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

(Chrono)logical Sorting

Sort file names chronologically and move up the directory hierarchy

By Becca Thomas

The way Unix sorts files can sometimes be a frustrating thing. You may wonder why it can’t recognize that files named for months should be arranged in chronological order rather than alphabetical order. The file whose name begins with ‘‘dec’’ should follow the file named ‘‘nov,’’ not precede it, but you have to tell Unix how to do it.

Robert Nicholson provides a Perl utility that is useful for sorting file names chronologically instead of the default alphabetical sorting order used by Unix utilities. The approach provides a template for sorting other file-name formats. David Wood submits a small shell function that provides a command-line shorthand for ascending the directory hierarchy.

By the Dates

Dear Dr. Thomas:

I recently wanted to scan some past Wizard’s Grabbag listings, so I set up the `mirror` Perl script (see Wizard’s Grabbag in the November 1994 *UnixWorld Hands-on*), to download all the listings from UUNET. However, when I began to look at the files, I noticed that you use an alphabetical naming scheme, which doesn’t lend itself to viewing files in order of publication because the shell returns its argument list sorted alphabetically, not chronologically.

I couldn’t rely on the files’ modification time and just use `ls -rt` because this attribute isn’t always consistent with the file name. As an example, if I had the files named INDEX, apr90, jan89, dec90, apr89, jan90, and dec89, I couldn’t run `more *` to view the files in chronological order (jan89, apr89, dec89, jan90, apr90, and dec90).

The key is to sort the list by date rather than alphabet. I thought about the language features needed to solve this problem. The first thing I considered was the C language `qsort(3)` routine, which allows you to specify an arbitrary comparator function to define how sorting takes place. However, I had recently done some Perl programming and knew that Perl would let me do the same thing. Furthermore, Perl provides not only all the necessary language features-support for associative arrays, string manipulation, and a flexible sorting scheme-but also offers instant cross-platform portability so I could share my tool with others.

The result is the `dateorder` Perl program [Listing 1A], which reads files named in a month-year (*mmmyy*) format on its command line, and returns the names in year, month order. [Part B shows sample usage.]

Robert Nicholson / Swiss Bank Corp. / London

Explanation of `dateorder` *operation*. Line 11 displays the correct command line to use and terminates the script if no invocation arguments are supplied. Lines 13-15 define an array that associates the three-character month names with their ordinal calendar position.

Lines 18-22 save the command-line arguments that begin with a *mmmyy* prefix (such as jun92 or aug94). In particular, the month_and_year_prefix() function is called once for each command-line argument (value in `$file`). If this function returns a one for true, it stores the file-name argument in the `monthfiles[]` array, which is indexed by an integer that is incremented after each value is stored.

Line 24 calls a sort routine that orders files names stored in the `monthfiles[]` array using the `by_date()` comparison routine.

Two arguments are passed to the comparator function (lines 28-44), which returns -1, 0, or 1 depending on whether the first value is less, equal to, or greater than the second. Perl passes these values by reference and their pointers are swapped accordingly. The function works by associating any month's three-character prefix with its corresponding integer value. This value is then catenated with the two-digit year part of the file name, and a numerical comparison is made to determine whether the first value is greater than, less than, or equal to the second.

To support *UnixWorld's* current naming scheme, a significant limitation was introduced: The program assumes a two-digit year and will not function when we hit the year 2000. However, it would be a trivial modification to support four-digit year values.

The month's integer value is adjusted to ensure that we always have a two-digit month value. Consider what happens when we compare the dates dec91 and jan94. If we didn't adjust the value, these would translate into 9112 and 941, respectively. Then, when the values are compared, the first date (dec91) would have been regarded as more recent than the second (jan94). Adding 11 to each month's integer value ensures that we are always comparing four-digit values.

Now let's look at the `by_date()` function more closely. The two values to be compared are passed to this function in the `$a` and `$b` global variables. Line 30 extracts the month substring prefix from `$a` and line 31 does the same for `$b`. Then lines 33-34 extract the year substring from these same variables.

Lines 37-38 convert the month substring to its equivalent ordinal value (for example, jan = 1 and dec = 12). Here the `\L` operator converts any capital letters in the user -supplied month substring to lower case so the result can be used as the "key" in the `%monthorder` associative array. The ordinal value is obtained from the array.

Lines 41-42 create the integer values used for the comparison by line 43. Eleven is added to each month index value so it will always be a two-digit value. The two-digit year value is then catenated to the adjusted month value in order to yield the desired four-digit number.

What's that funny looking operator (<=>) used in line 43? Well, the Perl authors affectionately refer to this item as the ‘‘spaceship’’ operator. It's actually a shorthand for a three-way comparison: If the first value (`$aall` here) is less than the second (`$ball`), the ‘‘spaceship’’ operator returns -1; if they are equal, it returns 0; but if the first is greater than the second, 1 is returned by Perl's sort operator on line

24.

What about the `month_and_year_prefix()` function defined on lines 47-61? Line 48 stores the argument passed to the function in the local variable `$filename`. Line 50 checks if the file name is long enough, returning false (zero) if not. Lines 52 and 53 extract the month and year prefix values storing them in similarly named variables. Lines 55-59 have the function return true (one) if certain conditions are met: the name refers to an ordinary file, the month prefix occurs in the `%monthorder` array defined earlier, and a two-digit value-presumably representing the year-was used. If these conditions aren't met, the function returns false (zero).

Tester's Comments

My two-line hack does something similar [see Listing 1C]. Of course, it's not as portable as a Perl program because the `ls` output format may differ from system to system and not all sort commands understand the month order (`-M`) option. My alternate command line [Listing 1D] simulates `dateorder` even more closely.-- Endre Bálint Nagy, Hunix Ltd., Budapest, Hungary.

Up the Hierarchy

Dear Dr. Thomas:

I've enclosed a shell function named up that I find handy to move one or more directories toward the root directory. This simple function takes one optional argument, which specifies how many levels to ascend. Thus, instead of entering **`cd ../../..`** to go up three levels, you simply enter **`up 3`**. Several of my coworkers initially thought this function useless. How hard is it to type **`cd ../..`** anyway? But now they are all hooked on it.

This Bourne-shell-compatible function was developed using Korn shell version 88 and tested using the Bourne shell running under System V Release 3.2.3 on a 3B2/1000.

David Wood / Cliffwood, N.J.

Portability Note: Shell functions are available starting with the Bourne shell that was shipped with System V Release 2 and all versions of the Korn shell.

Caution: If you've defined a Korn shell alias using the same name as our function, you'll get a syntax error when you attempt to define the function. In this case you'll need to either rename the alias or the function.

Notes: Part A of Listing 2 shows the `up()` function definition and Part B shows a version ‘‘ported’’ to the Korn shell. The latter also checks for a valid ‘‘level- count’’ argument.

Tester's Comments

The Bourne shell version works fine. I couldn't test the Korn shell version (might be time to install `pdksh` on my FreeBSD systems).-Endre Bálint Nagy, Hunix Ltd., Budapest, Hungary.

Corrections

Sorry, we goofed. Last month we published two domain names with typos. The domain name `cse.inl.edu` should be `cse.unl.edu` and `frc.doc.ic.ac.uk` should be `src.doc.ic.ac.uk`.

Sorry for any confusion.

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

Go To Content Search

Last Modified: Tuesday, 22-Aug-95 15:46:25 PDT