December 1997 • $11.50
Vol. 3 No. 12

ZIFF-DAVIS
THE COBB GROUP

INSIDE SOLARIS™
Tips & techniques for users of SunSoft Solaris

IN THIS ISSUE

Visit our Web site at
http://www.cobb.com/sun/

Displaying a directory tree with find and sed

by Alvin J. Alexander

We often run into situations where we need to see a tree-like view of our file systems. Being able to see our file systems in such a way is helpful when we're looking for a lost file or when we're trying to see how another user or company has organized a particular file system.

Although Solaris doesn't come with a built-in `tree` command, we'll demonstrate how you can use Solaris' `find` and `sed` commands in a single pipeline to create your own customized tree utility. We'll also see why `sed` is such a powerful command.

Figure A

```
|____home
|    |____aja
|    |    |____menus
|    |    |____cobb
|    |    |    |____1997
|    |    |    |    |____may
|    |    |    |    |____jun
|    |    |    |    |____jul
|    |    |    |    |____aug
|    |    |____bin
|    |    |____lib
|    |    |____tmp
|    |    |____customers
|    |    |    |____ge
|    |    |    |____ford
|    |    |    |____kfc
|    |    |    |____asl
```

A `tree` command can show you a simple graphical representation of your file-system hierarchy.

Tree-command output

To see where we want to be, let's look at how we want our tree output to appear. Figure A shows how we want the output to look when we use our new `tree` command to view the directory */home*. We created this output by typing *`tree /home`*, where *aja* is the only subdirectory of the */home* directory.

In Figure A you can see that the */home* directory is at the top of the listing, and *aja* is a subdirectory of */home*. Beneath *aja* are several other subdirectories, including *menus*, *cobb*, *bin*, *lib*, *tmp*, and *customers*. Now that we have our vision of the output we want to generate, let's create our program.

Finding the directory structure

First, when building our `tree` command, we must determine the file system's tree structure. It turns out that you can use the `find` command, specifying the `-type d` option, to print out a list of the directory structure. No further options, other than the usual `-print`, are required. Therefore, we'll use this basic `find` command for our `tree` script:

```
find $dirToSearch -type d -print
```

Notice that we're using a variable named `dirToSearch` in this command. We're putting a variable

SOFTBANK
ZIFF-DAVIS

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
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
Louisville, KY 40220
Internet: cobb_customer_relations@zd.com

Staff
Editor-in-Chief ..... Marco C. Mason
Contributing Editors ..... Al Alexander
Print Designer ..... Margueriete Winburn
Editors ..... Karen S. Shields
Joan McKim
Publications Coordinator ..... Linda Recktenwald
Managing Author ..... Eddie Tolle
Product Group Manager ..... Michael Stephens
Circulation Manager ..... Mike Schroeder
Publisher ..... Jon Pyles
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.
Postmaster: Send address changes to:

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

at this point in the `find` command to make our program more flexible. We want our shell script to be flexible so an end user can use the `tree` command in any of these ways:

```
$ tree
$ tree /home
$ tree /home /user /bin
```

Within our `tree` script, we'll test to see how many command-line arguments the user has supplied by checking the Bourne shell variable `$#`. For instance, if the user types

```
$ tree
```

then the number of command-line arguments is zero, and we'll assume that our user wants a tree view of the current directory. In this case, we'll set `dirToSearch` as follows:

```
dirToSearch=.
```

If the number of command-line arguments is greater than zero, we'll assume that the user has specified one or more directory names on the command line. So we'll set `dirToSearch` equal to the number of command-line arguments the user supplied. We'll do this with the Bourne shell variable `$*`, which contains the actual command-line arguments the user entered:

```
dirToSearch=$*
```

With these decisions in mind, the first cut of the source code for our `tree` utility looks like this:

```
#!/bin/sh

if [ $# -gt 0 ]
then
    dirToSearch=$*
else
    dirToSearch=.
fi

find $dirToSearch -type d -print
```

The final line in our program runs the desired `find` command. The only problem with using the `find` command is that when we run our script, we get the output shown in Figure B rather than the `tree` output we want:

Figure B

```
/home
/home/aja
/home/aja/menus
/home/aja/cobb
/home/aja/cobb/1997
/home/aja/cobb/1997/may
/home/aja/cobb/1997/jun
/home/aja/cobb/1997/jul
/home/aja/cobb/1997/aug
/home/aja/bin
/home/aja/lib
/home/aja/tmp
/home/aja/customers
/home/aja/customers/ge
/home/aja/customers/ford
/home/aja/customers/kfc
/home/aja/customers/asl
```

The output of our `find` command shows the directory hierarchy, but it's not what we want.

The repeated information is just too dense, so a minor change in a directory name might not be immediately apparent. Drawing the directory structure as a tree makes it simpler to see the hierarchy of our file system when we look at it. So we must find a way to format the data so it appears the way we want. This is where `sed` comes into play.

Reformatting the directory tree

If we wish to have `sed` to reformat our output, we must design the appropriate transformation rules. We formulated our set of rules by comparing what we have to what we want. At first glance, how to get what we want isn't readily apparent, but if we compare our output one line at a time, the process gets a bit simpler. This line shows what we get with the `find` command:

```
/home/aja/cobb
```

Here's what we really want:

```
|    |    |___cobb
```

After some consideration, we find that we can transform the line

using rules. First, we'll *convert any group of characters followed by a forward slash into a vertical bar followed by four underscores.* Running the output of `find` through this rule converts the output in Figure A to that shown in Figure C.

Figure C

```
|____home
|____|____aja
|____|____|____menus
|____|____|____cobb
|____|____|____|____1997
|____|____|____|____|____may
|____|____|____|____|____jun
|____|____|____|____|____jul
|____|____|____|____|____aug
|____|____|____bin
|____|____|____lib
|____|____|____tmp
|____|____|____customers
|____|____|____|____ge
|____|____|____|____ford
|____|____|____|____kfc
|____|____|____|____asl
```

Converting a group of characters followed by a slash into four underscores and a vertical bar improves the output, but we're not finished yet.

As you can see, with just the first rule, we're almost there. However, the horizontal lines don't suggest the appropriate grouping that we want. The first rule indiscriminately ties each filename all the way to the leftmost vertical line, indicating that the file is placed in the starting directory.

Our output just isn't treelike yet. So now we must formulate our second transformation rule. Again, let's examine the line we have and compare it to what we want. We have

```
|____|____|____cobb
```

and we want

```
|    |    |____cobb
```

The second rule was easier to deduce. We want to keep only the underscores before the filename and convert the rest to spaces. Since all other underscores precede a vertical bar, we'll simply *convert all groups of underlines followed by a vertical bar into spaces* to get the results we want. This operation converts Figure C into Figure A. (Please keep in mind that this is only one possible method. You may implement a different technique for your environment. You might include error-checking, for example, which we're ignoring for our example.)

Implementing the rules

Now that we know the transformation rules we want to use, all that remains is to apply them. When we pipe the output of the `find` command through `sed`, we use the `-e` option to enforce our rules. Our first rule for the `sed` command is easily the most complicated. It looks like this:

```
s;[^/]*/;|____;g
```

The `s` tells `sed` that we're going to create a substitution rule. The semicolon tells `sed` that we're using semicolons as our delimiters. So, in effect, we're telling `sed` to find the pattern `[^/]*/` and replace it with `|____`. (The pattern matches any collection of characters followed by a forward slash.) Finally, we use the `g` flag to tell `sed` to apply the rule each time the pattern occurs on the line.

> Quick Tip: Most people use a slash (/) as the delimiter for `sed` commands. However, if you're using a slash in your regular expression, you must escape the slash inside your regular expression. In cases like this, it's much easier to use an alternate delimiter for `sed`. You may use any character other than a backslash (\) or newline for your delimiter.

We can test our `sed` command by typing the following command, which provided us with the output shown in Figure C:

```
$ find /home -type d -print | sed -e
➥'s;[^/]*/;|____;g'
```

Now we need only add our second transformation rule to convert the output to the form shown in Figure A. Our second rule simply converts the pattern "`____|`" into "`    |`", using the `sed` command:

```
s;____|;    |;g
```

We can read this code as "search for a pattern of four underscore characters followed by a vertical bar; if you find this pattern, replace it with a series of four blank characters followed by a vertical bar."

We can add the second rule by appending it to the end of our first rule, separating the rule with a semicolon, like so:

```
sed -e 's;[^/]*/;|____;g;s;____|;    |;g'
```

After applying these rules, we'll see the results we want.

Listing A: *tree shell script*

```
#! /bin/ksh

# tree
# Draw a tree-structured directory hierarchy
# usage: tree <dirToStart> ...

if [ $# -gt 0 ]
then
    dirToSearch=$*
else
    dirToSearch=.
fi

find $dirToSearch -type d -print | \
    sed -e 's;[^/]*/;|____;g;s;____|;    |;g'
```

Building the tree script

Now we simply put the code together into a shell script so we can use it whenever we want. Our new shell script, named `tree`, is shown in Listing A. When we use our `tree` command on our */home* directory, the output generated by our `tree` command is shown in Figure A.

Conclusion

The `tree` command can be a useful utility to have in your administration arsenal. Also, building our own `tree` command demonstrates a real application of the `sed` command. ❖

FIND TIP

Finding a file just a few minutes old

by Alvin J. Alexander

You glance at the clock just as it strikes 2:00 p.m.; it looks like you'll survive another Monday. While working at the server console, you relax a little and sip an afternoon java (the beverage, not the current computer fad). Suddenly—without warning—you notice that the system seems to be slowing down. You see the lights flickering furiously on the 20GB disk array. You wait, but the flickering shows no signs of slowing down. A quick check of the `ps` command fails to show anything significant. "Something's wrong," you mumble, as your pulse quickens.

Thus begins an anxious moment in the life of an administrator. As you watch, it appears as though a rogue process and a runaway file are rapidly consuming the free space on a file system. The question is, can you find the file and the process, or must you wait until something breaks?

Defining the problem

Because we can't identify any processes gone bad, we decide that the best approach is to use the `find` command to locate the runaway file. Although we don't know the name of the file or the user who has turned loose the rogue process, we do know that the symptoms began just a few moments ago, at 2:00 p.m. Searching for files by looking at their creation or modification times seems to be the answer.

Find's -atime, -ctime, and -mtime options

System administrators quickly familiarize themselves with the `-atime`, `-ctime`, and `-mtime` options of the `find` command in order to locate files by date range. These options are convenient to use and rather flexible.

The `-atime` option allows you to specify the time the file was last accessed, `-ctime` lets you specify when the file was created, and `-mtime` lets you specify when the file was last changed. They all work identically and take a numeric argument, `days`, like this:

```
$ find / -atime days
```

If `days` starts with a plus (`+`) symbol, you're specifying files more than `days` old; if it begins with a minus sign (`-`), you're specifying files less than *`days`* old. If it's just a number, you're specifying files exactly *`days`* old.

The only problem is that on a computer with a large file system and many users, you could wade through a tremendous number of files, using such a coarse resolution as a day. In our example, with half the day gone and the system failing, the command

```
$ find / -mtime -1 -print
```

may find a lot of extra files that we don't have the time to examine. So, can we search only for files that have been created or modified in just the last 20 minutes?

The solution

Thankfully, the answer is yes. To search for files modified after a certain time, you can follow a two-step process. First, you create a file whose modification timestamp is the desired time, then you can use the `find` command's `-newer` option to locate any files newer than this file.

Returning to our example, suppose we decide at 2:10 p.m. to start looking for the file that seems to be growing without bound. Since we noticed the problem at 2:00 p.m., we'll create a file with a timestamp of 1:50 p.m.

Creating a file of the right age

You can create a file with the appropriate modification date and time with the `touch` command. If you haven't used the `touch` command before, you'll see that it's a unique command you can use to update the timestamp on files. Using `touch`, you can make a file look very old or very new, just by changing its access or modification time. You might do so for a variety of reasons, from updating the timestamp of old files to include them in tape backups, to `touch`ing a file so that `make` will notice the new date and recompile a file.

The `touch` command normally allows you to set the modification date and time of a file to the current date and time, using the syntax

```
$ touch filename
```

If the file doesn't exist, `touch` creates it.

However, we don't want a file with the current time; we want one with a previous time. Fortunately, the `touch` command provides the `-t` option to set the last-accessed date and time, and the `-m` option to set the last-modified date and time. When we specify the time with `touch`, we must specify at least the month, day, hour, and minute. (See the `man` page for `touch` for further details.) So, we can use the `touch` command to create an empty file in the */tmp* directory with a modification timestamp of 1:50 p.m., like this:

```
$ touch -mt 08301350 /tmp/empty_file
```

Looking at the file with the `ls -l` command, we can verify that it has the proper timestamp:

```
$ ls -l /tmp/empty_file
-rw-r--r-- 1 root other 0 Aug 30 13:50
➥/tmp/empty_file
```

Finding newer files

The second step in our search for the runaway file is to use the `find` command with the `-newer` option. We tell `find` to locate any files in the local file system that are newer than our */tmp/empty_file*, which appears to have been modified at 1:50 p.m.:

```
$ find / -newer /tmp/empty_file -local -print
```

Notice also that we add the `-local` option to our command, telling `find` not to check any NFS file systems. It's obvious that our local hard disk is churning, so we don't want to waste time looking elsewhere on NFS-mounted file systems. We can also add the `-type f` option to tell `find` to locate only normal files and ignore directories, links, and other file-system objects.

Once the `find` command locates the new file, we can identify the user and process that created the runaway file. If it really is some type of runaway process, we can terminate the process and remove the file.

Extending the technique

As you can see, the `-newer` option is a powerful feature of the `find` command. You'll notice that the `-atime`, `-ctime`, and `-mtime` options allow only a resolution of whole days. By combining the `touch` command and the `-newer` option, you can have greater control over the files you locate with the `find` command.

Suppose you want to find a file that someone was editing during lunch break on Wednesday (three days ago). Rather than examining the entire list of files modified three days ago with the `-mtime 3` option, you can create two empty files: */tmp/before*, whose modification time is before the user began lunch on Wednesday, and */tmp/after*, whose modification time is after lunch was finished. Then you can tell `find` to locate all the files newer than */tmp/before*, but not newer than */tmp/after* with these commands:

```
$ touch -mt 08271145 /tmp/before
$ touch -mt 08270115 /tmp/after
$ find / -newer /tmp/before ! -newer /tmp/after
➥-local -print
```

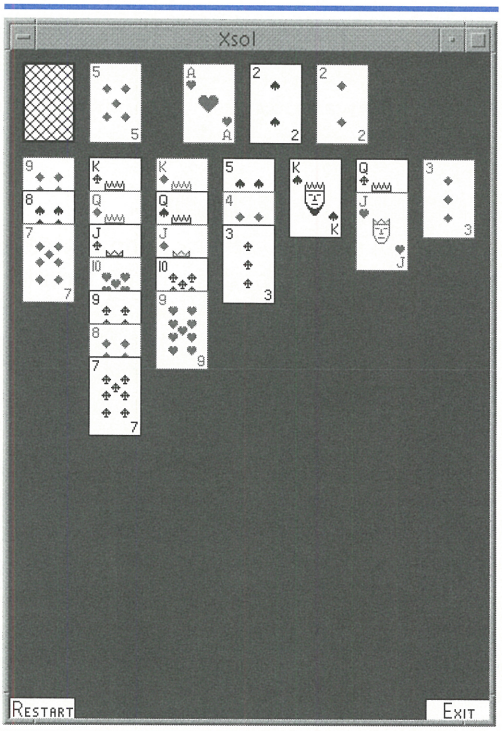
Conclusion

The combination of the `touch` and `find` commands is so good at locating files modified within a precise time period that we use it almost to the exclusion of the `-atime`, `-ctime`, and `-mtime` options. To identify files that change during particular shifts, or when the one-day level of granularity of the `-atime`, `-ctime`, and `-mtime` options of the `find` command is insufficient, we recommend using the `touch` command in conjunction with `find`'s `-newer` option. ❖

What do you do when the system is working?

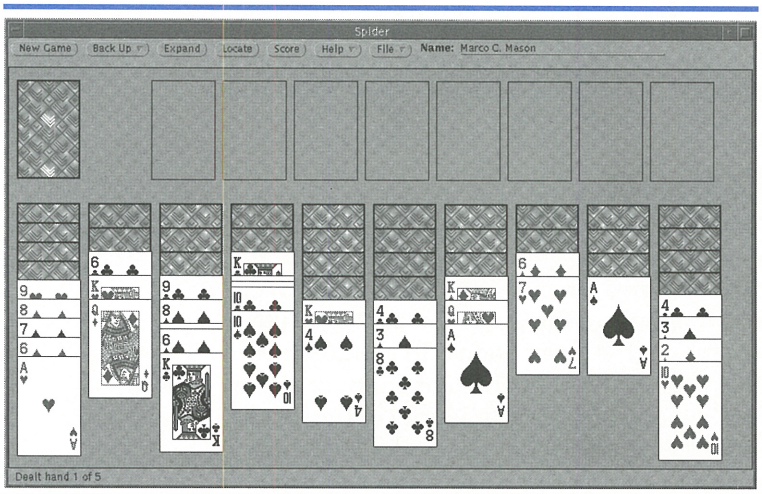
We all know that the best time to take care of things is when your system is working—before anything's broken. However, if you keep your system working perfectly, you'll eventually find yourself in a strange and rare predicament: You'll have something known (by people other than system administrators) as *free time*. For those of you who are unfamiliar with this concept, we'll explain what free time is and what you can do with it.

Figure A



The `xsol` program presents an interesting challenge in arranging patterned rectangles.

Figure B



The spider program offers more challenge in the guise of more rectangles and different movement rules.

What's free time?

As a system administrator, you probably haven't had any free time before. So how do you recognize it, and what do you do with it? While the formal definition of free time is complex and hard to understand, you can detect it with this simple method: When you've completed all the tasks required for the day, look at the clock. If it's well after quitting time (the usual case), you have no free time, thus no bothersome worries about what to do with or about it.

However, it's possible that the clock will indicate a time *before* quitting time. Now you're in that strange zone we call free time. So what do you do with it? Obviously, *more work!* You could get a head start on some of the chores you've scheduled for tomorrow. However, *don't be tempted to fall into this trap!* If you succumb to this temptation, you'll probably have the same problem tomorrow!

You may as well face your fate now. So, what to do? Some advocate taking a long lunch, but this is rarely acceptable since free time usually occurs well after lunch. However, if you're really hungry, you might try it.

Suitable programs

Solaris provides a couple of programs to help you consume your free time in a productive manner. If you've installed the `SUNWoldem` and `SUNWxwdem` packages, then you have two suitable programs with which to consume your free time. Both programs use skills with which you're already proficient, i.e., clicking and dragging, so you needn't learn any new skills to operate them.

However, the first of these programs, `xsol`, presents a challenge to your administrative skills. It displays a somewhat chaotic arrangement of colored rectangles, shown in Figure A, that you must organize. Moreover, you must

move these rectangles according to the rules specified in the `xsol man` page, which can make the task particularly difficult.

Once you've mastered these skills, you may want to try the `spider` program, as shown in Figure B. It's considerably more interesting and challenging, since it presents even more colored rectangles (104, as opposed to 52). Its user interface is quite pleasant, making it less tedious to operate. Also, the rules are similar, but different enough that it will continue to challenge you even after you've mastered `xsol`.

Summary

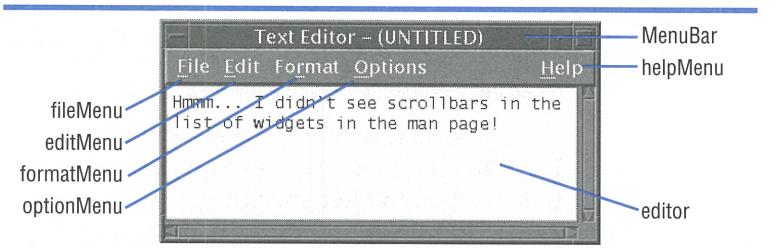
As a system administrator, you're used to facing difficult situations with confidence. However, unfamiliar situations can still have you reaching for a manual. In the event that you're ever faced with that rare dilemma called free time, we hope that we've helped you find an appropriate solution to it. ❖

SYSTEM CUSTOMIZATION

Customizing your X windows programs

The CDE Style Manager offers many customization options you can adjust so that your system will look just the way you want. However, it doesn't always offer you the options you'd like. Your CDE programs are *much* more configurable than you'd guess after using the Style Manager. In this article, we'll introduce you to some of the basic features that X windows offers for configuring your system.

Figure A



The `dtpad` application is built from various widgets, as you see here.

Resources and widgets

X allows you to customize almost everything on your display. Each application window is built from various widgets, such as buttons, text boxes, scroll bars, and menus. Some of these widgets may, in fact, be built from other widgets in a hierarchical fashion. Each widget has a name to identify it. For example, the hierarchy of widgets for the CDE text editor (from the `dtpad man` page) is:

```
dtpad (Dtpad)
   main (MainWindow)
      bar (MenuBar)
         fileMenu (PulldownMenu)
         editMenu (PulldownMenu)
         formatMenu (PulldownMenu)
         optionsMenu (PulldownMenu)
         helpMenu (PulldownMenu)
      editor (DtEditor)
```

So the `dtpad` application is built from a MainWindow widget named main. Then main is built from two widgets: bar (a MenuBar widget) and editor (a DtEditor widget). The bar widget, in turn, contains five widgets, each of which are Pulldown menu objects: fileMenu, editMenu, and so on. Figure A shows a `dtpad` application where we've labeled some of the widgets.

Some of these are standard widgets, such as MainWindow, MenuBar, and PulldownMenu. Many X applications also create their own widgets. For example, the DtEditor widget is the heart and soul of the `dtpad` application, and it handles all the editing. The StatusBar may also be a custom widget.

All standard X widgets have a set of configuration options, called resources, that govern their appearance and behavior. Similarly, the applications have resources available for customization.

How are resources accessible?

The Style Manager application works by modifying a database of resources used by various applications. When you choose colors, fonts, a keyboard, and mouse characteristics, you're simply telling the Style Manager to make the appropriate modifications to its database.

The resource database that the Style Manager edits is named *~/.dt/sessions/current/dt.resource*. If you examine it, you can see the format used to store resource settings. The designers of X decided to store the resource settings in ASCII, which simplifies reading through the resource settings and learning how to use them. Figure B shows a few lines from a *dt.resource* file.

Figure B

```
dtsession*displayResolution:     2723
dtsession*sessionLanguage:       C
Dtwm*0*ws3*backdrop*image:       Ankh
Dtwm*0*ws0*backdrop*image:       Crochet
Dtwm*0*FrontPanel*geometry:      +88-10
Dtwm*0*ws2*backdrop*image:       Toronto
Dtwm*0*ws1*backdrop*image:       Pebbles
Dtwm*0*helpResources:    \n\
Dtwm*0*initialWorkspace:         ws0
Dtwm*0*workspaceCount:  4
*0*ColorPalette:         Default.dp
*background:     #AE00B200C300
*foreground:     #000000000000
```

Resource specifications in file dt.resource *are in ASCII to make them simpler to read, understand, and change.*

The structure of this file is interesting. Each line contains a resource definition where the word on the left specifies the resource name, and the word on the right specifies the value of the resource.

The resource name is actually a path to a resource, which contains asterisks as delimiters and is terminated with a colon. When you break the word into smaller words at the asterisks, the first word is usually the application name, the last is the name of the resource being selected, and any intermediate words represent the widget hierarchy used to get there.

The CDE desktop is just another X application (named `dtwm`, for Desktop Window Manager). If you read the `man` page for `dtwm`, it's easy to interpret the line

```
Dtwm*0*ws3*backdrop*image: Ankh
```

After specifying the `dtwm` application, the next word is 0, which specifies the display. Next comes ws3, specifying the third workspace; then the name of the final widget in the list—the backdrop widget. The last word tells us the name of the resource we're modifying, namely *image*. The value to the right of the colon specifies the value, so this line is telling us that the image on the backdrop in the third workspace is *Ankh*. The other lines are interpreted similarly.

If you want to perform any really fancy customizations, you'll have to start searching through your `man` pages to do so. First, you must find out what widgets your applications are built from, then read the `man` pages on the appropriate widgets.

Please note that some widgets are containers that may hold other widgets, while some widgets are built from other types of widgets. The `dtwm` application, for example, has a container widget, named 0, that's a display containing workspace widgets (ws0...), each of which contains a backdrop.

On the other hand, if you read the `man` page for XmPushButton, you won't see a resource to set the button's caption. The reason is that an XmPushButton is built from an XmLabel, giving the XmPushButton all the capabilities of an XmLabel. The `man` page only tells you what differentiates the XmPushButton from an XmLabel. An XmLabel, in turn, is built on top of an XmPrimitive, and the `man` page only documents the behavior and resources that differentiate an XmLabel from an XmPrimitive. Likewise, an XmPrimitive is built from the Core widget.

In fact, *all* widgets are built on top of the Core widget. This is the fundamental description of a widget. You'll find that much of a widget's basic behavior is specified in the Core widget. The point is, when you want to customize a resource for a particular application, be prepared to dig around for a while, locating all the documentation on the widgets and resources you'll need.

The X files

So, you want to customize your applications. First, you must decide where to put your customizations. When you start `dtwm` on your system, it will look for resource definitions in the following files, from lowest to the highest priority:

- */usr/dt/app-defaults/$LANG/Dtwm*
- *~/Dtwm*
- *$RESOURCE_MANAGER* or *~/.Xdefaults*
- *$XENVIRONMENT* or *~/.Xdefaults-host*

The first entry, */usr/dt/app-defaults/$LANG/Dtwm*, has the lowest priority. Any later database will override the settings in this database. Since this database affects everyone's system, you don't want to change it unless you absolutely must alter the default behavior of the system for everyone. Normally, you'll want to limit your changes to your machine or that of a particular user, so you should place your

changes in one of the directories off the appropriate home directory.

Most other UNIX environments use the *~/.Xdefaults* file to store customizations. Putting your customizations here allows you to totally erase your *~/.dt/sessions/current/dt.resources* file, regenerate it, and still keep the resource settings for your applications. Please note, however, that `dtwm` uses the settings in *~/.dt/sessions/current/dt.resources* as its highest priority, so any resources you customize in *~/.Xdefaults* will be overridden if they appear there.

An example

Now, let's do a simple example. When we start the `dtterm` application, shown in Figure C, we begin customizing it immediately. We prefer to turn off the menu bar, because it can occupy a significant amount of real estate on the screen. (Also, since we can invoke a pop-up menu with all the same features, there's really no reason to keep it.)

When I began reading the `man` page for `dtterm`, I found that the resource for controlling the menu bar is named, appropriately enough, menuBar. So I added the following line to my *~/.Xdefaults* file:

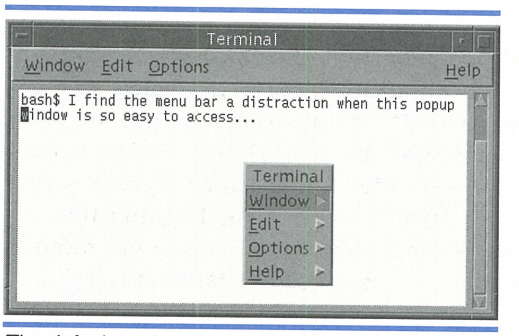
```
Dtterm*menuBar:  false
```

Now I'll never need to turn off the menu bar again. The `dtterm` comes up just the way I want, as shown in Figure D.

Conclusion

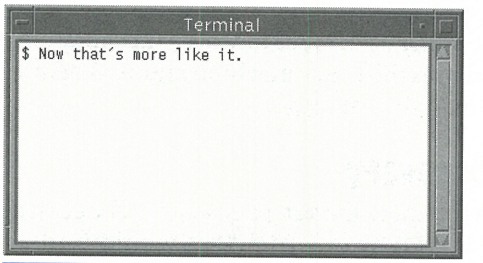
Don't be shy about customizing your X applications. Ultimately, the more comfortable you are in your environment, the more productive you're likely to be (unless, of course, you go overboard and spend all your time customizing things). This customization also pays off in other ways: You'll better understand the structure of X applications and widgets, so perhaps you'll want to create your own X applications. ❖

Figure C



The default settings for the `dtterm` application leave something to be desired.

Figure D



Now the `dtterm` operates the way I want it to.

SYSTEM CONFIGURATION

Some hidden CDE man pages

Everyone's misplaced something at one time or another. Looking for a misplaced object can lead to a frustrating search throughout the area until you find the item. On the other hand, you won't search for something if you don't know it exists!

Well, Sun seems to have misplaced the basic `man` pages for CDE. Sure, you can find the information in the help viewer, but that's not always the most convenient way. Although a GUI does make some things simpler, if you know exactly what you want, it's easier just to type in your request.

When I've tried to read the `man` page for the `dtterm`, I've been greeted with this message:

```
bash$ man dtterm
No manual entry for dtterm.
```

Then I remember that there's no `man` page for `dtterm`, and I have to use the online help viewer.

The mystery

One day, while working on a software developer's machine, I received a page from someone with a question on `dtterm`. I called up the `man` page for `dtterm`, answered the question, and went on with my work. That question led me to a good idea for a script program, so when I returned to my desk, I again called up the `man` page for `dtterm` to check out more of the details. I received this familiar refrain:

```
No manual entry for dtterm.
```

Then it dawned on me! All this time, I thought there *was* no `man` page for `dtterm`; yet I was looking at it not five minutes ago. Why could I read the `man` page at the other machine and not at my own? Just to verify the situation, I logged into the other machine and was again able to read the `man` page.

The discovery

Now that I knew the `man` page existed, I decided to find it. I knew of a trick that would find which package a file belongs to (which we describe in the article "Which Package Does This File Belong To?"), so I entered the command

```
$ grep dtterm.1 /var/sadm/install/contents
/usr/dt/share/man/man1/dtterm.1 f none 0444 bin
bin 34353 22291 811602575 SUNWdtma
```

I now know that the man page exists in the `SUNWdtma` package. But just what *is* this package? I ask Solaris with the command

```
$ pkginfo SUNWdtma
system      SUNWdtma      CDE man pages
```

I really wasn't expecting to see a package specifically marked CDE `man` pages. I'd have thought that it would be part of the basic CDE installation. I'd really expected the `man` pages to be in a more obscure location. Now the question became "Why don't I have the CDE `man` pages on my system?" After installing CDE on so many systems, I've never noticed an option for `man` pages. At this point, I'd forgotten that the computer where I found them belonged to a software developer, or I'd have realized that the package was in the developer CDE packages. Instead, I used the brute-force approach and placed the Desktop 1.1 CD-ROM into my drive and typed

```
$ cd /cdrom/cdrom0
$ find . -name "SUNWdtma" -print
./CDE/PowerPC/cde-developer/SUNWdtma
./CDE/sparc/cde-developer/SUNWdtma
./CDE/x86/cde-developer/SUNWdtma
```

That's when I noticed that the `man` pages were part of the CDE developer installation.

After digging around, I noticed that the `SUNWdtma` package contains many `man` pages that ordinary users might want: `dtmail`, `dtpad`, `dtterm`, and others. It also contains items that probably only a developer would want. However, I believe Sun misplaced this package because there's another package in the CDE developer installation that contains `man` pages, `SUNWdtmad`, which Solaris describes as

```
bash$ pkginfo SUNWdtmad
system     SUNWdtmad     CDE developer man pages
```

All the hardcore programming `man` pages reside in this package.

Installing the CDE man pages

Now comes the easy part. Since I know the `man` pages exist, I just need to install them on my machine to have access to them. Once we've mounted the Desktop 1.1 CD-ROM, we can install the `man` pages by typing:

```
$ pkgadd /cdrom/cdrom0/CDE/x86/cde-developer
SUNWdtma
```

Now that we have the `man` pages, we can read them from dumb terminals, pipe them through utilities, and use them like any other `man` page on the system. ❖

Are you a good tipper?

Do you have any great Solaris tips that you've discovered? If so, send them our way!

If we use your tip, it will appear on our weekly online ZDTips service. (Visit **www.zdtips.com** to check out all our available tip services.) It may also appear here in *Inside Solaris*. Your byline will appear with the tip, along with your E-mail and/or Web addresses.

Send your tips to **inside_solaris @ zd.com.**, fax them to "Solaris tips" at (502) 491-4200, or mail them to

Inside Solaris
The Cobb Group
9420 Bunsen Parkway
Louisville, KY 40220

Screen savers can be a bad thing

Walking through the hallways of office buildings can be amusing—many computers all running different screen savers. It's amazing how many CPU cycles are consumed by frivolity. On some computers, it's no big deal. Windows 95 computers, for example, often have fancy screen savers that reflect the fanciful tastes of their operators.

However, in a Solaris environment, screen savers can have their dark side. We don't want to discourage using screen savers, but in this article, we'll warn you about two potential problems with them.

CPU performance

One of the reasons that Windows 95 computers aren't adversely affected by screen savers is that only one person uses the computer. So the CPU cycles used by a screen saver are simply wasted if they're not drawing flying toasters.

However, many Solaris systems have multiple users and/or multiple tasks. The CPU cycles and RAM used by your screen saver aren't available to other tasks or users. If you want to run a screen saver, be aware of the impact it may have on other tasks.

If you're the only user of your computer, you have no problem. If others use it too, set the priority of your screen saver low enough so that even your low-priority background jobs have precedence over it, and your users won't be slowed down significantly.

Also, different screen savers vary in their resource requirements. Some are very demanding, while others place only a light load on the system. Choose your screen saver based on the number of users and background tasks you normally run.

Network performance

Another consequence of running a screen saver is the network bandwidth it consumes. Even though a screen saver doesn't normally consume any bandwidth by itself, many networks have workstations that are little more than X terminals. In these situations, the screen saver can be running on one computer and displaying on another. Since screen savers continually run animated sequences all over the screen, they can generate quite a bit of network traffic.

Play it safe

We came across an amusing situation about a year ago. A company had an applications server sitting in the corner, hard at work running database applications. Someone in passing noticed that the screen was ugly and was worried about screen burn-in. Rather than turning the monitor off, he activated a favorite screen saver.

Later, the system administrator got a call from one of the users of the database application: The system was acting very sluggish. Without thinking about it, the system administrator walked over to the computer, hit a key to dismiss the screen saver, and began poking around. He couldn't see any reason for the system to be slow. When the users tried out the system, sure enough, it worked just fine.

This cycle continued for a couple of days before the system administrator realized that the system was operating fine *when he looked at it*. Then he considered what he was doing, and he noticed the screen saver. When he returned it to a simple, boring screen saver, everything went back to normal. ❖

Using your boss' computer

One of my favorite cartoons shows an engineer complaining to his boss about the disparity in their computers. The boss has a shiny new SPARCstation on his desk, but only watches the screen saver, while the engineer must perform serious number crunching on a meager 80286.

This sort of situation happens frequently enough to be tragicomical. Of course, with Solaris, you're not limited to the computer on

your desktop. Instead, you can let another computer do some work for you. After all, "the network *is* the computer."

Spreading the work around... delegate

Solaris is a multitasking, multiuser, as well as network-based system. By taking advantage of all three features, you can get some complex jobs done more quickly than you could if you were limited to your own computer.

Suppose you're working late on a project, and almost everyone has gone for the day. Think of all the available CPU power on those desktops that you could put to good use! You could compile on one machine, run testing on others, and perform more tasks on still other machines, without crippling your own machine where you're editing the results.

In order to use the other computers on the network, you'll need an account on the machines you want to use. This requires that you think ahead and ask the system administrator to set up some accounts for you. On the other hand, if you're the system administrator, you can set them up yourself.

Text mode

For many purposes, you need only an account and a text-based login, and you're ready to go. You can use `telnet` to access the other computer. For example, if you want to use the machine named bugs, you could access it like this:

```
taz% telnet bugs
Trying 198.70.147.66...
Connected to bugs.warner.com
Escape character is '^]'.

UNIX¿ System V Release 4.0 (bugs)
```

Potential security loopholes

When you're in a small workgroup and trust everyone, it's often convenient to set up an */etc/hosts.equiv* file so that users on other computers can log in remotely to yours without specifying a password. This can be convenient, and it helps you avoid typing in your password dozens of times each day. Each user can also use *.rhosts* to specify other computers or specify user IDs on other computers that have the ability to log into your account without requiring a password.

For small groups, this can be fine. It makes the computers convenient to use. However, it can become a security nightmare as more and more people and hosts enter the lists. The more computers and accounts that allow unauthenticated login, the less secure your system becomes. With a foothold in your system, malicious users can more easily locate and exploit other loopholes.

If you're administering a system, limit the hosts in your */etc/hosts.equiv* directory to those that you really trust. You should also audit your system frequently for all the *.rhost* files to see who's allowing easy access to whom. Because of the possible security problems associated with */etc/hosts.equiv* and *.rhosts*, you'll want to monitor them closely on your computer. On our site, we use the script shown in Figure A to list the current settings of these files. You may want to use this script as a starting point for your own system.

Be sure to read the `man` pages on *hosts.equiv*, `rlogin`, and `rsh`, and think about the ramifications.

Figure A

```
#!/bin/ksh
# HostAudit.sh - Display the contents of the /etc/hosts
# and all the .rhosts for each active account on the
# system.

# 1) Display current value of /etc/hosts.equiv
if [ -f /etc/hosts.equiv ]; then
    echo " "
    echo "*** /etc/hosts.equiv ***"
    cat /etc/hosts.equiv
fi

# 2) Generate a list of home directories for all active
#    accounts
# 3) Sort them and remove duplicate entries
# 4) If directory has .rhosts file, display it

awk -F: '{ print $6 }' < /etc/passwd |\
sort -u |\
for ACCT in 'cat'; do
    if [ -f ${ACCT}/.rhosts ]; then
        echo " "
        echo "*** $ACCT ***"
        cat ${ACCT}/.rhosts
    fi
done
```

The HostAudit.sh script prints all the hosts and user accounts that have easy access to your computer.

Yes, they can make life simpler, but at a potential cost. By monitoring these authorization files you can keep any surprises to a minimum!

```
login: marco
Password: password
bugs%
```

Now you can use bugs as if it were your own machine—to edit, compile, debug, or whatever.

The only problem with using the `telnet` command is that you must log in each time you access the other computer. Also, if you tend to refer to information that's on the screen, you'll become annoyed with all the space that `telnet` uses when you log in.

Solaris offers an alternative to `telnet`: the commands `rlogin` and `rsh`. You can use `rlogin` in a similar fashion to `telnet`, but it won't consume as much of the screen when validating your password. You can also set up an */etc/hosts.equiv* file or an *.rhost* file in your user account(s) to prevent the need for entering passwords at all. (See the `rlogin` and `hosts.equiv man` pages for details.) If you choose to set up these files, watch for security loopholes. (See "Potential Security Loopholes" for more details.)

We went ahead and set up an *.rhosts* file in our home directory on bugs that contains the single line

```
taz
```

This tells bugs that I can log in remotely from host taz without specifying my password, because I trust taz to properly authenticate users.

Now, when we use `rlogin` to connect to bugs from taz, it looks like this:

```
taz% rlogin bugs
Last login: Tue Sep 16 11:14:22 from
taz.marcorp.com
bugs%
```

If I attempt to log in from another computer with the following, bugs prompts for a password, since other computers aren't listed in the *.rhosts* file:

```
daffy% rlogin bugs
Password:
Last login: Tue Sep 16 11:16:07 from
taz.marcorp.com
bugs%
```

Running a single command

Sometimes you need to run only a single command on another computer. You don't want the overhead of logging into the new computer, running the command, then logging back out. This is when the `rsh` command comes in handy. You tell `rsh` which command to execute and the computer to execute it on, like this:

```
taz% rsh bugs ls
a.out      hello.c
```

The first parameter, bugs, is the name of the computer where you want to execute the command. The rest of the command line is the command you want to execute, `ls` in this case. Again, `rsh` will check security, so you must specify the password to your account if you attempt to run the command from an untrusted host.

The `rsh` command then pipes your standard input stream to the remote shell's standard input stream. It also pipes the remote shell's standard output and error streams to your standard output and error streams, so the command you're executing acts largely as if it were executing on your computer. For example, if you want to copy a directory from */tmp/src* to */tmp/dest*, you might use the command

```
taz% cd /tmp/src
taz% tar cf - | (cd /tmp/dest; tar xf -)
```

Since the `rsh` command fixes the standard streams for you, you can use a similar command to copy the directory */tmp/bugsStuff* on the host bugs to the directory */tmp/bugsStuffOld* on host taz:

```
taz% mkdir /tmp/bugsStuffOld
taz% cd /tmp/bugsStuffOld
taz% rsh bugs \( cd /tmp/bugsStuff\; tar cf - \)
➥| tar xf -
```

Quick Tip: The `rsh` command has one other nifty feature. You can give it an additional name, and it'll treat its new name as shorthand notation for the name of the host on which to execute your command, thus saving you some typing. Giving it the name bugs, for example, lets you use 'bugs' as shorthand for `rsh` bugs, so listing your home directory on bugs becomes:

```
taz% bugs ls
```

To give `rsh` a new name, just create a symbolic link to the `rsh` command with the name of the host you're going to use. So, if you frequently want to access commands on the host bugs, you might create a symbolic link to bugs, as we show below. (Please note that we're assuming that the directory *~/bin* is in your path.)

```
taz% ln -s /bin/rsh ~/bin/bugs
taz% bugs ls
a.out      hello.c
```

That's all for now

You can easily let other computers help you out when your own computer isn't going to be fast enough. For normal text applications, this should be all you need. Next month, we'll show you how to use another computer when it needs access to an X server. If you have idle computers on your network, make use of them! ❖

ADMINISTRATION TRICK

Which package does this file belong to?

Have you ever needed to know to which package or packages a file belongs? If you don't know the secret, finding the package(s) can be quite tedious. First, you begin reading `man` pages about a (potentially) related topic, hoping to find the name of the file in the FILES section. If the name is there, then you quickly check the Availability section to see which package(s) it's in.

However, this technique isn't dependable: Some `man` pages omit the Availability section, some files won't be mentioned on `man` pages, and many times you just don't know any commands related to the file in question.

When you install a package, it checks for conflicting files, permissions, etc., so there must be a database of information about files in the packages. After looking around, we found the file */var/sadm/install/contents*, which is just what we need. Figure A shows a few lines from this file on our test machine.

Figure A

```
/etc/profile e etcprofile 0644 root sys 700 50375 814624099
➥SUNWcsr
/etc/protocols=./inet/protocols s none *SUNWcsr
/etc/prtconf=../usr/sbin/prtconf s none *SUNWcsr
/etc/prtvtoc=../usr/sbin/prtvtoc s none *SUNWcsr
```

The /var/sadm/install/contents file contains the information that the packaging commands use to find conflicts.

As you can see, the first column contains the name of the file or directory of interest, followed by some fields that describe the file. At the end of the line is a list of the packages associated with the file or directory. Therefore, to find the package associated with a file, you simply use `grep` to search for the file. For example, if you want to know to which package the *asy.conf* file belongs, you can search for it like this:

```
$ grep asy.conf /var/sadm/install/contents
/platform/i86pc/kernel/drv/asy.conf f none 0644
root sys 1755 16982 850956637 SUNWpsdcr
```

Here, we see that the *asy.conf* file is part of the `SUNWpsdcr` package. Now, if you want to find details about the package, you can use the `pkginfo` command, like this:

```
# pkginfo SUNWpsdcr
system      SUNWpsdcr      Platform Support,
Bus-independent Device Drivers (Root)
```

If you want more detailed information about the package, use the `-l` option to see the long description. Please keep in mind that the information is oriented towards package installation issues rather than towards description. It still contains plenty of useful information though, as shown here:

```
# pkginfo -l SUNWpsdcr
   PKGINST:  SUNWpsdcr
      NAME:  Platform Support, Bus-independent
Device Drivers (Root)
  CATEGORY:  system
      ARCH:  i386
   VERSION:  1.0.0,REV=95.10.27.15.21
   BASEDIR:  /
    VENDOR:  Sun Microsystems, Inc.
      DESC:  Platform Support, Bus-independent
             Device Drivers, (Root)
    PSTAMP:  apache970319103111
  INSTDATE:  Aug 18 1997 18:07
   HOTLINE:  Please contact your local service
             provider
    STATUS:  completely installed
     FILES:     59 installed pathnames
                12 shared pathnames
                 9 directories
                34 executables
              1157 blocks used (approx)
```

If you need still more information about the package, you can reverse your `grep` statement

to discover the other files the package installs. We'll look for the other files that the SUNWpsdcr package installs with the following statement:

```
$ grep SUNWpsdcr  /var/sadm/install/contents
/kernel d none 0755 root sys SUNWxwdv SUNWxwmod
SUNWcoff SUNWpcmem SUNWpcmci SUN
Wpcelx SUNWpcser SUNWpsdcr SUNWcsr SUNWos86r
/kernel/drv d none 0755 root sys SUNWxwdv
SUNWpcmem SUNWpcmci SUNWpcelx SUNWpsdc
r SUNWpcser SUNWcsr SUNWos86r
/kernel/drv/cmdk f none 0755 root sys 14288
47379 839648677 SUNWpsdcr
/kernel/drv/cmdk.conf f none 0644 root sys 646
54089 814617021 *SUNWpsdcr
/kernel/drv/objmgr f none 0755 root sys 5220
37069 814617020 *SUNWpsdcr
•  •  •
```

The next time you want to determine which package a file comes from, don't forget about this trick. It can save you a lot of time! ❖

SHELL-PROGRAMMING TIP

Creating infinite loops with the while command

by Alvin J. Alexander

Have you ever had to troubleshoot a problem on a Solaris system where you needed to see something in the output of the `ps -ef` command, but the process ran so quickly you could never type `ps -ef` fast enough to catch the output? I was once working on a problem with a user who was trying to log into a Solaris system through a modem. Something was wrong with the login process, and I couldn't figure it out.

By sheer coincidence, I typed `ps -ef` just as the remote user connected to the system, and I saw one process listed in the output referring to my UUCP dialer program. The last field on this line of the `ps -ef` output even showed my modem initialization string. Once I saw the command that the dialer was sending to the modem, I knew I had the wrong modem initialization string in the program. I fixed the initialization string, and the problem disappeared.

I admit that I solved that problem by sheer luck. If I hadn't typed the `ps -ef` command at just the right time, I would've never seen the dialer program running as a process—it just doesn't stay in the process table long enough for me to keep typing `ps -ef` manually and looking at the output.

A better way

As a result of this experience, I developed a new problem-solving technique. Now, any time I need to run a program that will give me information about my computer, but whose information is available only briefly, I run the program continuously in an infinite loop in another window.

So if I were to troubleshoot that UUCP problem again, I'd open two windows. In the first, I'd type this short multiline command at the Korn shell command line:

```
# while true
> do
> ps -ef | grep -i uucp
> done
```

Because the first line of the command is `while true`, this command will run forever in an infinite loop. The test never becomes false, and just like that bunny, the `while` command just keeps going and going and going. In most cases, infinite loops aren't beneficial, and we try to avoid them. In this case though, such loops are just what we need.

In order to solve the UUCP problem, I must keep my eyes open for any process dealing with UUCP, so the command I select tells Solaris to list all the processes (`ps -ef`), print the information on any of them that contain the string uucp (`grep -i uucp`), and continue doing so over and over, as fast as possible. Now I can experiment with the problem at hand, with the best possible chance of getting the necessary information. When I receive the output I need, I can kill the infinite loop by typing [Ctrl]C (or whatever interrupt key we've configured).

The pause that refreshes

You really don't want to run every command as fast as possible. Sometimes, you may want a periodic snapshot. In such a case, you can include a `sleep` command in your loop to act as a time delay. For instance, if you want a command to run every five seconds, your infinite loop would look like this:

```
$ while true
> do
> your_command_here
> sleep 5
> done
```

Since developing this procedure, I've used it to solve many other problems with modems, login processes, terminals, disk drives, and printers. However, the secret to success with the infinite loop is not the loop itself, but the logic you put inside.

Quick tip: If you haven't entered Solaris commands on multiple lines before, please note that the shell doesn't accept a line of input until it's completed. Instead of running immediately after you press [Enter] on the first line, the shell doesn't execute the `while` loop until it sees the `done` statement that completes the statement.

Whenever you press [Enter] like this and you see a different prompt, > in this case, the shell is telling you that you've not finished the command, and it's waiting for the rest of it. You can now take all the space you need to complete your command (in this case, enter all the commands you want between the `do` and `done` keywords).

Conclusion

As our computers get faster and faster, it becomes more difficult to gather the clues you need to fix a problem—the problem manifests itself in a very brief interval and is then gone. By executing your monitoring program repeatedly, you can run your test program in the foreground and have a better chance to catch any clues you need to fix that problem. ❖

Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685) 1. Publication Title: **Inside Solaris**. 2. Publication number: **0013674**. 3. Filing date: October 1, 1997. 4. Issue Frequency: **Monthly**. 5. No. of Issues Published Annually: **12**. 6: Annual Subscription Price: **$115** (**$135** Foreign). 7. Complete Mailing Address of Known Office of Publication: The Cobb Group, 9420 Bunsen Parkway, Louisville, KY 40220. 8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher (Not printer): The Cobb Group, 9420 Bunsen Parkway, Louisville, KY 40220. 9. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor: Publisher, John Jenkins, The Cobb Group, 9420 Bunsen Parkway, Louisville, KY 40220; Editor, **Marco Mason**, The Cobb Group, 9420 Bunsen Parkway, Louisville, KY 40220; Managing Editor, Michael Stephens, The Cobb Group, 9420 Bunsen Parkway, Louisville, KY 40220. 10. Owner: Ziff Davis Publishing Company, 1 Park Avenue, New York, NY 10016; Softbank Holdings Inc., 10 Langley Road, Suite 403, Newton Center, MA 02159. 11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities: None. 13. Title of Publication: **Inside Solaris**. 14. Issue Date for Circulation Data Below: November 1997. 15. Extent and Nature of Circulation—A. Total No. Copies (Net Press Run): Average No. Copies Each Issue During Preceding 12 Months, **4,309**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **5,001** B. Paid and/or Requested Circulation—1. Sales through dealers and carriers, street vendors and counter sales (Not mailed): Average No. Copies Each Issue During the Preceding 12 Months, **0**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **0**. 2. Paid or Requested Mail Subscriptions: Average No. Copies Each Issue During the Preceding 12 Months, **3,881**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **4,428**. C. Total Paid and/or Requested Circulation (sum of 15b(1) and 15b(2)): Average No. Copies Each Issue During the Preceding 12 Months, **3,881**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **4,428**. D. Free Distribution by Mail (Samples, complimentary, and other free): Average No. Copies Each Issue During the Preceding 12 Months, **41**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **44**. E. Free Distribution Outside the Mail (Carriers or other means): Average No. Copies Each Issue During the Preceding 12 Months, **0**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **0**. F. Total Free Distribution (sum of 15d and 15e): Average No. Copies Each Issue During the Preceding 12 Months, **41**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **44**. G. Total Distribution (Sum of 15c and 15f): Average No. Copies Each Issue During the Preceding 12 Months, **3,922**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **4,472**. H. Copies Not Distributed. 1. Office use, left over, unaccounted, spoiled after printing: Average No. Copies Each Issue During the Preceding 12 Months, **387**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **529**. 2. Return from News Agents: Average No. Copies Each Issue During the Preceding 12 Months, **0**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **0**. I. Total (Sum of 15g, 15h(1), and 15h(2)): Average No. Copies Each Issue During the Preceding 12 Months, **4,309**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **5,001**. Percent Paid and/or Requested Circulation (15c/15g x 100): Average No. Copies Each Issue During the Preceding 12 Months, **98.95%**; Actual No. Copies of Single Issue Published Nearest to Filing Date, **99.02%**. This Statement of Ownership will be printed in the December issue of this publication. I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including multiple damages and civil penalties). Director-Fulfillment Operations.