NAME

rshe − execute a command on a remote machine with local environment

SYNOPSIS

rshe [**-l** *rsuer*] [**-n**] [**-d**] [**-e** *environ_file*] [**-x**[=*var-list*]] [**-V**] [**-?**] *hostname command*

AVAILABILITY

BCS Starbase Project on Sun SPARC hardware only, SunOS 4.1.3 or higher, SunOS 5.4 or higher

DESCRIPTION

The **rshe** program connects to the specified *hostname* and executes the specified *command*. This program is very similar to the **rsh** program and is inspired by it and the **RX** program which is part of the **REXEC** remote execution package (not related to the common Berkley **REXEC** daemon) developed at AT&T Bell Laboratories. The **rshe** program uses the same daemon program on the remote machine as the **RSH** program does. This is both a blessing and a curse. It is a curse since the program's behavior is restricted somewhat by the fact that the same daemon is used as **RSH** uses. It is a blessing because many remote machines now-a-days will not be running any other remote execution daemons and we still want to get the best out of an already bad machine execution environment. Normally, the entire local environment is sent over to the remote machine (because this is what is usually desired in a tightly coupled local machine group) but this behavior is changed if the **-x** option is used.

The **rshe** program copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **rshe** normally terminates when the remote command does.

If you omit *command*, the **rshe** program will try to execute whatever your SHELL environment variable is set to. If there is not SHELL environment variable, it will try to execute '/bin/sh' on the remote machine. Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. See EXAMPLES.

Hostnames are given in the *hosts* database, which may be contained in the **/etc/hosts** file, the Internet domain name database, or both. Each host has one official name (the first name in the database entry) and optionally one or more nicknames. Official hostnames or nicknames may be given as *hostname*.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rshe** from the machines listed in the remote machine's **/etc/hosts** file. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. The entry permits the user named *username* who is logged into *hostname* to use **rshe** to access the remote machine as the remote user. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's **.rhosts** file, then the access is denied. The hostnames listed in the **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the **hosts** database; nicknames may not be used in either of these files.

rshe will not prompt for a password if access is denied on the remote machine.

OPTIONS

Options can appear anywhere on the command line, either before or after the specification of the remote host, but must appear before the specification of the remote command. Option key letters can also be freely concatenated together. If a concatenated option key requires an associated value string, it is picked up as the next tokenized string on the command line.

−l *username* Use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username.

−n Redirect the input of **rshe** to **/dev/null**. You sometimes need this option to avoid unfortunate interactions between **rshe** and the shell which invokes it. For example, if you are running **rshe** and invoke a **rshe** in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The **–n** option will prevent this.

–x[=[var_list]] The **- x**option is used to send some local environment variables over to the remote system instead of all of the local environment variables. This is useful where local environment variables, like **PATH** and **LD_LIBRARY_PATH** may not have the proper meaning on the remote system. If the **- x**option is NOT specified, then the local environment variable **RXPORT** (another inspiration of the AT&T **REXEC** package) is consulted for a list of local environment variables to send over to the remote machine. This variable is ONLY used if no **- x** option is specified on the program invocation. When used, this variable should contain a comma separated list of environment variable names (no intervening white space between list names). Only these named environment variables are sent over to the remote execution environment but only if the environment variable exists on the client machine.

–e environment_file
The **- e**option is used to have an environment file executed on the remote machine before the user specified command is executed. For heterogeneous machine environments, ideally, each different machine could have a file in the user's home directory that could be executed to provide the proper environment for the user requested command. Normally, a user's **.profile** type file serves this function but is not always the best choice when executing many remote commands. Usually, a file is created with a list of environment variables which are the essential set needed for proper remote execution of commands. This file should be able to be executed by **/bin/sh** and should export any environment variables that are desired to be in the environment of the user requested command to be executed. This file can be the user's **.profile** type file but that may have undesirable side effects. The specified file (what ever it may be) is executed with its standard input, output and error redirected to the **/dev/null** file. The specified environment file specified will not have a controlling terminal either. If this option is used, the specified file is executed before any further environment is sent over either by default (the entire existing local environment) or those local environment variables specified with the **- x**option.

–d This option is used to have the remote command executed in the same directory as the current working directory on the local machine. If the remote machine cannot find or change to this directory, it exits with an error return code. The default directory where the user's specified command is executed is the user's home directory on the remote machine.

–V This option will cause the program to print its version to the standard error output and then it will exit.

–? This option will cause a brief help message to be printed to standard error output and then the program will exit.

ENVIRONMENT VARIABLES

RXPORT This variable is used if no "- x" option was specified at program invocation. This variable should contain a comma separated list of environment variable names. These names and the associated values of these variables from the client machine will be sent over to the remote machine environment if the corresponding local variable exists.

LOGNAME This variable contains the local user login name. It can be changed by the user under certain circumstances to specify alternate login names for certain operations. Note that this variable can never be used to circumvent security (assuming no bugs anywhere) since a check is always performed to verify that the variable contents can be authentically associated with the current read UID. It is really only useful to play with this variable when a single UID has two or more valid username names associated with it.

EXIT CODES

Returns ***0*** upon successful completion, ***1*** otherwise.

EXAMPLES

The following command:

```
example% rshe lizard cat lizard.file >> example.file
```

appends the remote file **lizard.file** from the machine called ‘‘lizard’’ to the file called **example.file** on the machine called ‘‘example.’’

FILES

/etc/hosts
/etc/hosts.equiv
${HOME}/.rhosts
/etc/passwd

SEE ALSO

rsh(1), **rex**(1), **rl**(1), **rx**(1), **rexec**(1), **rlogin**(1), **vi**(1), **in.named**(1M), **in.rshd**(1M), **hosts**(4), **hosts.equiv**(4), **passwd(4)**

NOTES

When a system is listed in **hosts.equiv**, its security must be as good as local security. One insecure system listed in **hosts.equiv** can compromise the security of the entire system.

You cannot run an interactive command (such as **vi**(1)); use **rlogin** if you wish to do so. This is a major short coming of I/O development on the UNIX operating system platform. Even with STREAMS, I/O has not progressed significantly to even match some of the capabilities of other (now almost obsolete) operating systems. How will the UNIX operating system stack up to MS- Windows, MS- Windows NT, or Apple’s MacOS ? we will see in time.

Stop signals stop the local **rshe** process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

Sometimes the **–n** option is needed for reasons that are less than obvious. For example, the command:

```
example% rshe somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf –
```

will put your shell into a strange state. Evidently, what happens is that the **tar** terminates before the **rshe**. The **rshe** then tries to write into the ‘‘broken pipe’’ and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking **rshe** with the **–n** option avoids such incidents. This bug occurs only when **rshe** is at the beginning of a pipeline and is not reading standard input. Do not use the **–n** if **rshe** actually needs to read standard input. For example,

```
example% tar cf – . | rshe sundial dd of=/dev/rmt0 obs=20b
```

does not produce the bug. If you were to use the **–n** in a case like this, **rshe** would incorrectly read from **/dev/null** instead of from the pipe.

CAVEATS

A significant difference from this program and the more popular **RSH** program is that this program will not execute multiple commands enclosed in quotes like the **RSH** program will. This program will only execute a single supplied program along with its arguments (no funny business).

The **- n**option should not have to exist. It does exist because of some fundamental flaws in the way in which I/O is handled in both old (non- STREAMS) and new (STREAMS) versions of UNIX. These I/O limitations do not exist in operating systems such as DEC’s RSM- 11M or DEC’s VMS but, of course, these operating systems suffer from other problems (possibly quite numerous) also. The **- n**option simple directs the **rshe** program to not read its standard input and to instead return an end- of- file indication to the remote command’s standard input. Normally, the **rshe** program continuously, and naively, reads its standard input because it is too stupid to know when the remote command issues a read to its standard input. This behavior can be ultimately traced to a fundamental flaw in the I/O subsystem of the UNIX operating system. The UNIX operating system's I/O STREAM head (STREAMS is assumed for this discussion) does not provide any indication of how much data is requested by the program above the STREAM head. For this reason, any and all STREAMS modules and ultimately programs residing on the client side of a network connection have to assume that the server side program continuously wants input data. This could have been long ago fixed in UNIX (circa 1982 at least) but no...ooo! The powers that were thought that they knew how to make operating systems and did not understand the need for a formalized "attention" type indication from programs issuing reads on the other end of a communications pipe (network or otherwise).

There are too many "remote execution" programs and most all of then, like this one (RSHE), are poor excuses for the original real thing ; namely, REXEC nad its children RL, RX, et cetera from AT&T. Sun Microsystem's ON program and its daemon RPC.REXD have the potential to come close to the AT&T original but it is not even as good as it could be. Finally, be warned, at the time of this writing, that Sun's ON remote execution facility is extremely buggy, often leading to machine crashes of the client side, server side or both! It's no wonder that it is usually, and shipped from Sun as so, turned OFF!

AUTHOR

David A.D. Morano