreeBSD systems administration.

Cy's FreeBSD journey also began thirty-five years ago. After trying a Linux distro with Linux kernel 0.95 and seeing it didn't support UNIX domain sockets, he tried an experimental Linux kernel. After a disastrous month of restoring EXTFS filesystems corrupted by the experimental kernel, Cy posted a query on the FreeBSD and NetBSD USENET newsgroups. The only person to reply to Cy's question was Jordan Hubbard from the FreeBSD project. Since Jordan was the first and only person to answer, Cy decided to try FreeBSD first. He's been using FreeBSD since 2.0.5. He became a ports committer in 2001 and a source committer eleven years ago. He is currently employed by a Canadian subsidiary of a large managed services provider.