Cross-Platform Command Lines

By Jerry Peek

In an editorial a few months ago, Editor-in-Chief Martin Streicher pointed out that, whether we like Microsoft or not, it's a fact that many of us use Windows systems. Some of us use Macintosh computers with OS X. Each of those systems has a different graphical interface. Yes, you can install the X Window System on Macs and PCs, but wouldn't it be great to have the same standard interface to all of those systems, right out of the box?

One universal interface is already there! It's the *command line*, the time-tested (and time-improved) keyboard interface. If you're a long-time Windows user, you may not be familiar with all of the latest Windows command line features — or how similar it is to the familiar Linux (and Unix) shells.

If you've been reading this column for some time, you've seen a lot about command lines. (We'll cover another graphical tool soon!) But this month, let's look at even more tips that experienced users can appreciate: time-savers (and cross-platform-confusion-savers) that you can probably use on all of your systems, running Linux and otherwise.

When You Aren't Using Linux

Before we start crossing platforms, let's see how to get a command line prompt on Mac OS X and Windows.

In OS X, it's simple to get a command line: just open the *Terminal* application. By default, you'll get a prompt from *tcsh* or *zsh* (or you can choose other Unix-like shells). OS X has a lot of FreeBSD Unix inside. Once you've adapted to the differences between Linux/Unix filesystems and OS X's rather unique structure — and you've found command-line utilities like *CpMac* and *MvMac* from the Apple's Developer Tools CD — you should be ready to work from the command line.

In newer versions of Microsoft Windows, when you choose *Command Prompt* from the Start menu, you launch the Windows shell named *CMD*. As we'll see, *CMD* has a lot of the power of a Linux shell. In older systems like Windows 98, though, you'll have only the original DOS shell, *COMMAND*. It's much more limited than *CMD*, and only a few of the tips presented here will work. In this column, we focus on *CMD* under Windows XP Professional.

A great alternative to *CMD* — and, especially, to *COMMAND* — is one of the Unix-like shells you can get from platform-crossing packages. These packages come with some or all of the familiar utilities like *grep*, *sed*, and Perl. One package that'll make you feel right at home is the freely-available GNU/Cygwin software for Windows. You can download it for free from http:// www.gnu.org/doc/windows.html.

CONFIGURE YOUR WINDOWS SHELL WINDOW

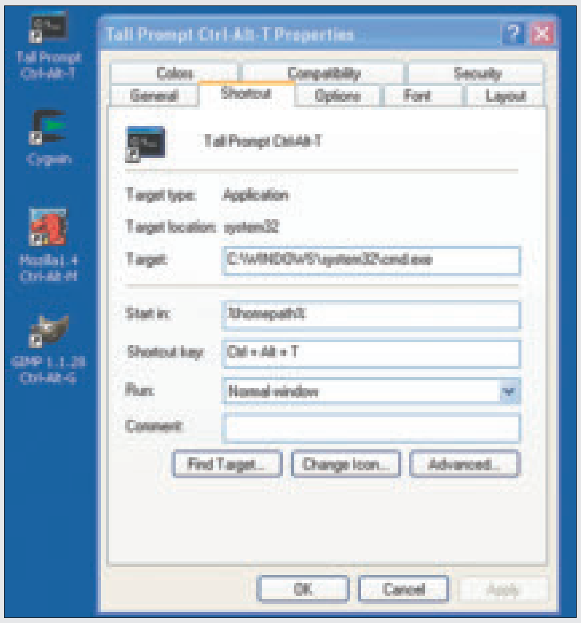
When you use a Linux shell, it typically starts in your home directory. You might also have it configured to use a taller window by default. If you aren't happy with the way your Windows *CMD* shell opens, you can change that.

Instead of making these changes to the program itself, consider making a shortcut and setting its properties. You can trigger this shortcut with a keystroke combination, like CTRL-ALT-S, to pop open a shell window without needing to find its icon and click on it.

Figure One shows icons for the *CMD* shell and a few cross-platform applications (the Cygwin *bash* shell, the Mozilla web browser, and the GIMP image editor), as well as the Properties window for the command prompt.

When you open the Properties box (right-click on the shortcut's icon), you can set the starting directory by typing `%homepath%` under Windows XP. (This month's Power Tip tells how to set the *homepath* environment variable to a shell-friendly directory pathname.)

Click in the Shortcut Key box and press the combination of keys you want to set as the startup keyboard shortcut. Check other tabs, like Layout and Font, for other default settings that you might want.



Tall Prompt Ctrl-Alt-T

Target type: Application
Target location: system32
Target: C:\WINDOWS\system32\cmd.exe
Start in: %homepath%
Shortcut key: Ctrl + Alt + T
Run: Normal window
Comment:

FIGURE ONE: Windows desktop: icons and shell window properties

Filename Completion

Most modern Unix/Linux shells provide *filename completion* to save keystrokes: type the first few characters of a file or directory name, and press the TAB key, and the shell fills in the rest of the name. This works great under OS X too, of course. The good news is that completion works under *CMD*, too — which makes those long Windows names a snap to type.

For instance, here's how you could copy the file *status_log.txt*. Type the copy command (*cp* or *copy* under Windows) and a space. Then type the first few characters of the filename you want to copy — and press TAB. (Under *csh*, press ESC; in *ksh*, press ESC twice.) If you've typed enough of the name to make it unambiguous, the shell finishes the name for you. (Otherwise, it may ring the bell or show you the first possible matching name. In this case, try pressing TAB again. Some shells need ESC followed by an equal sign (=). You should get a list of all possible matches.)

After the first argument is done, you can use completion on additional names. For instance, if you want to copy the file from the current directory into the directory *\Documents and Settings\jpeek*, you could next type the first few characters of the destination directory pathname, then press TAB (or, if needed, ESC). Here's an example of this, step by step, under Windows *CMD*.

```
C:\d\jpeek>copy stTAB

C:\d\jpeek>copy status_log.txt \DocTAB

C:\d\jpeek>copy status_log.txt "\Documents
and Settings"\jTAB

C:\d\jpeek>copy status_log.txt "\Documents
and Settings\jpeek"
```

(In this article, **TAB** indicates that you should press the TAB key, not type T-A-B.) The Windows shell saw that the destination pathname contained spaces, so it automatically added quotes around the name — and moved the quotes after the user typed more of the pathname. Unix-like shells may put a backslash (\) before each space.

Now you can press ENTER to run the command line. You've just copied a file in (probably) less time than you could scroll and click through lists of directories and files in a graphical file browser.

Shortening Pathnames with Wildcards, Part I

A *wildcard* lets you shorten one or more pathnames on a command line. Using wildcards, you type just some of the name and let the shell fill in the rest. (A *pathname* specifies the location of a file or directory, like */local/prj/afile* on Unix-type systems or *C:\windows\foo.exe* on Windows. There's more in the Power Tools article "What's in a Pathname?" available online at http://www.linux-mag.com/2002-10/power_01.html.)

Unix-like shells expand wildcards before running the command line — so the program you're running doesn't have to "understand" wildcards. The old DOS *COMMAND* shell had only limited support for wildcards, but *CMD* is almost Unix-like. We won't go into detail about wildcards here (for details, see http://www.linux-mag.com/2003-12/power_01.html, which will be available in March 2004).

LISTING ONE: Saving typing with wildcards

```
C:\d\jpeek>dir
 Volume in drive C is Programs
 Volume Serial Number is D195-4DB2

 Directory of C:\d\jpeek

12/05/2003  07:36 AM    <DIR>     .
12/05/2003  07:36 AM    <DIR>     ..
12/08/2003  09:51 AM    <DIR>     linuxmag
08/09/2002  12:30 PM    <DIR>     My eBooks
09/13/2003  02:05 PM    <DIR>     My Music
12/01/2003  08:51 AM    <DIR>     My Paintings
09/08/2003  09:35 AM    <DIR>     My Pictures
08/16/2003  09:08 AM    <DIR>     _gimp1.1
          0 File(s)              0 bytes
          8 Dir(s)  12,310,236,160 bytes free

C:\d\jpeek>cd *es

C:\d\jpeek\My Pictures>dir
 Volume in drive C is Programs
 Volume Serial Number is D195-4DB2

 Directory of C:\d\jpeek\My Pictures

09/07/2003  06:19 PM    <DIR>     .
09/07/2003  06:19 PM    <DIR>     ..
09/08/2003  03:34 AM    <DIR>
030906_1_lakeshore_boat_ride
09/08/2003  08:44 AM    <DIR>     030906_2_zoo
09/08/2003  09:37 AM    <DIR>     030906_3_mag-
nificent_mile
09/08/2003  09:39 AM    <DIR>
030907_1_Chicago_River_walk
09/07/2003  05:52 PM    <DIR>
030907_2_Loop_art_etc
          0 File(s)              0 bytes
          7 Dir(s)  12,310,236,160 bytes free

C:\d\jpeek\My Pictures\>cd *lake*

C:\d\jpeek\My
Pictures\030906_1_lakeshore_boat_ride>
```

Linux and OS X users know that a question mark (`?`) matches any single character in a file or directory name. The old COMMAND shell didn't work that way — but *CMD* does.

Let's see an example. Your directory has lots of files in it, some of whose names end with *.log*. You want to move (rename) those files into the directory *\Documents and Settings \Administrator\logs*. The shell can do the job, like this:

```
C:\d\jpeek>move *.log \DocTAB
```

Use completion to finish the directory pathname, and you'll have moved all of those files in 5 seconds or so. That's probably much faster than using a graphical file manager, where you'd need to search for those names by scrolling and clicking through a list in a GUI file browser — or change the browser to sort by extension, then select the first and last *.log* file.

Shortening Pathnames with Wildcards, Part II

Wildcards aren't only handy for copying and renaming. They also can help to select one name from many names — where the unique part of the name is somewhere in the middle. *Listing One* shows a directory listing. We want to get to the folder *My Pictures*. Filename completion would take a bit of work because we'd need to type `My Pic` to get an unambiguous name. It's a bit easier to use a wildcard to match the end of the name, `*es`.

Once we get to the *My Pictures* folder, we're faced with another maze of long names. Only one of the names has *lake* in it, though, so the wildcarded pattern `cd *lake*` takes us there in a flash.

Remembered Strings: Variables

Do you often type the same text on a command line or in a text-entry field of a graphical application? For instance, do you regularly access a particular directory (folder) buried deep in the filesystem?

You can store its pathname in a *variable* and make it available to each shell — and, in general, in other running programs, too. This is similar to making a "Favorites" entry in Windows Explorer or a bookmark in many Web browsers: the variable has a name and an associated value.

Unix-type shells support two kinds of variables: shell variables and environment variables. We'll cover environment variables here.

Let's say you have a folder with photos of your trip to Chicago stored in */home/zoe/photos/2003/chicago*. In Bourne-type shells like *bash* and *ksh*, you could type these two commands:

```
CHICAGO="/home/zoe/photos/2003/chicago"
export CHICAGO
```

POWER TIP: Make XP Paths more Linux-like

Linux and Unix users often put personal files in and under their home directory, a place that's easy to reach from the shell. On Windows, though, users have a folder named *My Documents* which typically isn't near the filesystem root. Instead, it's buried somewhere like *C:\Documents and Settings\jpeek\My Documents*. If you're installing a new user (where no applications have locked-in the path to this folder), it can make sense to set both *My Documents* and the home directory (`%HOMEPATH%`) to a shorter pathname.

Under Windows XP, a user can change the location of *My Documents* by right-clicking its icon in Windows Explorer, choosing Properties, and clicking the Target tab. Then in "Target Folder Location," in the "Target" box, type a short pathname like *C:\u\zoe*.

An administrator can change the location of a user's home folder by going to the Control Panel, choosing Administrative Tools, opening Computer Management, expanding Local Users and Groups, clicking Users, right-clicking the user's name, and choosing Properties. On the Profile tab, under Home Folder, in the Local Path box, type a pathname like *C:\u\zoe*.

Most Bourne-type shells also accept an abbreviated sequence like this:

```
export CHICAGO="/home/zoe/photos/2003/chicago"
```

And, because your home directory's pathname is also available in the environment variable named *HOME*, you can even shorten that a bit more to:

```
export CHICAGO="$HOME/photos/2003/chicago"
```

C-type shells like *tcsh* have a different syntax. Here are two ways to set `CHICAGO`, with and without `HOME`. (You can also use ~ in place of `$HOME`.)

```
setenv CHICAGO
"/home/zoe/photos/2003/chicago"
setenv CHICAGO "$HOME/photos/2003/chicago"
```

You can type those commands at any shell prompt. They'll take effect in that shell session and in any programs started from it. To make *CHICAGO* available in every shell, put the command in the shell's setup file, like *.bashrc*, in your home directory. You also may want to set it in a setup file that's read before your window manager starts. That will make the variable available to all shells and all programs run from your window system.

In Microsoft Windows, the setup and syntax are different

See Power, *pg. 66*

Power, *from pg. 38*

but the overall idea is the same. You can set a shell variable within one *CMD* shell and it will apply to that session. If you want to use the variable every time you log on, store it in a system setup area.

The command line is the time-tested and time-improved universal interface for Linux, Mac OS X, and Windows

In Windows XP, for instance, on Control Panel, choose System, click the Advanced tab, and click the Environment Variables button. Under early versions of Windows, you can set system-wide variables in the *C:\AUTOEXEC.BAT* file.

The syntax for *CMD* looks like one of these:

```
set
CHICAGO="C:\d\jpeek\photos\2003\Chicago"
set
CHICAGO="C:%HOMEPATH%\photos\2003\Chicago"
```

To use your `CHICAGO` variable on Unix-like shells, put `$` before its name, like:

```
% cp $CHICAGO/*lake*/*.jpg /tmp
% cd $CHICAGO
```

From *CMD* or from some Windows dialog boxes (like the Properties box shown in *Figure One*, surround the variable's name with `%`:

```
C:\>cd %HOMEPATH%
C:\d\jpeek>copy
  %CHICAGO%\030905_adams\*.jpg
```

(Variable names are case-sensitive on Unix-like shells, but not on Windows, so we could have used `%homepath%` and `%Chicago%` above.) To see a list of variables and their values, type *printenv* or *env* on Unix-like systems, or *set* on Windows.

To Find Out More...

To read more about shell features, check any good book about Linux or Unix — or read this magazine! For details about *CMD*, see http://www.microsoft.com/technet/prodtechnol/ winxppro/proddocs/cmd.asp.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for over 20 years. He's happy to hear from readers at jpeek@jpeek.com.

Guru, *from pg. 42*

After typing this command, you'll be asked for a password. Type the password on the server associated with the account you're using (which need not be the same as the account whose password you're changing).

Another common example of *net* in use is in joining a domain. You might do this if you've set up a Samba server that is not a domain controller, but when you want the system to be part of a domain. The command to do the job might look like this:

```
$net -s penguin -U adminuser JOIN
```

This command joins the local computer to the domain controlled by the `penguin` computer, using the `adminuser` administrative account. To work with a Samba domain controller, you will need to have created an appropriate domain trust account for the machine you're adding, or at least have configured the `add machine script` option in *smb.conf* to do the job automatically. After issuing this command, you should be able to set `security = domain` and associated options in *smb.conf* to have the server defer to the domain controller for authentication tasks.

Yet another use of *net* is modifying the group mapping database. This database is used to map Unix groups to SMB/CIFS groups. For instance, suppose you want to create a group for summer interns on your domain. You can do so by creating an appropriate Linux group (say, `interns`) and then running a command like the following:

```
$net -s penguin GROUPMAP ADD \
  ntgroup="Summer Interns" unixgroup=interns
```

This command sets up a domain group called `Summer Interns` and associates it with the Linux group `interns`. Windows users who are added to this group will create files with `interns` group ownership on the Linux system, and access files with `interns` group permissions.

Overall, the *net* command is a powerful tool for server and domain management. It's responsible for some of Samba 3.0's improved domain integration features, and net will undoubtedly become more powerful and important in the future.

Learning to use this command will serve you well in managing Samba servers and clients.

Roderick W. Smith (rodsmith@rodsbooks.com) is the author or co-author of twelve books, including Advanced Linux Networking *and the upcoming* Definitive Guide to Samba 3.