

NAME

make-redir – run "make" but automatically redirect its results to DESTDIR

SYNOPSIS

make-redir [*MAKE_ARGUMENT**]...[*install*]

* DESTDIR=*directory* and -k/--keep-going are especially common.

DESCRIPTION

The make-redir program runs "make" in a way that redirects file installations to descend from the \$DESTDIR directory, even if the provided makefile doesn't natively support \$DESTDIR. In most cases, simply replace "make install" with "make-redir DESTDIR=*directory* install" (adding options as necessary).

The make-redir program attempts to solve a common problem for packagers of software and for users who directly install source code: Inadequate DESTDIR support. Software is typically installed on Unix/Linux from a source code package by typing "make install". Historically, this script would write directly to privileged directories to perform the installation (e.g., /usr/bin), or at least to a directory that merged all installed files (e.g., \$HOME/bin). However, packagers of software using most modern packaging systems (such as .rpm and .deb) instead require that files be written to some intermediate directory, even though they will be run from a different filesystem location. Similarly, users who directly install source code often use programs such as GNU stow, encap, or toast. These programs install each package in its own separate directory, but work best if the installed programs are invoked through a different directory (which merges the contents of multiple packages, simplifying the PATH and supporting plug-ins). Thus, for both many packagers of software and for many users who directly install source code, a program should be installed in one directory, yet actually be run from a different directory. This is easy to do if the installation script supports the "DESTDIR convention"; simply run "make DESTDIR=*something* install", which would install the files descending from DESTDIR, even though the files are configured to run without the DESTDIR prefix. Unfortunately, many source packages don't support the DESTDIR convention.

The auto-destdir tools "make-redir" and "run-redir" will automatically redirect file reads/writes to follow the DESTDIR convention, without requiring any changes to the build files (e.g., a makefile). Attempts to write to directories that are normally root-privileged (such as /usr/bin) will instead be redirected to descend from a designated destination directory. This destination directory is specified by the environment variable REDIR_DESTDIR or DESTDIR, with REDIR_DESTDIR taking precedence.

You should use make-redir (not run-redir) if you are redirecting "make", because make-redir performs additional helpful actions. If you pass DESTDIR=*something* as a parameter, it automatically sets the DESTDIR environment variable. The make-redir command also sets the overrides INSTALL="install -p" CP="cp -p" MKDIR="mkdir" MKDIR_P="mkdir -p" mkdir_p="mkdir -p" LN="ln", so that makefiles with absolute pathnames of common commands will work correctly and preserve timestamps. (Add overrides on the command line if these are not acceptable.) By default make-redir also sets special SHELL and MAKESHELL values, so that even invocations with a full path name (like /bin/cp) will be redirected if they begin a line in a makefile.

Internally, make-redir runs "make" using run-redir; run-redir does the actual file redirection. Use run-redir if you need to redirect file writes without first running "make".

The auto-destdir tools are designed to "just work" given the limitations described below (in particular, only "wrapped" commands should read/write files that must be redirected). Filenames will not be redirected twice, so it's safe to use run-redir and make-redir even with makefiles that do support DESTDIR. If a write is redirected, all parent directories are first automatically created if they exist in the unredirected filesystem. This means that an attempt to "cp" some file to /usr/bin/xyz will create the directory "\$DESTDIR/usr/bin" if /usr/bin already exists, then copy the file to "\$DESTDIR/usr/bin/xyz". If a file is being read, a file written inside \$DESTDIR will have precedence over a file that isn't; this nicely simulates the overwriting of an old file with a new one. Since make-redir and run-redir set the PATH, they will work even if there are many levels of indirection, and even if they use tools other than make (as long as they end up eventually invoking the usual commands such as cp and install).

If the "make install" works except that it uses a few "unwrapped" commands, consider using the "-k" ("--keep-going") option, so that make-redir will keep going instead of failing on an error. Then, re-perform

the actions that would have failed. Alternatively, you may be able to perform one or two actions "by hand" first, and then invoke make-redir to do the rest.

The programs make-redir and run-redir can be run as root, but in most cases, you should not run them as root. The installation program itself may require root to run (e.g., to set ownership), even if files are written to DESTDIR. If you just want to record permissions (e.g., chown, chmod), consider also using "fakeroot" to record the intended permissions instead of running as root. When creating an rpm package, the permission-setting is typically irrelevant, since in most cases it is the rpm spec file that will ultimately set the permissions. If the permission-setting is irrelevant, you may want to run make-redir as non-root and pass it the "-k" ("--keep-going") option so that make will keep going instead of failing on an error.

A file/directory is not redirected if its full pathname begins as a typical user directory (specifically /home/ or /Users/) or temporary directory (/tmp/, /usr/tmp/, /var/tmp/, or /opt/tmp/). A file/directory is redirected if its full pathname begins with a directory name that is normally privileged: /usr/, /etc/, /boot/, /bin*/, /dev/, /lib*/, /opt/, /sbin*/, /selinux/, /srv/, /sys/, and /var/. If neither case applies, it is not redirected.

You can override some values, though few will need to do so. By default, make-redir invokes "make"; this can be changed by setting the environment variable REALMAKE to whatever "make" program should be invoked instead. By default, make-redir sets a special SHELL value (so it can override commands that give full pathnames); this special value does some processing and then invokes /bin/bash. Set the environment variable REALMAKESHELL to the name of the shell to use after doing this filtering if you do not want it to use /bin/bash. You can disable this filtering entirely by setting the make variables SHELL and MAKESHELL on the command line to the name of the shell program to use (e.g., make-redir SHELL=/bin/sh MAKESHELL=/bin/sh); if make can be called recursively, also set the environment variables with the same name. If you do not want any makefile variable overrides, set the environment variable DESTDIR and use "run-redir make ..." instead.

EXAMPLES

The following configures, builds, and then "installs" a program. The "installed" files will descend from /tmp/stuff, e.g., directory /tmp/stuff/usr/bin will contain files that were intended to be installed in /usr/bin:

```
./configure --prefix=/usr
make
make-redir DESTDIR=/tmp/stuff install
```

For example, if "cp myprogram /usr/bin" is one of the instructions in the makefile for the "install" target, then (presuming /usr/bin exists) it will actually perform the equivalent of:

```
/bin/mkdir -p /tmp/stuff/usr/bin
/bin/cp myprogram /tmp/stuff/usr/bin
```

You can use make-redir inside an RPM .spec file. Be sure to "BuildRequires" auto-destdir, and set the DESTDIR to be the RPM build root, as follows in the spec file:

```
...
BuildRequires: auto-destdir
...
%install
rm -rf "$RPM_BUILD_ROOT"
make-redir DESTDIR="$RPM_BUILD_ROOT" install
```

Some RPM spec authors may prefer to use %{buildroot} instead of \$RPM_BUILD_ROOT.

IMPLEMENTATION APPROACH

The auto-destdir tools (including run-redir and make-redir) work by modifying the PATH environment variable and providing a few "wrappers" that override typical installation commands like "cp" and "install". These commands with wrappers, called "wrapped commands", use the values of \$REDIR_DESTDIR or \$DESTDIR (with REDIR_DESTDIR taking precedence), and will redirect reads and writes of files under certain conditions. The "redirection" comes by simply changing the name of the file source or destination, and then calling the "real" program being wrapped. The internal function redir_name takes the original filename, and how it will be used (e.g., for reading or writing), and determines if it should be renamed (and

if so, to what). In addition, `make-redir` redirects all shell commands (using `SHELL` and a special wrapper for shell commands); any command that begins with the prefix `/usr/` or `/usr/bin/`, followed by a wrapped command's name, will have that prefix removed.

This may seem to be an odd implementation approach, but it has many advantages. The "obvious" way to implement redirection is to manipulate `LD_PRELOAD` and wrap the `open()` and `creat()` calls. But as the `fakeroot(1)` documentation notes, this approach creates "problems, as demonstrated by the `libtricks` package. This package wrapped many more functions, and tried to do a lot more than `fakeroot`. It turned out that a minor upgrade of `libc` (from one where the `stat()` function didn't use `open()` to one with a `stat()` function that did (in some cases) use `open()`), would cause unexplainable segfaults (that is, the `libc6` `stat()` called the wrapped `open()`, which would then call the `libc6` `stat()`, etc). Fixing them wasn't all that easy, but once fixed, it was just a matter of time before another function started to use `open()`...". `LD_PRELOAD`-based approaches fail on commands that are statically linked, and the commands usually used for installation (e.g., `cp`) are often statically linked. Others tools (like `fakeroot`) also use `LD_PRELOAD`, so using `LD_PRELOAD` means that these other tools cannot be used simultaneously. Other "obvious" approaches are to use `ptrace()`, or redirect file writes using the kernel (e.g. with `chroot`), but these turn out to have many problems too, as discussed in <http://www.dwheeler.com/essays/automating-destdir.html>.

In contrast, since `run-redir` and `make-redir` don't manipulate `LD_PRELOAD` to manipulate `open()` or `creat()`, they are immune to deadlocks caused by changes in the C library, work with statically-linked commands, and work with tools like `fakeroot`. No special privileges are needed to run these programs, nor do they require `setuid` installation, eliminating many security concerns (compared to a `chroot`, `setuid`, or kernel-based approach). No kernel driver is used, nor is `ptrace()` used, so these tools should be fairly portable to any POSIX-based system with `bash`. All work happens very quickly, since nearly all commands simply run as usual, and even the wrapped commands merely run a short shell script before running as usual. The implementation is entirely a set of small `bash` scripts with trivial dependencies, so the `auto-destdir` tools can be easily included in a package's build process without slowing the build process down (even if the build process requires re-installing dependencies on each run).

In practice, this approach is sufficient to automate `DESTDIR` for typical installation scripts. Typical "make install" programs, even if they are deeply recursive, only use a very few commands to actually write files to their final locations in trusted directories, and usually invoke commands without using the full pathnames of the commands (e.g., "install -p myprog /usr/bin"). As a result, typical installation scripts are easily redirected by this program.

More information about various implementation alternatives, their pros, and their cons, is here: <http://www.dwheeler.com/essays/automating-destdir.html>

LIMITATIONS

Only wrapped programs will be redirected to the new destination directory; any other program will not be redirected. Currently, the "wrapped" commands are (alphabetically):

chmod, *cp*, *install*, *ln*, *mkdir*, *mv*, and *touch*.

If the `makefile` invokes wrapped programs using absolute pathnames (e.g., calling `"/bin/cp"` instead of `"cp"`), then the command will be redirected anyway if it is the absolutely first command at the beginning of a line in a `makefile` and is invoked directly or via `makefile` variable expansions like `$(CP)`. Absolute pathnames at the *beginning* of a line are handled because `make-redir` sets `SHELL` to a special shell wrapper that handles this particular case. However, absolute pathnames for wrapped programs are not handled in any other place (doing so would require the shell wrapper to re-implement shell parsing); e.g., program names stored in shell variable expansions will not be intercepted. Thus, `makefile` lines like `"cd X; /usr/bin/install myprogram /usr/bin"` will not be redirected. In these cases, you should change absolute pathname invocations like `/usr/bin/install` to unnamed-path invocations like `install`.

Capabilities built into the shell are not redirected, including I/O redirection such as `"<"`, `">"`, and `">>"`. This means that `"cat x > /usr/bin/program"` will not be redirected; use `"cp x /usr/bin/program"` or similar instead. Similarly, `"test"` and `"["` (which are normally shell built-ins) aren't redirected. Changing this would require `auto-destdir` to re-implement a full shell.

The wrappers presume GNU command line options, and will not understand other options even if the underlying tool would. In practice, this is almost always irrelevant; "make install" typically only uses highly-portable commands to copy files and make directories, and does not use unusual options.

The "rm" and "rmdir" commands are not redirected simply because these are unlikely commands during a "make install". Similarly, there are no "whiteout" files to record when a file is removed from \$DESTDIR, because this does not seem useful for "make install".

In cases where this wrapper is not sufficient, you may need to modify the installation script. If significant modifications to the installation script are required, you may as well add DESTDIR support directly to the installation script. The auto-destdir tools are designed to handle common cases, so that people can focus their time on the cases that need special handling.

GUIDANCE FOR SOFTWARE CREATORS

When writing the commands for "make install":

- Only use the wrapped commands to install files and directories in their final positions (e.g., install, cp, ln, and mkdir).
- Don't use full pathnames of wrapped commands (e.g., "/usr/bin/install"); use their basename (e.g., "install").
- Don't use redirection (e.g., ">") to write to files in privileged directories; use redirection to write to a "local" file (typically during the build process), and then use "install" or "cp" to install the result.

Supporting DESTDIR is still a good idea.

BUGS

As noted above, only "wrapped" commands are redirected. Some commands that aren't wrapped, and perhaps should be, include: install-info, ldconfig, rm, rmdir, chown, chgrp, and libtool. For permissions-related commands (like chmod, chown, and chgrp), consider also using fakeroot or make's "-k" option.

Options with optional arguments cannot accept arguments that begin with "-", due to limitations of getopt(1).

cp's --parents argument is supported, but it won't work if the original files are being copied from a redirected location.

When running make-redir, the makefile's SHELL value is ignored. If you need to use the SHELL value provided by the makefile, run make-redir and provide "SHELL=..." with the correct shell value. If you do this, wrapped commands must never be invoked using their full pathname.

AUTHOR

David A. Wheeler

REPORTING BUGS

Report bugs to <dwheeler, at, dwheeler dot com>

COPYRIGHT

Copyright © 2009 Institute for Defense Analyses. Released using the MIT license.

This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

SEE ALSO

run-redir(1)