Cygwin, Part Three

By Jerry Peek

The January 2004 "Power Tools" column "Cross-Platform Command Lines" (http://www.linux-mag.com/2004-01/power_01.html) presented ways to use the *Linux*-like features of Microsoft's shell *CMD*. That shell still requires some thought, though; you can't just type a Linux command and expect it to work just as it would in, say, *bash*.

Windows command-line utilities like *FIND.EXE* have evolved since the days of *DOS*, and some of them are quite handy. Of course, Windows also comes with graphical utilities like *Windows Explorer*, and there are thousands of third-party graphical tools for Windows. Yet in a lot of cases, those Windows applications can't match the power of the time-tested (and time-improved) GNU shells and utilities. The *Cygwin* package (http://www.cygwin.com/) is a Linux-like environment for Microsoft Windows that includes those standard shells and utilities.

In this third (and last) article of a series about Cygwin, let's look at some of the things that can be a struggle with standard Windows tools, but are easy to do with Cygwin utilities. We'll also see how to use Cygwin scripts and utilities from Windows shortcuts and menus.

Running Shell Scripts From Outside Cygwin

Because the Linux shells have many more features than the Windows *CMD* interpreter, you may want to run shell scripts instead of Windows batch files.

To run a shell script from outside of a Cygwin shell — such as a Start Menu shortcut file or a *CMD* prompt — you'll need to locate the shell executable, pass arguments to it, and possibly set the starting directory. If you make a Windows shortcut that points to the script, Windows will run the application in a text window by default. You can also set window properties like the font, window size, and the length of the command buffer. All of those help you see the results and scroll back to read all of the script's output. When the application exits, the window will close.

Listing One shows a *bash* script named *watchload*. It's designed to be run from the Cygwin user's home directory, where a configuration file named `.watchload-hosts` holds a list of remote hosts to monitor with the Cygwin *uptime* utility. The script has an endless `while` loop that uses the *bash* builtin command *read* to pause 60 seconds (`-t 60`) for the user to type a single character (`-n 1`). If the user doesn't type `q` or `Q`, or if `read` times out, the `while` loop repeats. (The script is a bit contrived, albeit to demonstrate things about running a script from a Windows shortcut file.)

LISTING ONE: The *watchload* script

```
#!/bin/bash
# watchload - run uptime(1) on remote hosts
# Run from $HOME — which has $hosts file
PATH=/cygdrive/c/cygwin/bin
hosts=$(cat .watchload-hosts)
while :
do
   date
   for host in $hosts; do
     ssh $host uptime
   done
   echo -n "To quit, type q: "
   read -t 60 -n 1

   case "$REPLY" in
   [qQ]) exit ;;
   *) echo -e '\n' ;;
   esac
done
```

Figure One shows the shortcut properties dialog as well as a window with the script running inside. The shortcut target is the Windows-style pathname (using backslashes) of the *bash* shell; the shell's argument is a Cygwin- style pathname (using forward slashes) to the script file. The different slash styles are because Windows locates the Cygwin shell through a Windows pathname, but the shell expects to see a POSIX-style pathname argument. To make the script run from the current user's home directory, the "Start in:" field points to the Windows path of that folder. The icon is set to a magnifying glass.

You can do the same thing from a *CMD* prompt by typing the following commands:

```
C:\> cd \cygwin\home\jpeek
C:\cygwin\home\jpeek> \cygwin\bin\bash
bin/watchload
```

Sending Files to a Script

For some users, an advantage of a window-based environment like Windows is that choices are listed on a menu somewhere. A Cygwin shell prompt, on the other hand, basically forces the user to know what they're doing and how to do it.

As you just saw, a good compromise is writing shell scripts that can be run from a Windows shortcut file. You can also

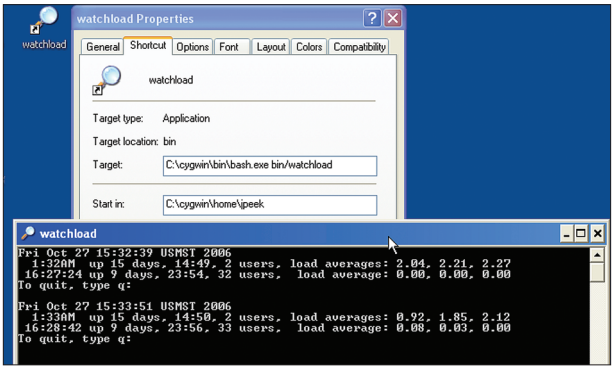

FIGURE ONE: The *watchload* shortcut properties and window

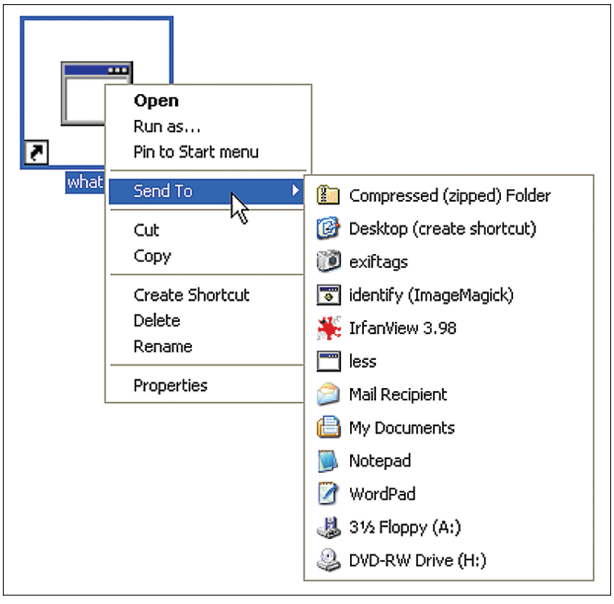

FIGURE TWO: The *Windows XP* "Send to" menu

install shell scripts and utilities on the Windows "Send to" context menu. (If you haven't seen this menu, point to an icon, click the right mouse button, and choose "Send to.") The "Send to" menu lets you right-click on an desktop icon or file(s) in Windows Explorer, then "send" the file to an application.

Windows XP allows "Send to" entries for individual users as well as a global "Send to" list that's given to all users. To add an entry to the menu, simply make a shortcut file in the proper *Send to* folder.

Figure Two shows a partly-customized "Send to" menu. The entry for *less* is a shortcut pointing to `C:\cygwin\bin\less.exe`; it's handy for viewing random files. (Look for details on *less* in this column within the next few months.) The *exiftags* and *identify* shortcuts point to little batch files and shell script files that run a utility, then pause until the user presses a key.

For example, here's the *sendto_identify.bat* Windows batch file. The `%1` passes a single parameter to the *identify* utility:

```
"C:\Program Files\ImageMagick-6.2.7-
Q8\identify" -verbose %1
pause
```

What do those "sent" parameters contain? Under what environment do these applications run? A shell script named *whatset.sh* shows you. Add a shortcut to `bash.exe whatset.sh` on a "Send to" menu. Then select one or more icons and "send them" to the script. *Figure Three* demonstrates sending two files to *whatset.sh*. *Listing Two* shows the *whatset.sh* script. It's just a series of commands, like *printenv* and *set*, wrapped in curly-braces so all of the command outputs can be piped to the same *less* process.

Once you've viewed all of the output, press `q` to quit *less* and close the" whatset" window.

Flexible File Searching

The *Windows XP* Search tool is integrated into Windows Explorer, and it has access to the *Indexing Service* for fast searches. However, the Search tool has a limited number of search parameters, and it's tough to use the list of files it finds unless you want to point and click on each match.

The Cygwin *find* (1) and *locate* (1) utilities are much more flexible. (Read more about *find* in the September 2002 "Power Tools" column "A Valuable find," available online at http://www.linux-mag.com/2002-09/power_01.html.) You can build the database for *locate* from the Cygwin *cron* facility.

An easy way to save the output of a utility for later re-use is by storing it in a shell variable. Here are examples for *bash* and *tcsh*:

```
$ files=`find /c ... -print`
% set files = `find /c ... -print`
```

If you want to pass the output of *find* (or *locate*) directly to

See Power, *pg. 54*

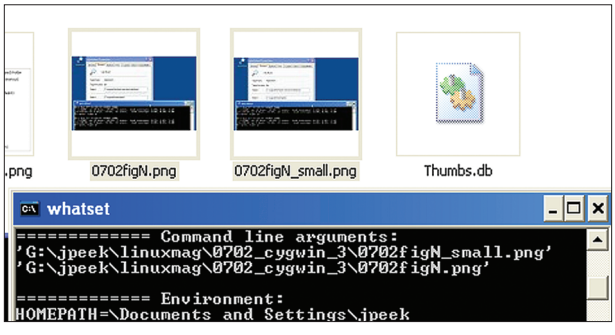

FIGURE THREE: "Sending" two files to *whatset.sh*

Power, *from pg. 17*

another utility, remember that spaces in Windows filenames can cause trouble. It's better to use `find -print0` and `xargs -0`. But shell variables generally can't store the NUL characters output by `find -print0`. In that case, use a temporary file instead:

```
find /c ... -print0 > /tmp/find.out
xargs -0 command < /tmp/find.out
```

When you copy or move lots of files with Windows Explorer, the long operation can quit in the middle due to some error, leaving you wondering what went wrong and how to recover from it

Searching Inside Files

The Windows Search tool can look for text strings in files, and the *findstr* utility has more sophisticated pattern matching. Both of those tools have limited choices for choosing the files you search. The Cygwin shells do a better job, with tools like *find* that give you lots of control:

```
$ find /c ... -print0 | xargs grep "..."
```

Cygwin's *egrep* utility lets you choose multiple (and very sophisticated) search patterns. You can also search all of the files in a directory with one *grep*, then pass the matched filenames to another (possibly different) *grep*:

```
$ grep -lZ "..." * | xargs -0 grep "..."
```

The *grep* option `-l` (lowercase "L") outputs only the matching filenames instead of the matching lines. The `-Z` (uppercase "Z") option uses NUL bytes to avoid problems with the spaces in many Windows filenames.

Copying and Moving Files

When you copy or move lots of files with Windows Explorer, the long operation can quit in the middle due to some error, leaving you wondering what went wrong and how to recover from it. The list of objects being moved or copied is shown one-by-one, so it can be tough to figure out what was done and what's left to do.

Cygwin's GNU *cp* and *mv* utilities have many more options, give you much more control, and have explicit error messages that make recovery easier. For instance, in one window you can run a command to do the copying, saving its verbose output in a log file:

```
$ cp -vrp * /c/bkup > /d/tmp/cp.log 2>&1
```

In another window you can watch the log file grow:

```
$ tail -f /d/tmp/cp.log
`aprog.exe' -> `/c/bkup/aprog.exe'
cp: failed to preserve ownership for
    `/c/bkup/aprog.exe': Permission denied
`bprog.exe' -> `/c/bkup/bprog.exe'
...
```

Cygwin brings Windows the level of tracking and control that you expect from a Linux system.

Excluding Files From ls

Last month's column mentioned the *bash* variable `GLOBIGNORE`. It lets you choose file types you don't want shell wild-

LISTING TWO: The *whatset.sh* script

```
#!/bin/sh
# whatset - show environment, command line arguments, etc.
# Usage: whatset

bin=/bin   # Executables in Cygwin space (Windows PATH may not
include it)

# Collect all outputs, pipe them to 'less' with a special prompt:
{

echo "Command line arguments:"
for arg
do echo "'$arg'"
done

echo "
Environment:"
$bin/printenv

echo "
Bash shell settings ('set' command): "
set

} | $bin/less -P'(less) Type f to go forward, b to go back, q to quit: '
```

cards to match. GNU *ls* has its own options to control which files are listed. For instance, the options `-B` and `--ignore-backups` won't list *Emacs* backup files. The options `-I` *pattern* (uppercase "I") and `--ignore=`*pattern* won't list entries matching *pattern*.

Here's a handy alias:

```
alias myls="ls -I '*.[oO][Bb][Jj]' -I
'*.[Ll][Oo][Gg]'"
```

The argument to `-I` is case-sensitive, so the square brackets allow extensions like `.obj` and `.OBJ`. Using the name *myls* instead of *ls* means you won't miss some files as you look around your system with *ls*. If the list from *ls* is cluttered, you can use *myls* instead.

Cygwin-Specific Utilities

Last month's column mentioned the *cygpath* utility that lists the locations of the Windows folders like the Desktop, translates Windows pathnames to POSIX/Cygwin equivalents, and more. Cygwin comes with other handy utilities like this. They're listed at http://cygwin.com/cygwin-ug-net/using-utils.html. One pair of utilities worth mentioning specially is *ps* and *kill*.

You can terminate a Windows window and all of the Cygwin processes running inside it, from the standard *Windows Task Manager* (which you open by typing Ctrl-Alt-Delete under Windows XP). Cygwin comes with a utility named *kill* that can terminate a single Cygwin process. There's also a utility named *ps* to list Cygwin processes. To end a Cygwin process, pass its PID number as an argument to *kill*.

Many Cygwin programs can be stopped (suspended) just as you would on a Linux system. Simply press Ctrl-Z while the process is the foreground (active) process. Get a list of jobs by typing `jobs`, and resume a job with `fg`. Stopping jobs lets you do a lot without opening multiple windows. For instance, you can keep several *man* pages available, just at the place you finished reading them before, and quickly pull up the one you want.

The Usual Linux Techniques

Of course, there's more to learn about Cygwin. If you look at a good introduction to Linux, you'll find that most of those techniques will work under Windows once you've installed Cygwin.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.

Advertisers' Index

The Advertisers' Index lists each company's Web address and advertisement page. To advertise in *Linux Magazine*, please contact adsales@linux-mag.com for a media kit containing an editorial schedule, rate card, and ad close dates.

Linux Magazine (ISSN 1536-4674) is published monthly by InfoStrada LLC at 330 Townsend St. Suite 112, San Francisco, CA 94107. The U.S. subscription rate is $29.95 for 12 issues. In Canada and Mexico, a one-year subscription is $59.95 US. In all other countries, the annual rate is $89.95 US. Non-US subscriptions must be pre-paid in US funds drawn on a US bank. Periodicals Postage Paid at San Francisco, CA and at additional mailing offices.

POSTMASTER: Send address changes to *Linux Magazine*, P.O. Box 55731, Boulder, CO 80323-5731.

Article submissions and letters should be e-mailed to editors@linux-mag.com. *Linux Magazine* reserves the right to edit all submissions and assumes no responsibility for unsolicited material. Subscription requests should be e-mailed to linuxmag@neodata.com or visit our Web site at www.linuxmagazine.com.