COBB

September 1997 • $11.50
VOL. 3 NO. 9

INSIDE SOLARIS™

Tips & techniques for users of SunSoft Solaris

in this issue

Filename completion in the C and Korn shells

By Al Alexander

I know few programmers and administrators who enjoy typing. Frankly, many of us aren't good typists, myself included. In fact, I avoid typing like the plague. Whenever I can find a shortcut that allows me to type less, I take it—especially in regard to long Solaris filenames and paths. In this article, we're going to discuss some shell filename-completion techniques to cut down on your keystrokes and reduce your typing errors.

The phrase *filename completion* refers to the process of using command-line shortcuts that let the shell do the dirty work of determining the filenames you're referring to when you type. In essence, it's a process that lets you type only a few characters to enter a 20-character filename. If you're not taking advantage of filename-completion capabilities by using wildcard characters and a few other techniques, you might be missing some of the features that make Solaris more powerful (and more fun) than other command-line environments. Let's look at a brief example:

Suppose you're working in a directory that contains these files:

```
$ ls
lone_ranger                 lone_wolf
lone_star_cafe              long_filename
alone_at_last               long_gone
long_long_long_filename
```

Suppose you wanted to use `vi` to edit the file *long_long_long_filename*. You could, of course, type *vi long_long_long_filename*, but that's a waste of keystrokes and prone to errors besides.

In the C shell

In the C shell, you can set a shell variable, `filec`, that can make your typing life easier when it comes to long filenames. Setting the shell variable is easy. Simply type

```
Devo% set filec
```

at your shell prompt. What does this do? Let's look at a few sample cases.

If you're using the C shell, and you want to edit the same file as in our previous example, you can type the following:

```
Devo% vi long_l*
```

This takes advantage of the power of the * filename metacharacter. In essence, the shell looks in the current directory and finds the list of files that begin with the letters *long_l*, then substitutes this file list in place of the *long_l** you actually typed. This is nothing new to you. As you're aware, if you type one of the characters incorrectly, you'll either get an entire list of files to edit, or you'll get the wrong file(s) to edit.

INSIDE SOLARIS™
Tips & techniques for users of SunSoft Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by The Cobb Group.

Prices
U.S. .... $115/yr ($11.50 each)
Outside U.S. .... $135/yr ($16.95 each)

Phone and Fax
US toll free .... (800) 223-8720
Local .... (502) 493-3300
Customer Relations fax .... (502) 491-8050
Editorial Department fax .... (502) 491-4200
Editor-in-Chief .... (502) 493-3204

Address
Send your tips, special requests, and other correspondence to:

The Editor, *Inside Solaris*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: inside_solaris@zd.com.

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to:

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
Internet: cobb_customer_relations@zd.com

Staff
Editor-in-Chief .... Marco C. Mason
Contributing Editors .... Al Alexander
W. Dean Stanton
Production Artists .... Marguerite Winburn
Natalie Strange
Editor .... Karen S. Shields
Publications Coordinator .... Linda Recktenwald
Editor-in-Chief of Online Publishing .... Darren McGee
Circulation Manager .... Mike Schroeder
Editorial Director .... Linda Baughman
Managing Author .... Eddie Tolle
Publisher .... Mark Crane
President .... John A. Jenkins

Back Issues
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $11.50 each, $16.95 outside the US. We accept MasterCard, Visa, or American Express, or we can bill you.

Postmaster
Periodicals postage paid in Louisville, KY and additional mailing offices.
Postmaster: Send address changes to:

Inside Solaris
P.O. Box 35160
Louisville, KY 40232

However, once you've set the `filec` variable, you can take a different approach to this problem. Now, after typing *vi long_l*, you can hit the [Esc] key. You'll see the entire filename "magically" appear on your command line, like this:

```
Devo% vi long_l[Esc]ong_long_filename
```

When you pressed the [Esc] key, the C shell performed this magic filename completion for you. This technique is similar to using the * filename metacharacter, but it has several advantages. It shows you the filename you're about to work with on the command line. If it's the right filename, you can press the [Enter] key to execute the command. If it's the wrong file, you can erase the characters and retry. If no file matches, you can edit the characters and press the [Esc] key to try again.

You can use the same technique to match filenames in any directory on the system. For example, you can edit the */etc/passwd* file by using the command

```
Devo% vi /etc/pas[Esc]swd
```

This filename-completion technique also works with directory names, since directories are simply a special form of file. Commands, too, are simply files with the execute permission turned on, so you can also use this technique with command names. For example, when I log into my computer and use the C shell, I can type */u[Esc]/op[Esc]/b[Esc]/ op[Esc]* to start the OpenWindows GUI environment. As I enter this keystroke sequence, the following command appears on the command line:

```
Devo% /usr/openwin/bin/openwin
```

As our convention is to show what you type in blue, you can see that you can type significantly fewer characters to get the same result. Because you immediately see the results on your screen, this technique can help avoid the mistakes when you use the * metacharacter and it doesn't match the files you expected. This isn't too important in our example when you type

```
Devo% vi lon*
```

but it can be very important when you type a potentially destructive command, such as

```
Devo% rm lon*
```

This, of course, would delete every file in the current directory whose name begins with the letters *lon*. If this is what you wanted, that's great. But if you just wanted to delete *long_long_long_filename*, then you'll have to break out your backup tapes.

Filename completion offers another great advantage. Assume for a moment that just as you typed

```
Devo% rm lo
```

you find that you forgot the name of the file you wanted to delete. To recover from this problem, you could cancel your current command by pressing [Ctrl]C, then type *ls lo** to show the appropriate filenames, then retype the `rm` command with your memory suitably refreshed.

However, filename completion offers you a simpler method. When you're at that same point in your command line, and you've typed *rm lo* but can't remember the rest of the filename to delete, just press [Ctrl]D to make the C shell display the list of files that currently match *lo*. For example, suppose you enter the following command line:

```
Devo% rm lo
```

Now you're wondering exactly which file(s) you want to delete. At this point, just press [Ctrl]D, and the C shell will display all matching files, then re-enter the partial command for you, like this:

```
Devo% rm lo[Ctrl]D
lone_ranger            lone_wolf
lone_star_cafe         long_filename
long_gone
long_long_long_filename
Devo% rm lo
```

This [Ctrl]D capability is perhaps even more useful than the [Esc] capability because it can be used any time you're uncertain about the complete name of a file (or group of files). Because it only displays the names of the files your current command matches, it's also a "safe" command to issue.

To further demonstrate these advantages, a typical command sequence involving both uses of the filename-completion capability might look something like this: You just decided that you wanted to delete the filename *long_long_long_filename*, so you start by typing

```
Devo% rm lon
```

Then, forgetting that there are other files in this directory that match the *lon** pattern, you press [Esc] here to "magically" complete the filename on the command line:

```
Devo% rm lon[Esc]
```

At this point, however, nothing happens at the command line (other than a quick flicker on the screen) because the C shell can't find a unique match. But now you don't despair, as you would if you had used the * metacharacter. Instead, you press [Ctrl]D to get a list of all the files that *do* match. Then you can enter a few more characters to make the match unique and press [Esc] again, allowing you to safely delete only the file you want.

The Korn shell

The Korn shell also has filename expansion, but you get to it in a slightly different fashion. Rather than using the `filec` shell variable, you use the `EDITOR` shell variable, setting it to `emacs`, `gmacs`, or `vi`, like this:

```
$ EDITOR=emacs
```

Now, to perform command-line completion, you can press [Esc]* to complete filenames. You must press two characters, rather than only one as used in the C shell, because the `EDITOR` variable enables so much more than filename completion. You can recall previously edited commands, edit your current command line, and many other operations. (We introduced this topic in the article "Using Command Line History in the Korn Shell" in our November '96 issue.)

When the pattern you type matches more than one filename, the response of the Korn shell to your command is very different than the response of the C shell. Where the C shell offered no response when we typed *rm lo[Esc]*, the Korn shell responds in an opposite manner by listing every file on your command line that begins with the *lo** pattern. Typing the command

```
$ rm lo[Esc]*
```

leads to this result:

```
#  rm lone_ranger lone_star_cafe lone_wolf
long_filename long_gone long_long_long_filename
```

The Korn shell response to the [Esc]* key sequence lets you see every file affected by your wildcard pattern, whereas the C shell quietly lets you know that more than one file will be affected by your command. Depending on your work style, you'll probably appreciate one of these approaches, but not both.

It turns out that `emacs` and `vi` mode offer (different) alternative forms of filename expansion. In `vi` mode, using [Esc]\ causes the Korn shell to add the longest common prefix for all matching files. So if you type *l[Esc]*, the Korn shell will expand it to *lon*, since all of the matching filenames (those beginning with the letter l) begin with lon. This way, rather than getting a complete list of all the files, you can get a single filename, giving the shell direction (in the form of a character or two to differentiate files) here and there.

If you happen to set the `EDITOR` variable to the value `emacs`, you'll find that it has a similar mode, but you use the much handier [Esc][Esc] rather than [Esc]\. I often use `emacs` mode because I find that it offers more flexibility. However, if you tend to use the `vi` editor, you'll find `vi` mode more intuitive. As always, you'll select the mode you prefer.

Another great feature of `emacs` mode is that it has a feature similar to the [Ctrl]D sequence found in the C shell. When you're uncertain about the filename(s) you're about to match, `emacs` mode allows you to type *[Esc]=* to produce a list of matching files. (Please note that `vi` mode has no corresponding feature.) So if we type

```
$ rm lon[Esc]=
```

the Korn shell produces this output:

```
1) lone_ranger
2) lone_star_cafe
3) lone_wolf
4) long_filename
5) long_gone
6) long_long_long_filename
```

This is a great feature because you can see that six files will be removed if you type nothing more than *rm lon** or *rm lon[Esc]**. I prefer this output format over that of the C shell. As with the C shell, it also re-enters the partial command for you, so you can continue. However, if you want to add to the end of your command line, you must use the add-to-end-of-line command, `A`.

In the real world, I use the `[Esc]=` command much more than the `[Esc]*` pattern because I like to verify that the files I'm about to operate on are the files I intended to operate on and no more.

Summary

We've presented a quick overview of the filename-completion operations found in the C and Korn shells. These options can save time *and* reduce the number of errors you make when typing in commands. For further information on this topic, you should definitely read the `man` pages on the C shell and Korn shell. (Especially the Korn shell—Read the sections on `emacs` and `vi` command-line editing modes for more great features that will save you time and effort.) ❖

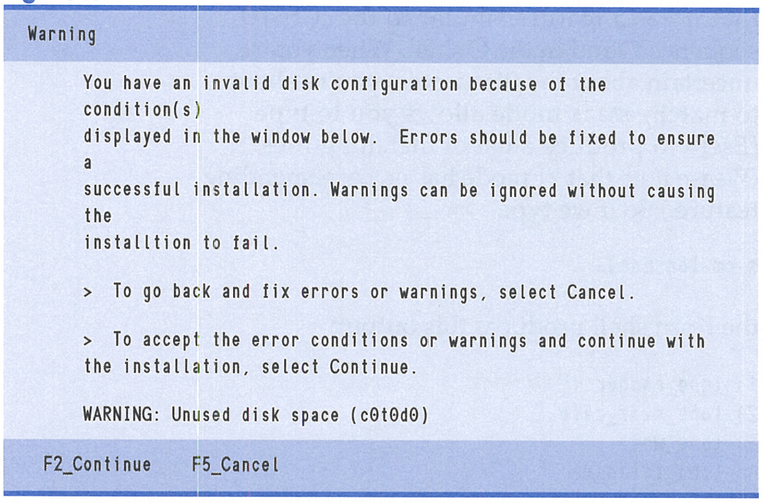
Alvin J. Alexander is an independent consultant specializing in UNIX and the Internet. He has worked on UNIX networks to support the Space Shuttle, international clients, and various Internet service providers. He has provided UNIX and Internet training to over 400 clients in the last three years.

Slicing your hard disk during installation

When you're installing Solaris on a new computer, you must choose whether to let the installation program select the sizes of your disk slices or to do it yourself. If you let the installation program automatically make the choices for you, you may find the choices to be inadequate for your application. However, some users experience a minor problem when they try to customize the hard-disk configuration. Some time after setting the disk configuration, the installation program complains with the error message shown in Figure A.

If you've encountered this warning, you probably did one of two things: You either started over and let the installation program make all the choices, or you continued forward. If you continued forward, you probably had this nagging doubt eating at you: Didn't you allocate all the disk space during the layout procedure? In this article, we'll show you how to customize your disk configuration when you install Solaris, as well as why this warning occurs and how to avoid it.

Figure A



```
Warning

      You have an invalid disk configuration because of the
      condition(s)
      displayed in the window below.  Errors should be fixed to ensure
      a
      successful installation. Warnings can be ignored without causing
      the
      installtion to fail.

      >  To go back and fix errors or warnings, select Cancel.

      >  To accept the error conditions or warnings and continue with
      the installation, select Continue.

      WARNING: Unused disk space (c0t0d0)

  F2_Continue    F5_Cancel
```

During some installations, if you customize the disk layout, your reward will be this annoying warning message.

Choosing auto-layout or custom layout

The starting point happens when it's time to decide whether to let the installation program configure your hard disk or do it yourself. During the installation procedure, you're presented with the choice in the screen shown in Figure B.

If you select the auto-layout feature, Solaris allows you to specify which file systems you want, then create a small root (/) slice, a small */usr* slice, and other slices as you specify, and use the remainder of the disk for */export/home*. However, the */usr* slice is always far too small for my needs—I like to install CDE at the default location of */usr/dt* and lots of GNU software at */usr/local*. Therefore, I always select Custom layout to ensure that the */usr* slice is large enough. If you were to press [F4] at this point, it would take you to the screen shown in Figure C.

If you press [F4] at this point, you must manually create the file system layout. If you're comfortable with that, go ahead and do so. When you're finished, skip to the section *Customizing Your Configuration*. If you're not comfortable manually creating all the file systems, go ahead and let the installation program do the auto-layout: Press [F2] to let the installation procedure take you to the auto-layout screen shown in Figure D.

On this screen, you can select from a list some file systems that the program will automatically create. After the installation program does the auto-layout procedure, you'll have another chance to customize your disk layout, as we'll see shortly.

Some people like to have just the root (/) slice and swap (particularly for individual workstations), while others like to manage more slices so that each will be tuned for different operations. You may want to refer to "No Matter How You Slice It..." in the September '96 issue to help you decide how to slice your disk and "An Introduction to File System Tuning" in last month's issue for methods to tune file systems on individual slices.

For our example, we'll select the defaults and press [F2]. The installation program will then create the slices you specified and one other: */export/home* to hold the home directories for your users, as shown in Figure E on page 6.

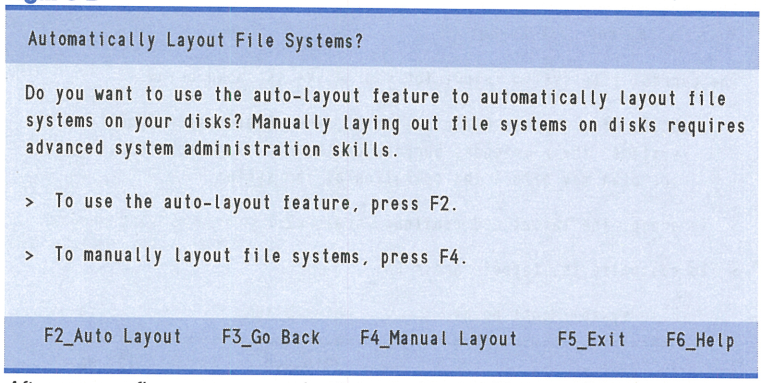
Customizing your configuration

Whether or not you elected to use the auto-layout feature, you should see the screen shown in Figure E. If you used the auto-layout, then you'll see the installation program's recommended configuration, as we've done. If you skipped the auto-layout feature, then you'd only see the overlap file system.

If you used the auto-layout feature and you're happy with the configuration that it offers, just press [F2] to accept the layout and continue. However, we're not going to do that. We're going to increase the */usr* slice (at the expense of the */export/home* slice). So we'll press [F4] now to get to the Customize Disk screen, shown in Figure F on page 6.

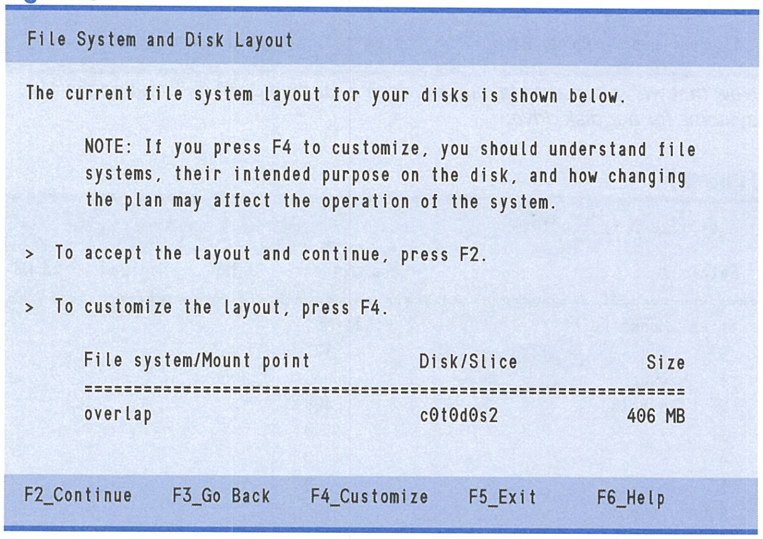
Using the arrow keys on this screen, you can move to the fields you want and edit the

Figure B


```
Automatically Layout File Systems?

Do you want to use the auto-layout feature to automatically layout file
systems on your disks? Manually laying out file systems on disks requires
advanced system administration skills.

> To use the auto-layout feature, press F2.

> To manually layout file systems, press F4.

F2_Auto Layout   F3_Go Back   F4_Manual Layout   F5_Exit   F6_Help
```

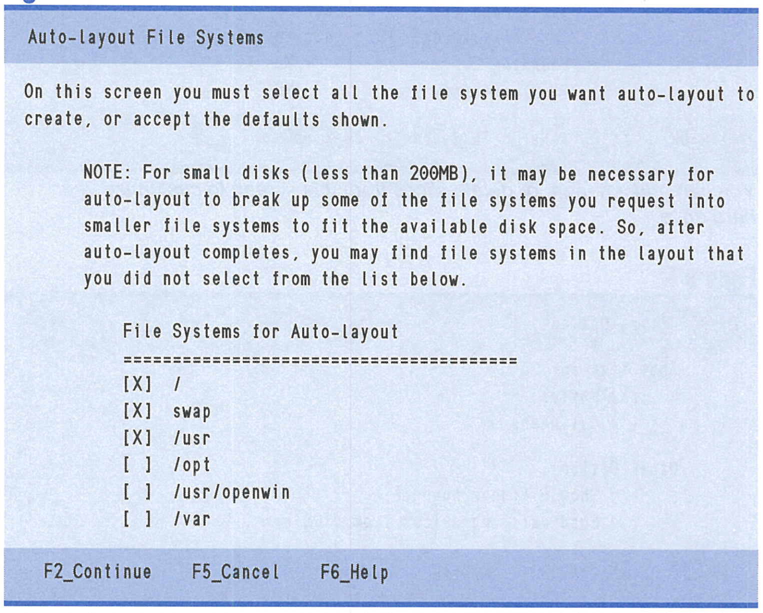
After you configure your console, time, and networking, the Solaris installation program wants to configure your hard disk for you.

Figure C


```
File System and Disk Layout

The current file system layout for your disks is shown below.

      NOTE: If you press F4 to customize, you should understand file
      systems, their intended purpose on the disk, and how changing
      the plan may affect the operation of the system.

> To accept the layout and continue, press F2.

> To customize the layout, press F4.

      File system/Mount point           Disk/Slice               Size
      ============================================================
      overlap                           c0t0d0s2               406 MB

F2_Continue   F3_Go Back   F4_Customize   F5_Exit   F6_Help
```

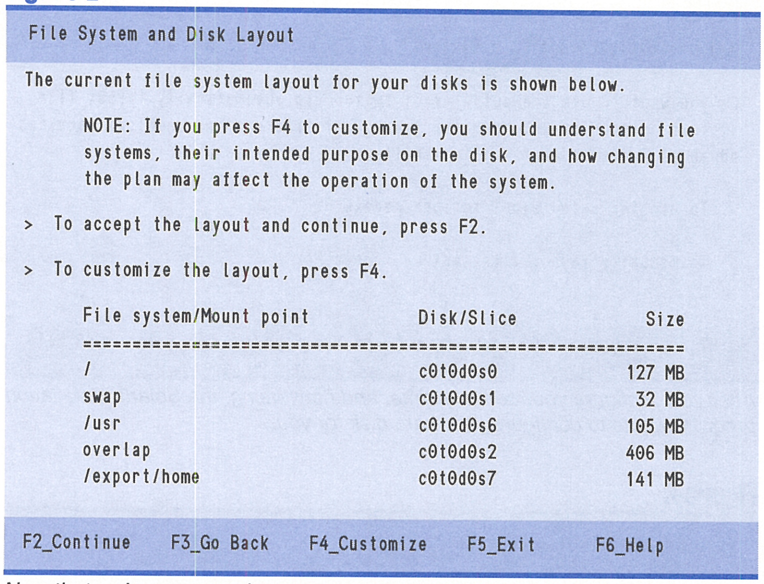
This screen shows you the current configuration of your hard disk and gives you another chance to change your mind.

Figure D


```
Auto-layout File Systems

On this screen you must select all the file system you want auto-layout to
create, or accept the defaults shown.

      NOTE: For small disks (less than 200MB), it may be necessary for
      auto-layout to break up some of the file systems you request into
      smaller file systems to fit the available disk space. So, after
      auto-layout completes, you may find file systems in the layout that
      you did not select from the list below.

          File Systems for Auto-layout
          ========================================
          [X]  /
          [X]  swap
          [X]  /usr
          [ ]  /opt
          [ ]  /usr/openwin
          [ ]  /var

F2_Continue   F5_Cancel   F6_Help
```

The installation program allows you to select file systems that it will automatically create for you.

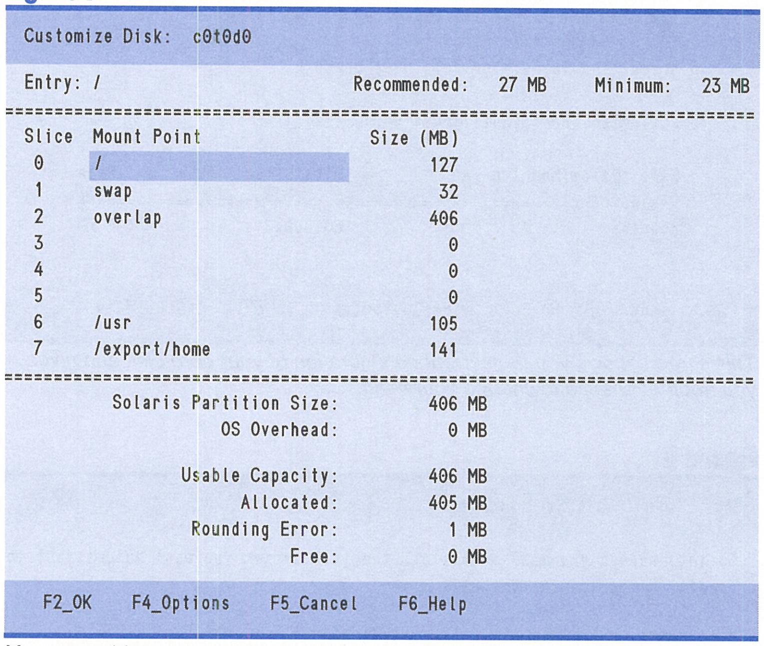
Figure E



```
File System and Disk Layout

The current file system layout for your disks is shown below.

     NOTE: If you press F4 to customize, you should understand file
     systems, their intended purpose on the disk, and how changing
     the plan may affect the operation of the system.

>  To accept the layout and continue, press F2.

>  To customize the layout, press F4.

     File system/Mount point            Disk/Slice             Size
     ============================================================
     /                                  c0t0d0s0             127 MB
     swap                               c0t0d0s1              32 MB
     /usr                               c0t0d0s6             105 MB
     overlap                            c0t0d0s2             406 MB
     /export/home                       c0t0d0s7             141 MB

F2_Continue    F3_Go Back    F4_Customize    F5_Exit    F6_Help
```

Now that we've run auto-layout, we have a starting point for creating the file systems for our disk drive.

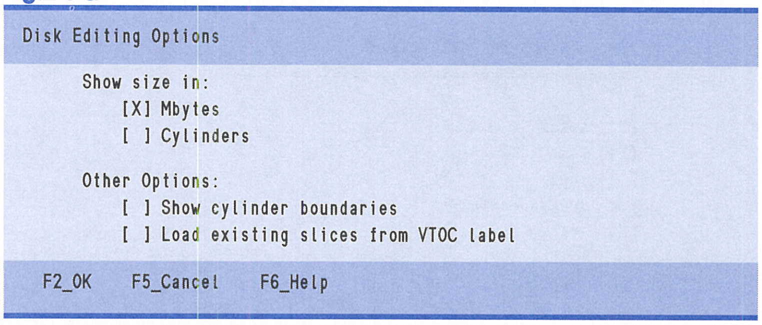
Figure F



```
Customize Disk:  c0t0d0

Entry: /                          Recommended:   27 MB    Minimum:   23 MB
==========================================================================
Slice  Mount Point                   Size (MB)
  0    /                                127
  1    swap                              32
  2    overlap                          406
  3                                       0
  4                                       0
  5                                       0
  6    /usr                             105
  7    /export/home                     141
==========================================================================
         Solaris Partition Size:        406 MB
                    OS Overhead:          0 MB

                Usable Capacity:        406 MB
                      Allocated:        405 MB
                 Rounding Error:          1 MB
                           Free:          0 MB

  F2_OK    F4_Options    F5_Cancel    F6_Help
```

You can add, change, or delete slices from this screen to customize your hard drive.

Figure G



```
Disk Editing Options

       Show size in:
           [X] Mbytes
           [ ] Cylinders

       Other Options:
           [ ] Show cylinder boundaries
           [ ] Load existing slices from VTOC label

  F2_OK    F5_Cancel    F6_Help
```

If you display the size in cylinders and show cylinder boundaries, you can see whether you're wasting any space.

text (for the Mount Point) or number (for the Size field) to get the results you want.

If you skipped the auto-layout feature, you must create your file-system mount points and select the sizes for your file systems from scratch. This is actually a very simple process; the hard part is knowing exactly what you want.

Note two important things about this screen. First, don't change the overlap slice. This slice is used by Solaris to keep track of the hard disk drive. You won't actually mount the overlap file system under normal use. Second, you must decrease the size of one slice before you can increase the size of another, or the installation program will complain when you try to use more space than the disk provides.

Also, don't leave any empty space on the disk (i.e., ensure that the Free: field says 0), or the installation program will complain with the screen we showed you earlier in Figure A. Unfortunately, that screen only shows up several screens later, when you're unsure as to just what it was you typed. You can go ahead and continue at this point, but if you do, you'll be wasting some disk space. It may not be much, but why waste *any* disk space on your system?

Please note that even if you're careful and allocate every bit of space so that the Free: field in Figure F is 0, you can get this error. This happens because you're working with MB when you allocate sizes for your partitions, while the installation program is actually working with cylinders. So the installation program approximates how many cylinders are in a megabyte. For some hard disk drives, the division won't come out even, so when you customize the hard disk, a few cylinders may remain.

Use every crumb...

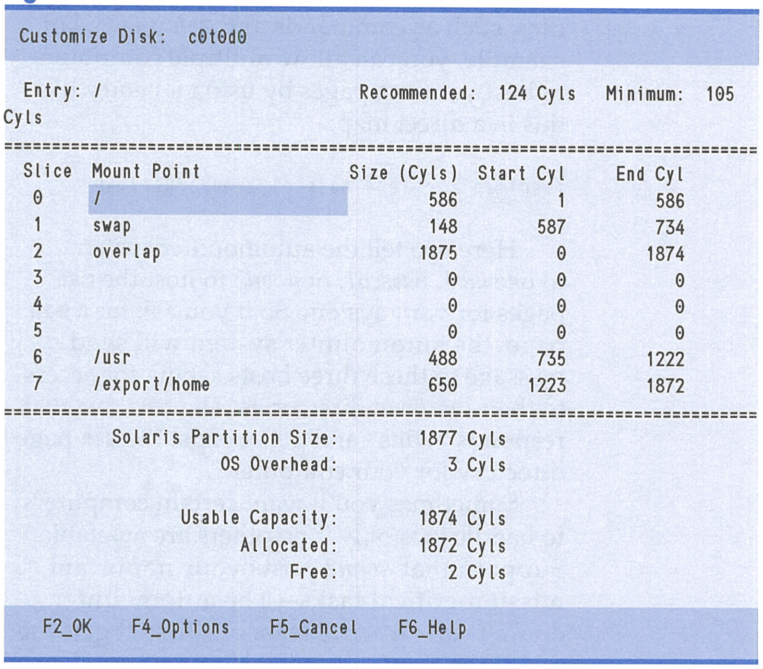
You can avoid this problem (and conserve space) by using this trick: Before you accept the disk layout, instruct the installation program to view the file system sizes in cylinders—not megabytes. This way, you can locate any unused cylinders and change your file systems to insert extra cylinders on the nearest slices. To do so, when you're finished with the screen shown in Figure E, press the [F4] key to bring up the options screen shown here in Figure G.

Set the Show Size In Cylinders and Show Cylinder Boundaries options, and you can see whether you're wasting any space on your hard disk. When you press [F2] to return to the Customize Disk screen, you'll notice that it's changed quite a bit, as shown in Figure H.

Now you can see the problem: In the Free field, two cylinders remain. We simply find any gaps and expand the slice just before each gap to take up any free space. Each time we've done this, we've found one gap at the end, though you might find gaps elsewhere.

First look at Slice 2, labeled overlap: This slice shows you the minimum (Start Cyl) and maximum (End Cyl) cylinder numbers available to you. In this case, we notice that the maximum cylinder number is 1,874; and looking at our */export/home* slice, we see that it goes up to cylinder 1,872. Since the Free field shows us that two cylinders remain unallocated and the */export/home* slice falls short of the maximum cylinder number by two cylinders, there's a single gap of two cylinders. So we increase the size of the */export/home* slice by two cylinders to eliminate the gap.

Figure H



```
Customize Disk:  c0t0d0

 Entry: /                          Recommended:  124 Cyls   Minimum:  105
Cyls
==========================================================================
 Slice  Mount Point                  Size (Cyls)  Start Cyl     End Cyl
  0     /                                 586          1          586
  1     swap                              148        587          734
  2     overlap                          1875          0         1874
  3                                         0          0            0
  4                                         0          0            0
  5                                         0          0            0
  6     /usr                              488        735         1222
  7     /export/home                      650       1223         1872
==========================================================================
          Solaris Partition Size:        1877 Cyls
                     OS Overhead:           3 Cyls

                 Usable Capacity:        1874 Cyls
                       Allocated:        1872 Cyls
                            Free:           2 Cyls

    F2_OK     F4_Options     F5_Cancel     F6_Help
```

You can add, change, or delete slices from this screen to customize your hard drive.

Conclusion

You needn't worry about configuring your hard disk during installation now that you know how to customize the disk slices. No matter what you do, the first few times you install Solaris, you'll probably wish you had chosen differently. But no matter, as you become more experienced, you'll be able to more expertly configure your system. Also, as a system grows and matures, many system administrators restructure their file systems to accommodate changing conditions. ❖

SYSTEM CONFIGURATION

An introduction to the automounter, part 2

Last month, we introduced you to the automounter system, showing you how it works and how it can automatically `mount` and `umount` file systems for you. We likened it to a fast system administrator built into your computer. This system administrator would monitor your usage of the computer, and when you attempt to access data in particular subdirectories, it would interpret the request as a request to `mount` the appropriate file system on another computer. Then, when you no longer need the data in that subdirectory, the fast system administrator would automatically `umount` the file system to save system resources.

In this article, we'll examine some of the more advanced features of the automounter system. Armed with this information, you'll be able to take full advantage of the automounter system at your site, making it simpler to configure your network of file systems. For the purposes of this article, we'll assume that your computer's name is *Servalan*, and other computers on your subnet are *nassco1*, *avon2*, and *lab4*, and one other computer, *bridge3*, is on another subnet.

Weighting factors

As we mentioned last month, you can free a great deal of disk space by allowing a few computers to host infrequently used static files, such as commands and `man` pages. For example, you can allow multiple computers to host your `man` pages by using an entry like this in a direct map:

```
/usr/man        lab4,nassco1,avon2:/usr/man
```

Here, we tell the automounter system to use *lab4*, *nassco1*, or *avon2* to host the `man` pages for your system. So if you ask for a `man` page, the automounter system will send a message to these three hosts asking for access to their */usr/man* directories. The first one that responds "wins" and gets to host the `man` page directory for your computer.

Sometimes you'll want certain computers to handle jobs only if no others are available. Suppose that *avon2* hosts your important, mission-critical tasks—The automounter doesn't care: If *avon2* responds first, it gets the additional work, even if *lab4* and *nassco1* are relatively unburdened. Or maybe *lab4* is a machine that some programmer restarts often. You probably don't want to run any services on *lab4* unless you've no alternative. Or consider this: If *nassco1* contains all the `man` pages already formatted, you can directly view the `man` pages; if another computer hosts the `man` pages for you, you must wait for your computer to reformat and display them.

In this scenario, you'd probably want *nassco1* to host `man` pages whenever possible, so you can quickly view them. However, if *nassco1* is unavailable, you wouldn't want to hang indefinitely. You'd want to try to get them from *avon2* or *lab4*.

One way you can do this is to assign weights to the computers to adjust their "desirability" to the automounter. The automounter system tries to use computers that have the lowest weight when possible. (Any computer with no weight is assumed to have a weight of zero.) So by assigning one weight, say 50, to *nassco1*, and an even heavier weight, say 100, to *lab4*, you can encourage the automounter system to use *nassco1* first, whenever possible, and *lab4* only when you must. You add the weights by simply putting the weight value in parenthesis next to the computer's name in a map, like this:

```
/usr/man    lab4(100),nassco1,avon2(50):/usr/man
```

As you can see, the order in which you enter the computer names is immaterial. Whenever the automounter system attempts to map to your */usr/man* directory, it first sends a message to all the computers having the lowest weight, *nassco1* in this case. If *nassco1* fails to respond, then it tries the next weight category, proceeding successively to heavier weights until it either finds a computer that can host the */usr/man* directory or until it runs out of candidates.

Please note that weighting doesn't work across the board: Computers on the same network segment always get priority over computers on other network segments. Suppose your map looks like this:

```
/usr/man       bridge3,lab4(100)
```

Since *lab4* is on your subnet and *bridge3* is on another subnet, *lab4* will always take precedence over *bridge3*, whether you like it or not. Sun configured it this way to help prevent network congestion. Smaller companies using a single network segment won't notice it, and larger companies with a segmented network must be able to control network usage. If *lab4* is really less desirable than *bridge3*, then don't bother specifying it as an alternate.

You'll find that weighting can be a useful tool. If you go to the trouble of creating a large, fast, finely-tuned file server, you probably want people to use it, rather than relying on one of their neighbor's computers to service some of your file requests. Thus, you can assign weights to all your back-up resources, so computers on the same subnet will use the big file server when it's available and use files on smaller computers only when it isn't available.

You can use it to retain particular services on specific machines, as well as to compartmentalize resources by department. For example, you might set up each department as an autonomous unit, with its own file servers, mail systems, etc. Then, using weighting, you can make computers in each department use their own departmental resources preferentially. Other departments may offer alternative services to each other—in case one department's file server goes down, for example.

Variables

The automounter system really starts displaying its power when you must configure lots of computers. It turns out that the automounter system provides a tool—variables—that makes it relatively easy to share your map files among computers. You can use variables to help you configure specific computers with custom options.

The automounter system provides plenty of variables to customize your map files. Table A shows the variables to configure your map files built into the automounter system.

Suppose you want to provide a big, beefy file server that hosts infrequently used executable files for all your computers. However, you may have computers running multiple versions of Solaris and multiple architectures. You may have a fleet of SPARCstations running Solaris 2.5 and a few running version 2.6 until your company approves it. You may also have a few PC-compatible computers running Solaris x86 2.5. Even with this assortment, you can create a map file that will work with *any* of these computers:

```
/usr/ucb     BigSrv:/Sol$OSREL/$CPU/usr/ucb
```

Once you distribute the map that contains this entry, each computer will interpret the statement according to the version of Solaris it's running and the CPU it's running on. Thus, an Intel-based computer running Solaris 2.5 will access its binaries from BigSrv:/Sol5.5/i386/usr/ucb, while a SPARCstation LX running Solaris 2.4 will get them from BigSrv:/Sol5.4/sun4m/usr/ucb. Similarly, the automounter system can keep the versions straight as well.

Creating your own

While Table A lists all the variables that the `automountd` system provides, you're not limited to using only those listed. You can also create your own. To do so, you simply use the -D*name*=*value* switch on the `automountd` command when you start it. This sets the variable named *name* to *value*.

With this technique, you can further customize your system. Suppose, for example, each department has its own server. You could easily configure each person's computer to use its department's server as the first choice and another department's server if the first choice is unavailable. You could set the DEPT variable to the appropriate value depending on the department the person is in. To do so, just start the automounter daemon (`automountd`) like this:

```
automountd -DDEPT=sales
```

Please note that you must modify the automounter system's startup script, */etc/init.d/AUTOFS* to do so. This will allow you even finer control over the automounter system.

Referring back to our example of keeping departments separate: We can define the variable DEPT to hold the department name and use it in our maps. So if you have sales, application development, R&D, manufacturing, shipping, and accounting departments, you can create some shorthand names for the departments and use them as variables for configuration.

If we name our file servers based on the departments, like *sls_srv* for sales, *rnd_srv* for R&D, and *man_srv* for manufacturing, then everyone can use a map like

```
/usr/local ${DEPT}_srv,sls_srv(10),
➥rnd_srv(20),man_srv(10):/usr/local
```

This map tells our computer to use our department's server when possible. If not, we'll try the sales or manufacturing servers next, since they have the next-lowest weights. If it turns out that they are unavailable, we'll finally try the R&D server.

Hostname aliasing

However, enforcing rigid names on your servers can often be problematic. For example, if your company is small, you might start out with a single server for sales and manufacturing. So what do you name your computer: Do you name it *sls_srv* or *man_srv*?

You may also revolt at having such a naming structure. Many Solaris users delight in naming their computers with themes: The sales department may name its computers after the seven dwarves, while the R & D group may name its computers after its favorite authors. Giving computers boring names like *sls_srv* is just another nail in the coffin of a delightful corporate culture.

A solution does exist that's elegant and preserves corporate culture; it also further simplifies automounter management. You can allow your users to give their computers any name they want; for management purposes,

Table A

Variable Name	Description	Example Value
ARCH	Architecture name	i86pc
CPU	Processor type	i386
HOST	The computer's name	Servalan
OSNAME	The name of the OS	SunOS
OSREL	The release of the OS	5.5
OSVERS	The OS version	Generic

You can use these variables in your map files to customize your system configuration.

you can nickname their computers (as described in the article "Giving Nicknames to Hosts" on page 11) based on services they provide. The automounter will gladly use aliases for computer names, so your map files can be simpler to configure.

Returning to our original question: Since you can give multiple nicknames to your computers, you can go ahead and name the computer *Mikado* (the manufacturing group names its computers after Gilbert & Sullivan operettas) and give it the nicknames *sls_srv* and *man_srv*. Thus, as far as the automounter system is concerned, they are two distinct computers. When the sales department gets its own server, you can simply move its files to the new computer, then move the nickname to the new computer. You won't have to change your rule maps.

Automounter management

Managing an NFS-based system can be troublesome, since you must modify the *vfstab* file (or script files that perform the `mount` and `umount` operations) on each computer. The secret to making the automounter system work best is to take advantage of various tricks that allow you to use a few configuration files to work for a large group of computers.

Understanding variables is an important first step, since variables allow you to make extremely flexible system configurations. You can partition services throughout your system with the proper use of variables. Combined with host nicknames, you can plan ahead for expansion, making expansion a simpler step. Once you create your map files, you should test them on a few computers and ensure that they do exactly what you want. Then you can distribute the map files *en masse*.

While planning is the key to making it all work together, you can never foresee all events, so you must be flexible. If you're finding it difficult to create a single set of maps to suit all purposes, then don't try. It may be inappropriate to make the manufacturing department use the same maps as R & D. The goal is to simplify administration. Having different maps for each department may be the easiest solution. In any case, it's better than the previous alternative of using NFS and hand-editing the */etc/vfstab* file on everyone's computers.

Conclusion

Now you should have a good understanding of the automounter system and what you can do with it. In this article, we've discussed some advanced configuration techniques you can use to get the most out of the automounter system. But by no means have we exhausted the options of the automounter system. In future issues, we'll revisit the automounter from time to time, showing you other interesting tips and tricks. ❖

ANNOUNCEMENTS

Solaris 2.6 is branded UNIX 95 compliant

The newest member of the Solaris family, Solaris version 2.6, has been branded "UNIX 95" by The Open Group. This branding means that version 2.6 conforms with The Open Group's XPG4 UNIX PROFILE specification, which shows Sun's commitment to working with other vendors to increase the popularity of UNIX. The membership of The Open Group includes all major UNIX vendors sharing a similar commitment.

For anyone who's followed Sun's activities in standardization and interoperability, Sun's compliance will come as no surprise. It's a logical extension of the work Sun has been doing for years, as shown in Table A. Each time The Open Group, of which Sun is both a sponsor and member, creates a new interoperability standard, Sun ensures that the next version of Solaris complies. For example, Solaris 2.5 complies with XPG4 Base 95, and Solaris 2.4 introduced compliance with the XPG4 Common Desktop Environment (CDE).

Impact on users

The UNIX 95 brand makes it easier for users to move among different versions of UNIX by standardizing the operation of many commands. (It builds on earlier efforts, such as POSIX.1.) Thus, you can be sure that the core set of commands you use on one UNIX

95-branded version of UNIX, such as HP-UX 10.2, will work on Solaris 2.6.

Impact on application developers

The XPG4 UNIX PROFILE specification also affects software development. Compliant versions of UNIX make a specific set of services available to programmers. Each service must operate the same way on each platform, making it simpler to create applications that may be ported to all the major UNIX versions.

Impact on maintenance

Another great advantage of the UNIX 95 branding is the vendor guarantee. When a company wants to certify its software with one of The Open Group's specifications, it must guarantee three things:

- The product works in conformance with the specification.
- Throughout the product's life, the vendor will maintain compliance with the specification.
- Any problem will be fixed in a timely fashion.

The future

The Open Group is currently well under way on the UNIX 98 and UNIX 98 WORKSTATION specifications, which include these new and important features:

- Thread support to improve performance on multiprocessor hardware
- Support for very large (> 2GB) files
- Dynamic linking extensions for code sharing and simpler maintenance
- Removal of architectural dependence to support larger word sizes (such as 64 bits)

Sun is already well on the way toward providing compliance with these new specifications, even though they aren't due to become standard until the end of the year. For more information on The Open Group, check out its Web page at *http://www.rdg.opengroup.org/*. ❖

Table A

X/Open Brand(s)	Sun's Compliant Systems
UNIX 95 and XPG4 UNIX PROFILE	Solaris 2.6
XPG4 Base 95	Solaris 2.5
XPG4 Common Desktop Environment	Solaris 2.4 with Solaris CDE 1.0.1
UNIX 93, XPG4 Base PROFILE	Solaris 2.5.1 (PowerPC), Solaris 2.4[1]
XPG3 Base PROFILE	Solaris 2.1[1]

[1] *When used with the SPARCompiler for SPARC systems or the ProCompiler for x86 systems.*

Sun continues its strong history of working with The Open Group to improve UNIX over the years.

SYSTEM MANAGEMENT

Giving nicknames to hosts

How often do people rename the hosts on your network? Many users will never rename their hosts, while others may do so frequently. The larger the network, the more likely someone will rename a host on any given day. If you're a nomadic worker moving your computer from site to site, how do you keep the computer names straight? Or what if you're moving some important services from one computer to another?

It turns out that Solaris' ability to give nicknames to hosts can ease all these situations. In this article, we'll show you how to give nicknames to hosts and how you can use nicknames to simplify these situations.

Using nicknames instead of host names

Why use nicknames at all? Just what do they buy you? Let's think about it for a moment:

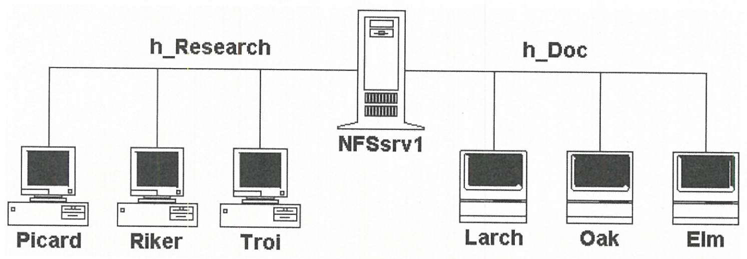
How do you select host names? Often, all the computers in a group are named with a common theme.

For example, the Research and Documentation group might name their computers after characters in their favorite television show, i.e., *Picard*, *Worf*, *Riker*, and *Troi*. A botany division might name their computers after trees, such as *Larch*, *Oak*, and *Elm*. These names provide a sense of unity for a group, but give no inkling of how the machine is actually used.

Giving nicknames to hosts allows you to give logical names to hosts that indicate what services they may provide. For example, if *Worf* has a large hard drive, you might want to use it to host all the `man` pages, freeing disk space on other computers. If you know that one of the computers hosts the `man` pages and you don't know which one, prepare to search all of them. You could give *Worf* a nickname, such as *h_man*, so you can find the host for `man` pages more easily. (We use nicknames starting with an h_ to indicate a nickname for a computer that's hosting a service.)

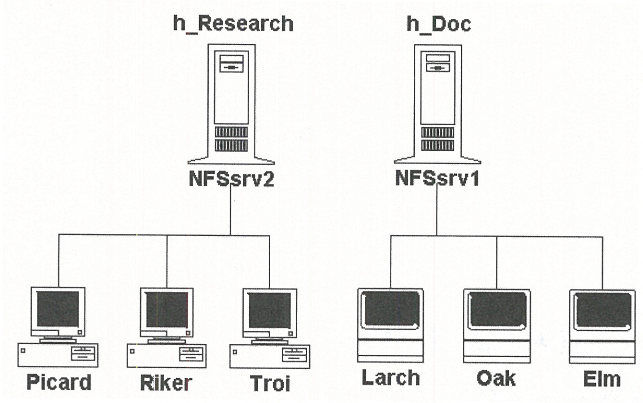
In some installations, you may even have named some of your computers with functional names, such as *NFSsrv1* and *printsrv2*.

Figure A



Before you split your network, notice that your Research and Documentation teams are already logically split.

Figure B



Since you used nicknames, you can physically split the departments without reconfiguring the client workstations.

This practice is great, until you must exchange one server with a more capable unit. While you can set up the new computer exactly like the one it will replace, then swap them out, it often makes more sense to simply add the new, larger computer to the network.

For example, suppose you have a single NFS server, named *NFSsrv1*. Unfortunately, both the Research and Documentation groups have exceeded the capacity of *NFSsrv1*. So you get another computer, name it *NFSsrv2*, and move the Research group to *NFSsrv2*. Now you must update the configuration of all the computers in the Research group to use the new NFS server.

Nicknames could help in this situation. Suppose you initially gave *NFSsrv1* two nicknames: *h_Research* and *h_Doc*. Your Research group would access the *h_Research* server, and your Documentation group would access *h_Doc*, as shown in Figure A. It doesn't matter to them that it's the same host.

Later, when you add the new computer, you could remove the *h_Research* nickname from *NFSsrv1* and add it to *NFSsrv2*, as shown in Figure B. If you copy the files to the new server, then your Research group needn't know that you've moved their server, and you needn't modify all their scripts and configuration files.

Alternatively, you may have multiple services on a single computer, such as mail, printing, and WWW proxy. If you find that two services demand too many of the same resources, you can move one to another computer. If you use nicknames to identify services, then you needn't worry about reconfiguring all the client computers.

Giving a nickname to a host

If you're not using a name service, you can add a nickname to a host by editing the */etc/hosts* file. The first column of */etc/hosts* specifies the IP address of the host computer, and the second column specifies the official, or canonical, name. Subsequent columns define nicknames for the host.

I'm a nomadic worker, and I use my computer, named *Servalan*, in multiple networks. In Figure C, I show excerpts from my */etc/hosts* file to indicate how I use nicknames. Each time I move *Servalan*, I edit the */etc/hosts* file to uncomment only the lines dealing with the network where *Servalan* is currently connected. (Please note that I must also edit other files when I move *Servalan* from network to network.)

You'll notice that each configuration has a host nicknamed *h_DNS1*. This way, I can use a common name to access our primary DNS server in all environments. Similarly, the *h_man* nickname gives Servalan access to a computer that holds the text for all `man` pages, so *Servalan* doesn't need to store them.

Adding nicknames with DNS

Maintaining the */etc/hosts* file can be an exercise in futility, especially when you're connecting to multiple networks. (Keeping track of the hosts in a single network is tedious enough.) A better way to use nicknames is to use them with a naming service.

If the administrators of the networks you're connecting are amenable, they may provide nicknames to hosts containing services you plan to use. This way, you can avoid the hassle of editing your */etc/hosts* file. Keep in mind, however, that you're passing the work on to the network administrator. However, they may want the advantages of using nicknames on their systems, so it may be possible to get all of them to provide this service. (Not all administrators will cater to you, so you must be comfortable with both DNS databases and your */etc/hosts* file if you're a nomadic user.)

Just how do you provide nicknames under DNS? You modify the DNS databases. Fortunately, the DNS databases provide a record type that allows you to specify an alias (i.e., nickname) for a host. The format of an alias record for DNS is

```
nickname.domain. IN CNAME host.domain.
```

where `nickname.domain.` is the fully-qualified name of the nickname, and `host.domain.` is the fully-qualified name of the host where you're adding the nickname.

Definition: A fully-qualified domain name (FQDN) is a name that ends with a dot (.) specifying the top-level domain. On the Internet, the domains com, net, mil, gov, etc., are all subdomains of ".". The Cobb Group's FQDN is cobb.com., note the trailing dot!

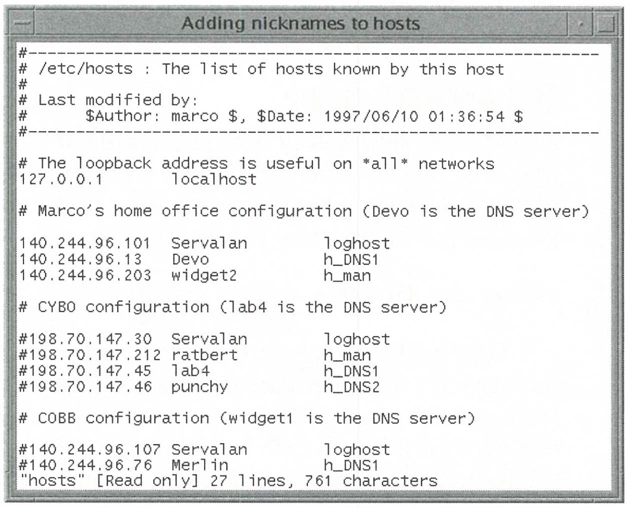
Figure D shows a sample DNS database file that specifies nicknames to some hosts on my home office network. *Servalan* runs Solaris x86 and hosts all the `man` pages as *h_man* and i86pc architecture executables as *h_i86pc*, whereas *widget2* hosts the SPARC executables as *h_sun4m*. *Devo* happens to host the primary CD-ROM drive as *h_CD1*.

Testing your nicknames

Once you create your nicknames, whether in the */etc/hosts file* or DNS, you should test them. (Please note that you can also use nicknames with NIS+ in the hosts table.) You can go through all kinds of trouble to test your nicknames. The `ping` command is one simple way to do so. You just `ping` the host name by its nickname. If all goes well, `ping` will respond by telling you that the host name is alive. Please note that the host name specified will be the canonical host name of the computer, as shown here:

```
$ /usr/sbin/ping h_man
Servalan.marcorp.com is alive
```

Figure C

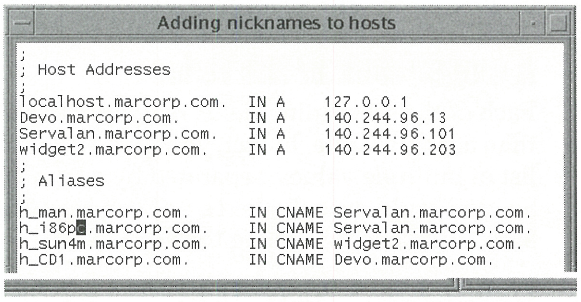


Adding nicknames to hosts

```
#-----------------------------------------------------------------
# /etc/hosts : The list of hosts known by this host
#
# Last modified by:
#       $Author: marco $, $Date: 1997/06/10 01:36:54 $
#-----------------------------------------------------------------

# The loopback address is useful on *all* networks
127.0.0.1         localhost

# Marco's home office configuration (Devo is the DNS server)

140.244.96.101  Servalan        loghost
140.244.96.13   Devo            h_DNS1
140.244.96.203  widget2         h_man

# CYBO configuration (lab4 is the DNS server)

#198.70.147.30  Servalan        loghost
#198.70.147.212 ratbert         h_man
#198.70.147.45  lab4            h_DNS1
#198.70.147.46  punchy          h_DNS2

# COBB configuration (widget1 is the DNS server)

#140.244.96.107 Servalan        loghost
#140.244.96.76  Merlin          h_DNS1
"hosts" [Read only] 27 lines, 761 characters
```

You can add nicknames to hosts in the /etc/hosts file.

Figure D



Adding nicknames to hosts

```
;
; Host Addresses
;
localhost.marcorp.com.   IN A     127.0.0.1
Devo.marcorp.com.        IN A     140.244.96.13
Servalan.marcorp.com.    IN A     140.244.96.101
widget2.marcorp.com.     IN A     140.244.96.203
;
; Aliases
;
h_man.marcorp.com.       IN CNAME Servalan.marcorp.com.
h_i86p█.marcorp.com.     IN CNAME Servalan.marcorp.com.
h_sun4m.marcorp.com.     IN CNAME widget2.marcorp.com.
h_CD1.marcorp.com.       IN CNAME Devo.marcorp.com.
```

The DNS database specifies nicknames for services on our (arbitrarily named) hosts.

For more information about DNS, read the article "Configuring Solaris as a Primary Master DNS Server" in our April issue.

Conclusion

Giving nicknames to hosts can be a powerful tool. If you use your computer in multiple networks, you can use a few nicknames rather than learning the names of many computers in multiple networks. If you want to expand your network, careful use of nicknames can ease the pain of expansion. Also, if your users change host names as often as their moods, you can be somewhat insulated from their carryings on. ❖

SYSTEM UTILITIES

The second Sunday of each month...

You probably already know that cron will execute jobs for you on a schedule (especially if you've read the article "Scheduling a Job for Periodic Execution" in our October '96 issue). For example, it's easy to make a script file execute every day at 3:00 a.m. But do you know how complex a cron schedule can really be? Well, with a bit of thought, you can make some pretty complex schedules.

The crontab entry

As you may recall, each user has a table of jobs that cron will execute. You can use the crontab command to list, edit, or remove your table of jobs. Each crontab entry follows this format:

```
minute hour day month weekday command
```

The first five fields (minute through weekday) specify the schedule, and the last field (command) specifies the job that cron will execute for you. When each field is true, cron will execute the command for you. If you want to ignore the field, you simply use an asterisk (*) for the value. Thus, to execute the df command every day at 3:00 a.m., you use the entry

```
0 3 * * * df
```

A closer look at the fields

Each of the scheduling fields may hold more than a single value. In fact, you may enter a list of multiple values, separated by commas. So, you could execute the ls -al command at 8:15, 12:15, and 4:15 using the entry

```
15 8,12,4 * * * ls -al
```

Here's where it starts to get interesting: You can also specify *ranges* of values. So, you can make interesting schedules, such as "I want to execute the foo bar command at midnight and each hour during the business day (8-5)." You could do so like this:

```
0 0,8-5 * * * foo bar
```

Combining the fields

Remember how we said that cron will execute your command when all the fields are true? This fact, combined with lists of values and ranges, allows you to make demanding schedules.

As an example, suppose you want cron to execute the xyzzy command at noon on the Wednesday after the second Sunday in any month that starts with a J. Yes, cron can do this, but it'll require a bit of effort on your part. Obviously, we're going to set the minute to 0, hour to 12, and weekday to Wednesday. (The weekday field is: 0=Sunday, 1=Monday, ...)

```
0 12 ? ? 3 xyzzy
```

Then, we look at a calendar to find the months starting with J, and we'll fill the month field with a list of these months (January=1, February=2, etc.):

```
0 12 ? 1,6,7 3 xyzzy
```

Now we have a line that will execute xyzzy at noon on a Wednesday in January, June, and July. Now we must figure out what to use for the day field to ensure that it executes only on the Wednesday after the second Sunday in the month.

Well, if the first Sunday fell on the first of the month, we'd want to run the command on the eleventh day of the month. Similarly, if the first Sunday fell on the seventh of the month, we'd want to run the command on the eighteenth of the month. Since there are only seven days in a week, the first Sunday *must* fall between the first and seventh, so we can specify a day range of 11 to 18, giving us the line

```
0  12  11-18  1,6,7  3  xyzzy
```

Since all fields must be true before `cron` will execute the command, it will execute only at noon (hour=12, minute=0) on the Wednesday after the second Sunday (day of week=3 (Wednesday), day of month=11 through 18), on a month starting with J (month=1,6,7).

Here's a simpler example: Let's execute the `plover` command at 1:00 on the second and fourth Monday of each month. Here again, we use the day of the week in concert with day ranges:

```
0  1  8-14,22-28  *  1  plover
```

Caveats

Looking back at our examples, do you wonder why we specified a time on each of them? If you don't specify a time, then `cron` will bog down the system executing your program every minute, *all day long* on the specified days. For example, the entry

```
*  *  1  *  *  ls -al
```

will execute the `ls -al` command every minute on the first day of the month: a total of 1,440 times. Be sure to consider the effect of any field that you choose to ignore.

You may also find that you have a schedule so complex, you can't find a way to schedule the command for `cron`. For example, how do you find the amount of free disk space at the start (7:30) and end (5:00) of each working day? How do you tie the proper hour and minute fields together. The answer is: You can't. In cases like this, just go ahead and make multiple schedules:

```
30   7  *  *  1-5  df
0   12  *  *  1-5  df
```

The moral is this: Don't get hung up on trying to figure out an impossible schedule. You're not limited to running only one entry. ❖

LETTERS

Packaging application updates

The article "Packaging Groups of Files for Distribution" [in the January issue] was on a very good subject and I liked it. Besides this, I also have the book *Application Packaging Developer's Guide*, which is also very helpful.

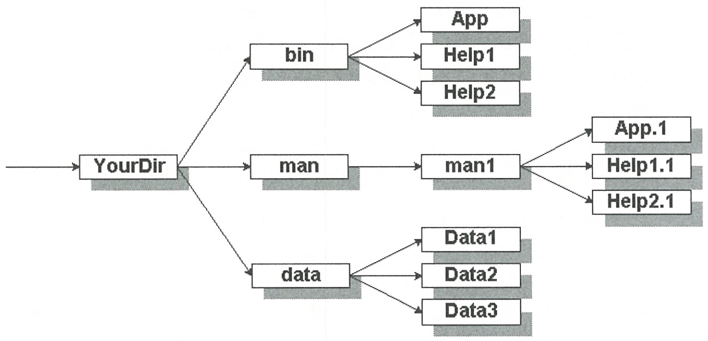
My problem is this: I can make a single package now, but I need information on how to write updates for my packages. (If there are some bug fixes, I need to make a bug-fix package and update the users' machines.) I can't find any useful book on this topic. Can you help me out? Thanks very much!

Qian Zhao
via the Internet

You can create an update package in several ways, depending on the method you normally use to distribute patches. For example, suppose you want to simply distribute a set of files to replace files in an already-installed package. Let's assume that you have a directory tree with all the files required by your package, as shown in Figure A.

When you created your first package, you used the method we described in the January issue. Now let's say one of your sites finds a bug in your *Help2* file that requires you to change *Help2*, *Help2.1*, and *Data3*.

Figure A



This directory tree contains all the files your package requires.

SunSoft Technical Support

(800) 786-7638

PERIODICALS MAIL

```
*************************3-DIGIT 300
2096 C 5147831 9806
```
[redacted]
```
CUMMING GA 30041-1232
```

Please include account number from label with any correspondence.

If you want to create an update package that simply replaces the updated files, you can follow almost the same steps you did to create the original package. The difference is that you'll include only those files that changed. Since you know the date you created your package, you simply change the `find` statement to use only the files that changed since your original release. So, rather than using

```
$ find . | pkgproto >prototype
```

you would use

```
$ find . -mtime X | pkgproto >prototype
```

where *X* is the number of days since you created your package. Using this as the starting point for your package, you can create a package containing only the new files. The remaining steps are basically the same as before.

You may want to manually remove or add some files. For example, if replacing *Help2* forces you to reset *Data1* to a known value, you can manually add *Data1* to the file *prototype*. (Or, you can `touch` *Data1* before the `find` statement, so it's automatically included.) Similarly, if a file, say, *Data2*, contains data that you (or your clients) may be changing, you don't want them to lose their data. If you use your program, the *Data2* file may have been modified since you created your package, so you must remove *Data2* from the file *prototype* before packaging the files.

If you want to replace only parts of files, you must be a bit more clever. First, you create the patches, such as with `diff` (for a text file). Then you can write a `postinstall` script that will incorporate your patches into the (already installed) files.

In order to protect your users from errors, you should patch the files in a temporary area and copy them to the working area only after all the patches are installed correctly and after you back up the original files. You should also provide a script that will undo the changes you just made, in case they cause the user more problems than the bug you're trying to fix. ❖

Solaris resources on the Internet

You can find many Solaris resources on the Internet. Some are for UNIX in general; others are specific to Solaris. Here are a handful of our favorite Internet resources:

- Sun Microsystems provides an excellent Web site for Solaris users at *http://access1.sun.com/*. You can find the latest drivers, patches, technical information, and answers to frequently asked questions at this site.
- The SunSolve page at *http://sunsolve.sun.com/* is another valuable resource. It's a subscription-based site, so access is limited for nonsubscribers.
- If you want to get binaries of your favorite GNU (and other generally-available programs) for SPARC, you'll want to check out S. Christenden's site at *http://sunsite.univalle.edu.co/Solaris/solaris_2.5_nof.html*.
- Solaris x86 users owe it to themselves to visit *ftp://x86.cs.duke.edu/pub/solaris-x86/bins/*, where Joe Shamblin of Duke University keeps a large selection of Solaris x86 programs. Youri Podchosov also makes a nice archive of Solaris x86 binaries available on his Web site at *http://ynp.dialup.access.net/attic/Solaris-x86/*.
- Solaris x86 users should subscribe to the Solaris x86 mailing list, provided by EIS Computers, Inc. (*http://www.eis.com*). To subscribe, send a message to *solaris-x86-request@mlist.eis.com* with the word SUBSCRIBE in the body of the message.

If you'd like to share your favorite Internet resources with other readers, send them to *inside_solaris@cobb.com*. ❖

Printed in the USA
This journal is printed on recyclable paper.