Open Computing ‘‘Hands-On’’: ‘‘Wizard’s Grabbag’’ Column: March 94

Terminal Printing

Printing to a terminal or PC printer port, and managing those insecure `.rhosts` files with a Unix power tool.

Many users access Unix systems using communication programs with terminal emulators on PCs. They often want to print a file that resides on the Unix host, but if they use the Unix print spooler (the BSD `lpr` or System V `lp` commands) the output goes to a printer connected to the Unix system or the network, not to the printer connected to the user’s PC. Greg Hinkel’s `termprint` program sends one or more specified files to the printer port on your terminal or PC. Each file is bracketed by terminal control sequences that enable or disable the local printer.

This month’s Unix Power Tool helps you manage those convenient but insecure `.rhosts` files, which establish user equivalence for the networking ‘‘r’’ commands.

Printing From Your Terminal

Dear Editor:

Here’s a simple program [Listing 1A] I wrote about two years ago that allows a user communicating with a Unix system to print text on a printer connected to their terminal-printer port. The `termprint` program accepts a Unix file as input and can print it on the printer connected to the user’s PC. Printing is started when the terminal receives a special printer-on code and stops when it receives the printer-off code.

For instance, the VT102 terminal was the first terminal Digital Equipment Corporation manufactured with a printer port. Data received by the VT102 could be sent to the printer port (in addition to the screen) by sending the printer-on code (`ESC[5i`) to the terminal. Receipt of the printer- off code (`ESC[4i`) disables the printer port. Most terminal emulators support the VT102 terminal and recognize these escape codes.

Invoke `termprint` with one or more arguments that name the file(s) you wish to print. The program is also designed to print its standard input. [Listing 1B shows examples of both invocations].

The printer-on code may skew the output if it’s included as part of the first page to print, so `termprint` inserts a form-feed character after the printer-on code and before the text begins. Also, a form feed is output before the printer-off code.

My program has been tested under Ultrix 3.x, Ultrix 4.x, SunOS 4.x, Solaris 2.x, OSF/1 1.2, and AIX 3.2. Because it’s a Bourne shell script and uses relatively standard Unix commands, it should run on most Unix hosts without modification.

Greg Hinkel / Oak Ridge, Tenn.

Tester's Comment: This program works fine on SCO systems communicating from a DOS system using Kermit 3.13. I managed to send a 100-kilobyte file safely to my printer, which surprised me a bit. The intention was to see what happened when the buffers filled up. However, in this case, Kermit stopped the modem--with hardware handshaking--nicely. If you do have a slow printer with little or no buffering or if you can't use hardware handshaking, your session may hang. Maybe add a `-p` *`DELAY`* option to install a simple pacing mechanism, like so:

```
# If p-option specified:
while read LINE ; do
    echo "$LINE"
    sleep $DELAY
done < $Tmpfile
```

Of course, lines that begin with a dash would be a problem if your version of `echo` recognizes options.

And how about printing nontext files, like Wordperfect documents? It wouldn't work with the terminal in canonical mode. Perhaps your readers should include a test to prevent printing of nontext files? *-Kees Hendrikse*

Tester's Comment: Terminfo descriptions normally do not include printer control commands. In fact, the Terminfo description of a VT220 with a printer isn't the same as the bare VT220.

Digital Equipment Corp.'s notion was a bit different when they designed the VT100 family: DEC's operating systems (RSX, VMS) asked the terminal with a proprietary escape sequence about installed options. The answer then was parsed to see printer option, keyboard type, and other stuff. The Terminfo mechanism isn't rich enough to support this approach.

If we have a real terminal with printer, we can name the pair as vt220+printer and then propagate the terminal name over the network. However, the situation is worse if we have some terminal-emulator over TCP/IP, such as FTP Software Inc.'s PC/TCP tn or Sun Microsystems Inc.'s PC-NFS Telnet. They claim themselves--via Telnet protocol--to be dec-vt220 and vt100 accordingly. In this case, every host on the local-area network should ask us about the value to be stored in the `TERM` environment variable.

I have one complaint about the `termprint` utility: The terminal may not have ''mc5'' and ''mc4'' Terminfo capabilities, but may be capable to print. By that I mean terminals with the ''mc5p=*n*'' Terminfo or ''pO=*n*'' Termcap capability, which enables the printer for ''*n*'' characters where ''*n*'' is less than 255.*-Endre Balint Nagy*

Implementation Notes: Line 11 uses `tr` to translate an arbitrary character--here a capital "X"--to the ASCII code represented by octal value 014, which is a form feed. This code value is then assigned to the `CRTL_L` shell variable.

Lines 12 and 13 provide locations for initializing the printer -on and printer-off codes if the Unix system doesn't have `tput` to retrieve these codes automatically from the Terminfo-capability database. Alternately, if you use Termcap, you could employ `tcap` instead of `tput`. [We provide the source for Jerry's `tcap` in Listing 2]. The `tcap` program is also documented in *Unix Power Tools*, by Jerry Peek, Tim O'Reilly, and Mike Loukides (O'Reilly and Associates Inc., 1993, ISBN 0-553-35402-7), and is distributed on the CD-ROM that accompanies the book.

Lines 28-41 fill the temporary file named by the `Tmpfile` variable with the text to be printed. If the command-line argument count is zero, line 33 redirects the standard input into the temporary file, after a form feed. Otherwise, lines 34-40 test that the files named on the command line are readable and, if so, appends their contents to the temporary file.

If line 44 determines that nothing has been written to the temporary file, the program terminates. Lines 49 and 50 check whether the printer-on or printer-off codes have been defined. If not, the `tput` command retrieves these codes. The ‘‘mc5’’ capability is the terminal-printer-on code, and ‘‘mc4’’ the off code.

The colons beginning lines 49 and 50 prevent the shell from attempting to execute the command named by the `PrintOn` and `PrintOff` variables. The colons let the variables be assigned to without interpreting the result as a command to execute.

If either printer code hasn’t been initialized, lines 53-56 report the fact and terminate the script.

Finally, lines 59 and 60 output the printer-on code, the form feed, the text to be printed, and the printer-off code.

Managing ‘‘rhosts’’

Dear Editor:

One of the handiest things about TCP/IP networking commands like `rlogin` and `rcp` is that you don’t need to type your password on the remote host. All you have to do is add a `.rhosts`file to the home directory on your remote account. The problem is that the `.rhosts` file can be a security hole. If you leave it in your remote home directory, crackers can access that account once they have access to your account on other machines.

I’ve included a shell script [Listing 3A] that allows you to enable or disable a remote `.rhosts` file. This script was one that didn’t make it into *Unix Power Tools*," the book I coauthored.

You should install this script with two names (that is, links): `rhyes` for `.rhosts` "yes" and `rhno` for `.rhosts` "no." Invoke each script with an argument that specifies the remote host to rename the `.rhosts` file for your account on that host. The `rhno` script renames `.rhosts` to `.rhosts.moved` so it won’t be found by the ‘‘r’’ command. Conversely, the `rhyes` command renames `.rhosts.moved` to `.rhosts`

The scripts use `ftp` to connect to the remote host and rename the target file. The `-n` option is specified for `ftp`, so any local `.netrc` file--which may provide for automatic log-in-is bypassed. Thus, the script user has to supply the password interactively for his or her remote account.

The scripts works on your local host, too. That is, invoke the script without a host name argument to rename the `.rhosts` file for your local account. [See Part B of Listing 3 for some usage examples.]

The script won’t stop crackers who edit your `.rhosts.moved` file then wait for you to run `rhyes` nor will it stop users who overwrite this script so it runs different commands, but it will help manage `.rhosts` files.

Install this script in a directory in your command-search path. Name the file `rhyes` make it executable, then create a link named `rhno`

Jerry Peek / O'Reilly & Associates Inc. / Sebastopol, Calif.

Editor's backgrounder: Unix systems have their own network applications comparable to the standard Telnet and Ftp TCP/IP applications, namely `rlogin` and `rcp` respectively. Additionally, the `rsh` command, which has no TCP/IP suite equivalent, executes a specified command on a remote machine.

These so-called ‘‘r’’ commands allow password-free access to "trusted" hosts and users. Two databases establish trust: `/etc/hosts.equiv` defines trusted hosts and users at the system level, and `$HOME/.rhosts` defines the trusted hosts and users for an individual user account.

The ‘‘r’’ command first consults the `/etc/hosts.equiv` file, followed by the `.rhosts` file for the user. The first match generally determines the action, so, except for "net groups," an entry in `.rhosts` doesn't override an entry in the `/etc/hosts.equiv` file.

Users often install an `.rhosts` file in their home directory to allow password-free access to a system that's not listed in the `/etc/hosts.equiv` file. For instance, user ‘‘beccat’’ on host ‘‘tventure’’ could log in without a password to an account with the same name on host ‘‘yin’’ if the latter account has an `.rhosts` file in the home directory containing the entry line: ‘‘tventure beccat’’.

Users might employ this file to establish equivalence for their various accounts. For instance, user ‘‘beccat’’ on host ‘‘tventure’’ could log in without a password to account ‘‘rathomas’’ on host ‘‘yin’’ if the latter account has an `.rhosts` file in the home directory containing the line: ‘‘tventure beccat’’.

This file can prevent password-free access for a user that happens to have the same account name on a remote machine. For instance, if user ‘‘beccat’’ on host ‘‘tventure’’ wishes to prevent password-free access by user ‘‘beccat’’ on ‘‘yin’’, the former account could have an `.rhosts` file in the home directory containing the entry: ‘‘yin -beccat’’.

The `.rhosts` file is necessary to allow password-free access for a remote superuser. For instance, to allow ‘‘root’’ on ‘‘yin’’ to access ‘‘root’’ on ‘‘tventure’’, the latter host uses a `/.rhosts` file containing the entry: ‘‘yin root’’.

Individual `.rhosts` files let the user control system security, which can be a problem. In fact, many site administrators disallow `.rhosts` files entirely. However, users often want to use these powerful network applications with password-free access. Of course, such sites should not be connected to the Internet else a cracker could masquerade as a legitimate user to gain unauthorized ‘‘r’’-command access.

Acknowledgments

I wish to thank the following readers for their help with testing this month's contributions (operating system): Kees Hendrikse, Echelon Consultancy, Enschede, The Netherlands (current SCO UNIX and Xenix versions); Endre Balint Nagy, Walton Networking Ltd., Hungary (Unixware Application Server, 1.0); and Steve Wright, CS Dept., University of South Carolina, Columbia, S.C., USA (AIX 3.2).

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content   Search

Last Modified: Tuesday, 22-Aug-95 15:46:37 PDT