January 1998 • $11.50
Vol. 4 No. 1

ZIFF-DAVIS THE COBB GROUP

INSIDE SOLARIS™

Tips & techniques for users of SunSoft Solaris

IN THIS ISSUE

1
The care and feeding of zombies

4
Make Wabi simpler with these tips

6
Changing the number of processes a user may start

7
Turning off the floppy eject dialog box

8
Sun Web Server v1.0

9
Making cron work 100% of the time

10
What's the difference between floppy0 and disk1?

11
How well do you know mkdir?

12
Use find to locate unneeded files

15
Getting megabytes of free space, absolutely free!

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

The care and feeding of zombies

When you're working on a system trying to find a problem, you can use the `ps -ef` command to get a picture of all the processes running on your computer. When you use this command, you see every process your computer is trying to manage, in a listing similar to the one in Figure A. (Please note that we trimmed out many processes for the purposes of this example.) This abbreviated `ps` listing shows a zombie process, as well as some of the process hierarchy.

Now, look at the shaded line. Normally, the CMD field shows the command you run to start a process, but this one shows `<defunct>`. So just what is a `<defunct>` process? Perhaps you know it better by its nickname: It's a zombie process.

While you may never see a zombie process on many systems, they're an all-too-common sight on others. Just what is a zombie process? How are they created, and what are they doing?

Creating a process

It'll be easier to understand the details of zombie processes if you know how processes work under Solaris. Let's briefly look at how to create a process and the structure of the process hierarchy.

When you begin a program in a shell, the shell starts (or spawns) a new process to do the work. So when

Figure A

```
$ ps -ef
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0 14:44:49 ?        0:00 sched
    root     1     0  0 14:44:49 ?        0:00 /etc/init -
    root     2     0  0 14:44:49 ?        0:00 pageout
    root     3     0  0 14:44:49 ?        0:19 fsflush
    root   222     1  0 14:45:33 ?        0:00 /usr/dt/bin/dtlogin
    root   260     1  0 14:45:37 ?        0:00 /usr/lib/saf/sac -t 300
    root   135     1  0 14:45:21 ?        0:00 /usr/sbin/inetd -s
    root   264   260  0 14:45:37 ?        0:00 /usr/lib/saf/ttymon
    root   263   260  0 14:45:37 ?        0:00 /usr/lib/saf/listen tcp
   marco   390   388  0 14:47:43 pts/3    0:00 ksh
    root   388   135  0 14:47:43 ?        0:00 in.telnetd
    root   374   222  0 14:47:25 ?        0:00 /usr/dt/bin/dtlogin
   marco   456   390  0 15:11:37 pts/3    0:00 ./defunct
   marco   457   508  0                   0:00 <defunct>
    root   472   390  0 14:59:34 pts/3    0:00 ps -ef
```

You get a list such as this with the `ps-ef` command.

ZIFF-DAVIS a SOFTBANK company

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

you run your `ps -ef` command, you'll see that the process' parent is the shell you're using.

A C program creates a child process with the `fork()` function, which, at first glance, appears to operate very strangely. When you call `fork()`, only the original process calls it, but both the parent and child process return from it!

How does that work? When you call `fork()`, Solaris makes an almost identical copy of the original process. Then, both the parent and the child processes may return from the `fork()` function.

That's all well and good. However, since you typically want the child to do something other than what the parent is doing, you need a method to distinguish between the child and parent processes. For the parent process, `fork()` returns the process ID of the child; for a child process, `fork()` returns 0. Thus, when you create a process using `fork()`, it will probably look something like this:

```
if ( 0 == fork() )
   {
   /* We're in the child process */
   }
else
   {
   /* We're in the parent process */
   }
```

The process hierarchy

Every process has a unique identifier, called the Process ID (PID). When you execute the `ps -ef` command, as we did in Figure A, you'll see the PID in the second column. Each process on your system was created by another process, its parent process. That process, in turn, was created by its parent process. So it stands to reason that there must have been a first process, which is the case.

When you boot Solaris, a boot loader loads the kernel, which initializes the basic services provided by the operating system. One basic service, the swapper, is run as process 0 and named *sched*. This process then starts three other basic processes: `init`, `pageout`, and `fsflush`. The `init` process begins the other basic services your computer provides. The `init` process knows what to do by reading the */etc/default* file, which tells it which processes to start and under what circumstances.

The `init` process then spawns the processes that manage the basic system services, which, in turn, generate other processes. In Figure A, process 472 is the `ps -ef` command. Tracing its genealogy, we find that its parent was `ksh`, whose parent was `in.telnetd`. Process `in.telnetd` was spawned by `inetd`, which was generated by `init`, which `sched` started.

All processes must have a parent process. So if a process with children ends, the `init` process (process 1) inherits the children. If the process has no children, it simply dies.

Now comes the interesting part. Since you can use multiple processes in your programs, you have a facility to obtain information about your child processes when the parent processes end. Therefore, when a process ends, it gives up all its resources, except that the process-information block remains until it's read by the parent process.

Zombie processes

In fact, a zombie process is just that—a process-information block that's waiting for the parent process to clean it up. While a zombie process doesn't consume any CPU time, RAM, or I/Os, it *does* consume a process-information block—a limited resource. Thus, most professionally written programs immediately clean up after child processes when the programs terminate, if they generate any child processes at all.

In order to demonstrate zombie processes, we've created a simple program named `zombies.c`, shown in Listing A. The program creates a child process, which waits for four seconds and stops. After creating the process, the parent runs a `ps` command to show you the operating child process. Next, the parent process waits for the child process to terminate and runs the `ps` command again so you can see the zombie process in the list.

For our test, we'll compile the program with gcc and run it, like so:

```
$ gcc -o zom zombies.c
$ zom
   PID TTY      TIME CMD
  1136 pts/2    0:00 zom
  1135 pts/2    0:00 zom
  1137 pts/2    0:00 sh
  1122 pts/2    0:00 bash
   PID TTY      TIME CMD
  1136          0:00 <defunct>
  1135 pts/2    0:00 zom
  1139 pts/2    0:00 sh
  1122 pts/2    0:00 bash
```

The shaded line shows us that when the child process ended, the parent didn't clean up after it. As a result, the zombie process sticks around long after it's dead.

Cleaning up zombie processes

To clean up a zombie process, a parent simply inquires about the state of the child process. Once Solaris sees the parent do so, it removes the zombie process from the process list. Solaris can then reuse the process-information slot.

Solaris provides several functions such as wait () to inquire about the status of a child process. Thus, you can eliminate a zombie process by calling the wait() function. The wait() function suspends the parent process until a child process terminates. Fortunately, when a process terminates, it sends the signal SIGCHLD to the parent process. So, you can clean up after a child that terminates by installing a signal handler for the SIGCHLD event. In that event, you can call the wait() function, knowing that it will return immediately, because the child is already finished.

As an alternative, you could investigate one of the other wait() functions, such as wait3(). This function has an option to allow the parent process to continue if none of the child processes have terminated.

If you'll look again at Listing A, you'll notice that it contains a function called SIGCHLD_handler(), which calls the wait() function. The line that installs the signal handler is included only if you compile the program with the symbol NOZOMBIE defined. We can compile zombies.c and run it like this:

```
bash$  gcc -0 nozom -DNOZOMBIE zombies.c
bash$  nozom
   PID TTY      TIME CMD
  1212 pts/2    0:00 nozom
  1213 pts/2    0:00 sh
  1122 pts/2    0:00 bash
  1211 pts/2    0:00 nozom
   PID TTY      TIME CMD
  1215 pts/2    0:00 sh
  1122 pts/2    0:00 bash
  1211 pts/2    0:00 nozom
```

We don't see a zombie process because once it terminated, Solaris sent the SIGCHLD event to the parent process. The parent process intercepted the SIGCHLD event, called the wait() function to clean up the zombie process, then went back about its normal business.

Another word about init

OK, how does a process get cleaned up when its parent terminates before the process does? If you'll recall, the child of a process that's terminated is assigned a new parent—init. Without seeing the source code for Solaris, we can't be sure, but we believe that init follows a procedure similar to the one we've just outlined to clean up the child processes. ❖

Listing A: *zombies.c generates a child process*

```
/***********************************************
* zombies.c - zombie process demonstration    *
***********************************************/

#include <signal.h>
#include <unistd.h>

void SIGCHLD_handler(int i)
   { wait(0); }

int main(int argc, char **argv)
   {
   char cmd[50];
   sprintf(cmd,"ps -u %s",getenv("LOGNAME"));

   /* Trap the SIGCHLD event, if desired */
   #if defined(NOZOMBIE)
   signal(SIGCHLD, SIGCHLD_handler);
   #endif

   /* Create a child process */
   if ( !fork() )
      {  /* Child process returns 0 */
      sleep( 4 );
      exit(0);
      }

   /* Show the results... */
   system(cmd);     /* running */
   sleep(6);        /* let them time out */
   system(cmd);     /* terminated */
   return 0;
   }
```

Make Wabi simpler with these tips

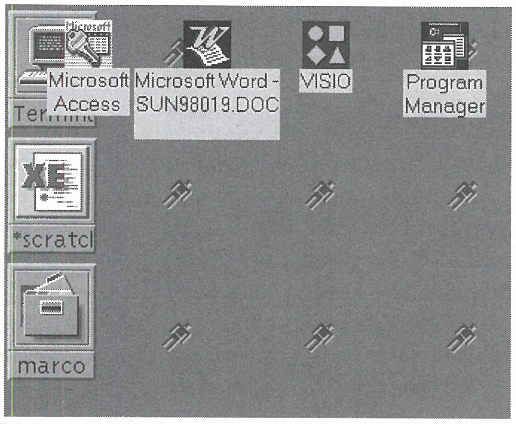
It's really too bad that Sun stopped development on Wabi. Since Wabi allows you to run many Windows programs, and the MAE permits Solaris to run Macintosh applications, Solaris has the potential for an incredibly flexible arena of applications.

Even though Sun stopped development of Wabi, it's still a very nice addition to Solaris. While Wabi doesn't handle Windows 95 applications, it does give you access to a wide range of Windows 3.11 applications, many of which are still available—and inexpensive, to boot.

Get the latest version

First and foremost, be sure you have the latest version of Wabi with the most recent patches. The latest version has some performance enhancements to make Windows applications faster, so they'll consume less CPU time.

Figure A



With the default CDE and Wabi installations, icons may clash.

At the time of this writing, the latest patch available is 103587-03, which brings Wabi up to version 2.2d. This patch also allows Wabi to operate on 24-bit displays, even though the color depth of your Windows applications will remain at 8-bit depth. Moreover, this patch allows you to run your monitor at optimal settings and retain the ability to run your applications.

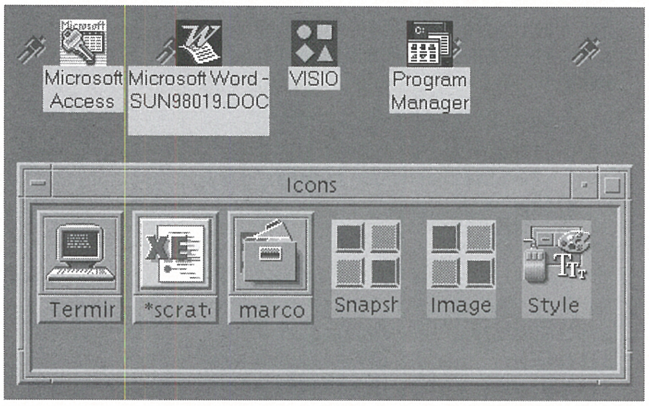
Icon collision

When you minimize programs in Wabi, they're minimized to the desktop, almost like normal CDE applications. Unfortunately, Wabi doesn't consult CDE to find icon locations that don't conflict with the CDE icons. CDE normally places the icons in a vertical column running from the left to the right, top down. Wabi, on the other hand, places them in horizontal columns top down, from left to right. So the first Wabi icon occupies basically the same location as the first CDE application, as shown in Figure A.

If you'd like to avoid the icon collision, you can simply tell CDE to place the CDE icons in an icon box. Then you can place the icon box in a location that doesn't conflict with the Wabi icons, as shown in Figure B.

Another benefit of using an icon box with CDE is that it's scrollable, so you can make the icon box fairly small. A small icon box will take less desk space, and you can still access any minimized application you want by scrolling through the box.

Figure B



An icon box with CDE allows you to move around the icon box to be out of the way.

Don't let Wabi take control of your floppy drive

For many, allowing the Wabi installation to take control of the floppy drive is a nuisance—especially if you're going to install from the hard disk drive, as we'll discuss in the next section. This way, you need not manually re-enable the Volume Manager's control of the floppy disk drive.

If you decide to allow the Volume Manager to retain control of your floppy drive, then you'll need to copy the Windows installation media onto your hard disk drive. Otherwise, you'll be unable to install the files that Wabi needs to complete the installation. (In order to avoid any copyright infringements, Wabi gets some of the files it needs from the Microsoft Windows distribution diskettes.)

Store your installation media on a hard disk

If you have an application that you're going to install for multiple users on a workstation, you can save time by placing images of the Microsoft Windows installation diskettes on your hard disk. You'll do well to place images of the installation diskettes for your other applications on your hard drive as well. This way, you can quickly install the software. You also need not keep track of the diskettes if another user wants to use Wabi on his workstation. You can just install the application for them from the hard disk drive.

Please keep in mind that you must read your license agreements carefully before you do this. You don't want to put yourself in a position where you may be violating the terms of your licenses.

When you load the disk images into the subdirectories, you need to use `volcheck` to tell the Volume Manager to mount the floppy; then copy the files from the floppy to the directory, and use `eject` to unmount the floppy disk and eject it. To simplify the process, we placed the `volcheck`, `cp`, and `eject` commands on a single line, shown in the following code, so that whenever a floppy finishes, we can just move to the appropriate subdirectory and use Bash's up-arrow to recall the long command line again for the next floppy. If you're using another shell, you can use that shell's history recall function for similar ease.

```
bash$ mkdir /dsk2/wabiapps/win311
bash$ cd /dsk2/wabiapps/win311
bash$ for J in 1 2 3 4 5 6 7 8;do mkdir
disk$J;done
bash$ cd disk1
bash$ volcheck; cp -R /floppy/floppy0 .; eject
/vol/dev/rdiskette0/disk1 can now be manually
ejected
bash$ cd ../disk2
bash$ volcheck; cp -R /floppy/floppy0 .; eject
/vol/dev/rdiskette0/disk2 can now be manually
ejected
```

Now, when you install Windows and Wabi prompts you to place the first installation disk in drive A, you can just tell it that you've placed the first diskette in the */dsk2/wabiapps/win311/disk1* directory. Then, the setup program will read the files it needs and finish the installation.

Some applications keep the diskette images separate, while other applications don't mind if you place all the files into a single directory before installation. Some applications have other rules, since naming collisions may occur between floppies.

If the installation manual doesn't contain instructions on how to put the installation media on a hard drive, you should probably first create a separate subdirectory for each floppy disk. We normally use the application name as the base directory, then name the subdirectories *disk1*, *disk2*, and so on. After you do so, you can create a directory, which we name *all*, and copy all the installation disks to it.

Once you have the media on your hard disk, it'll take only a few minutes to try a couple of experiments to determine the best way to install the application. If the application installs from the *all* directory, you can then delete the directories *disk1*, and others. If it fails, then try to install the application from the named subdirectories. If that works, delete the *all* directory.

Once we've installed the disk images, it takes only two to three minutes to install Windows on a new account, and three to four minutes to install Word 6. You'll definately notice the timesaving benefit.

File naming

When you're creating your application directories, be sure to use short, lowercase names. If you use any uppercase characters, or if you use a name that's too long, the system will "mangle" the names to make them acceptable for Windows. If you want names that are simple to type, then keep them short and lowercase.

Another handy trick is to keep the paths short. This way, when you're installing many applications, you need not type so much. If you can keep the paths short, you're way ahead of the game. In our case, because of the way we lay out our file systems and organize our files, we placed Windows at */dsk2/wabiapps/win311*. Since that's not a terribly convenient name to type, we created a symbolic link in the root directory named *win3* to point to the */dsk2/wabiapps/win311* directory. In order to specify the location of the first diskette, we need only enter */win3/disk1*.

Conclusion

Following these tips may save you some time. Now you need not track down those elusive installation floppies or remember to enable the floppy drive for Solaris after installing Wabi. Hopefully, you can get the user going in a few minutes, so you can get back to the myriad other tasks you must do each day. ❖

Changing the number of processes a user may start

Sometimes when your system gets incredibly slow, you might run a `ps -ef` and notice that a particular user has dozens of processes running. The user may simply be running several repetitive tasks, or she's written a shell script to use background processes. Perhaps a software developer has written a multithreaded program that rambled off into the weeds.

Solaris has an upper limit on the number of concurrent processes it allows on the system. (The default value seems to be about 1,000 on the machine we tested.) So if all the processes are in use by an errant program or user, you won't be able to kill them—after all, each program you run must start a new process to do so! About the best you can hope for is that some processes will expire from lack of resources so you can start killing the others.

Listing A: *num_procs.c*

```
/***********************************************
** num_procs.c - Create as many processes as **
** we can.                                   **
***********************************************/

#include <sys/errno.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char **argv)
   {
   int i=0, iLastProc = 0;

   /* Create a lot of processes */
   while ((iLastProc<100000) && (i!=-1) )
      {
      i = fork();
      if (i==0)
         { /* Child process */
         sleep(180);
         exit(0);
         }
      else if (i!=-1)
         /* Normal process start */
         iLastProc++;
      }

   if (i==-1)
      perror("num_procs");
   printf("Started %d processes.\n",
iLastProc);
   }
```

To illustrate, we've created a program named `num_procs` that will create as many processes as possible before it stops. The source code for `num_procs` is shown in Listing A.

If you compile and run num_procs, you should see something like this:

```
bash$ gcc -o num_procs num_procs.c
bash$ num_procs
num_zombies: Resource temporarily unavailable
Started 947 processes.
```

Once `num_procs` starts all the processes it can, it exits, which frees up exactly one process slot. If you run the `ps -ef` command, you'll see all the usual suspects, as well as a lot of processes named `num_procs`. Executing any normal command that uses a single process will operate just fine.

But the instant you execute a statement that requires more than one process, you'll have a problem. Your computer may appear to hang, or you'll get an error message.

Luckily, all the processes generated by our `num_procs` command live for only three minutes. Three minutes after you run `num_procs`, all the processes terminate, and your system will run normally once again.

It's bad enough that a user can create enough processes to hinder her own work. Even worse is the fact that the user can hinder *everyone* on the system. That's because Solaris, out of the box, allows a user to start any number of processes, up to the maximum number the system allows (if you'll recall, we mentioned that could be up to 1,000).

Fortunately, Solaris provides a method for you to change the maximum number of processes the system supports, as well as the number of processes any user may start. You can set these limits by changing the */etc/system* file. When it boots, Solaris reads this file to help it configure the system.

We care about two variables: `max_nprocs`, which specifies the number of processes for the system, and `maxuproc`, which sets the maximum number of processes any user may start. The following syntax sets these variables:

```
set variable=value
```

So if you want to limit the system to 500 processes and each user to 50, add the following lines to your computers */etc/system* file:

```
* Maximum # of processes for the system
set max_nprocs = 500

* Maximum # processes per user
set maxuproc = 50
```

Please note that any comments in the */etc/system* file must have an asterisk (*) at the beginning of the line. Also, remember that Solaris reads */etc/system* only when it boots up, so you'll have to reboot your computer to implement your changes.

When a user attempts to run too many processes, this simple change should affect only that user's system. However, when enough users hog multiple processes, they can still prevent you from having any processes left over for normal system maintenance—such as killing errant processes.

Conclusion

For many systems, you'll never need to change these process limits. However, spending a few minutes now to prevent a user from usurping all the processes on the machine may save you a few frustrating hours later.

Since many users may need a significant number of active processes, you don't want to be too restrictive. Your goal is to prevent a user from accidentally locking everyone out of the system—not to limit your users. So for our system, we retained the default maximum number of processes on the system (about 1,000) and set the user limit to 500. ❖

CONFIGURATION TIP

Turning off the floppy eject dialog box

As Solaris users, we know we have it good. Solaris provides many facilities to make our lives easier in one way or another. The Volume Manager, in particular, really simplifies the use of floppy disks and CD-ROMs.

With many other versions of UNIX, changing floppies isn't nearly so simple. If you're going to mount a floppy—or other removable device—you need to know the media type in advance, then issue the appropriate mount statement in order to mount the file system on the media.

Full autopilot

In the ideal situation, both your CD-ROM and floppy drive support automatic ejection *and* detect media insertion. Whenever you insert a floppy or CD-ROM drive, the Volume Manager detects it and examines the media to determine the type of file system it contains, then mounts it. Similarly, when you're finished with the media, you use the `eject` command, which writes any pending data to the media, unmounts the file system, and ejects the media.

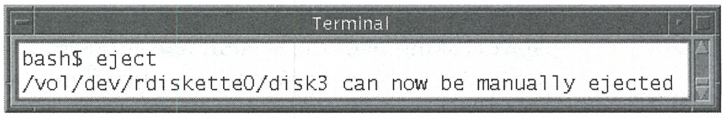
Some intervention required

However, some drives don't support automatic media detection. If the drive doesn't detect new media automatically, you'll need to run the `volcheck` command to tell the Volume Manager to check whether the drive(s) have new media. Normally, only floppy drives lack this ability.

Similarly, some floppy drives have no hardware to physically eject the media. In this case, the `eject` command does everything it can, short of ejecting the media. It still completes any pending writes and unmounts the file system. Then, instead of ejecting the media, it tells you that the drive is ready to be manually ejected.

When you use `eject` to tell the Volume Manager from the command line that you're finished with the diskette in a particular floppy drive, the Volume Manager tells you when it may be removed, as shown in Figure A. The Volume Manager also presents a pop-up dialog box to tell you when it's safe to remove the floppy, as shown in Figure B on page 8.

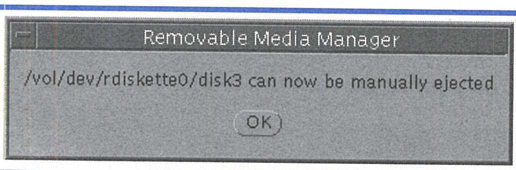
Figure A



```
bash$ eject
/vol/dev/rdiskette0/disk3 can now be manually ejected
```

The Volume Manager tells you when it's safe to remove the floppy after you use the `eject` command.

Figure B


Removable Media Manager

/vol/dev/rdiskette0/disk3 can now be manually ejected

This pop-up dialog box also tells you when it's safe to remove the floppy after using the `eject` command.

The fly in the ointment

Sometimes this pop up is inconvenient. The OK button isn't selected by default, so you can't just press the [Enter] key to dismiss it. Also, it doesn't have a hotkey you can use to dismiss it. The only simple keyboard combination you can use to dismiss the pop up is [Alt][F4]. Otherwise, you must grab the mouse, find the OK button, and click it.

The situation can be even worse when you've got several machines in proximity, and you're using a single workstation, telnetting into another. Unless you explicitly set the `DISPLAY` variable in your telnet session to point to your display, the pop-up message appears on the default display on the computer. Since the `eject` command already tells you when it's safe to eject the floppy, the ideal would be if we could just turn off the dialog box.

The solution

Turning off the dialog box is actually rather simple. The program that displays the dialog box is in the directory */usr/lib/vold*, and its name is `eject_popup`. Just rename the program to something else, say `eject_popup.old`, then the Volume Manager won't be able to find and execute it. We've used this technique before, and it's never caused us a problem.

However, some people wonder what may be going on inside the Volume Manager when it can't find the `eject_popup` program. For those of you who are a little apprehensive, you can always create a fake version of `eject_popup` that the Volume Manager will be able to find and execute. You just need a suitable program that won't do anything destructive. As it happens, the */usr/bin/true* program fits the bill perfectly. So you can simply create a link to */usr/bin/true* named `eject_popup` in the */usr/lib/vold* directory, like this:

```
Devo% su
# cd /usr/lib/vold
# mv eject_popup eject_popup.old
# ln -s /usr/bin/true eject_popup
```

Since we have retained the old version of `eject_popup`, we can always restore the original behavior if we want.

Conclusion

You may find the floppy disk's eject pop-up dialog box too intrusive in some situations and downright annoying in others. If the floppy disk's eject pop-up dialog box is in your way, you now know how to get rid of it. ❖

COMMUNICATIONS

Sun Web Server v1.0

If you want to try using a Web server to publish internal documents, you really owe it to yourself to get a copy of the Sun Web Server. It's easy to install and administer. Best of all, if you have a copy of Solaris 2.5.1 or Solaris 2.6, you already have a license to use it.

Simply point your Web browser to the address *www.sun.com/webserver/index.html* to start looking over the system requirements and feature lists. Since it's a new product, Sun's Web Server supports some new features, such as HTTP 1.1 support, and all the important features found on other Web browsers.

It comes in standard package format, so it's simple to install. This gives you the time to experiment with content, instead of worrying about how to set up and configure it. You administer it with a standard Web browser, and all the documentation is in HTML. What are you waiting for? Download it and try it out! ❖

Making cron work 100% of the time

While I was writing the article "A Simple Way to Synchronize the Computers on Your Network," I ran into a difficulty with the cron command. I wanted to execute the command:

```
date `rsh Test -l test date +'%y%m%d%H%M.%s'`
```

The response I'd get would be an E-mail from cron telling me about a problem:

```
To: marco
Subject: Output from "cron" command
Content-Length: 80
Your "Cron" job

date +'

produced the following output:
```

The problem with the % symbol

After some experimentation and reading the man pages for date and cron, I discovered this about the *crontab* file: "The sixth field of a line in a *crontab* file is a string that's executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a NEWLINE character."

When I dutifully preceded the percent symbols in the command line, I got better—but still useless—results. This time cron mailed me this message:

```
date: bad conversion
```

At this point, just to get the project going, I put the command in a script file and synchronized the computers. After I solved the problem for the software development team, I began working to discover why cron was acting so strangely.

I discovered that the reason for this behavior is a minor bug in the interaction between cron and quoted strings. The % symbols in the *crontab* file are intended to represent a newline character, so you can pipe input to commands that cron will execute. Whoever designed and wrote cron envisioned the need to actually use a literal % character. Thus, you were allowed to use a backslash to quote the % character.

So if cron encounters a % after a \, it knows to discard the \ and keep the %. However, if the % character happens to be inside a quoted string, then cron keeps *both* the \ and % characters. Unfortunately, retaining both mangles the output of the date command executed on the server. As a result, the client date command tries to interpret a string like \07:\33:\22, which results in the reported date conversion error.

In this case, we have two simple workarounds. The first is the one we mentioned previously: We created a script file to do the job, and our cron job simply executes the script file. The second workaround is to omit the quotes. Normally, you use quotes to prevent the shell from breaking a string into multiple parts at spaces, and to prevent the shell from interpreting any special characters. Since our format string doesn't contain any spaces and contains no special characters, we were able to omit the quotes. However, you won't always be able to successfully omit the quotes. So, be prepared to use a shell script, if necessary.

Using % successfully

Now that we've stumbled onto the special characteristics of the % symbol, let's try it out. When cron runs a job, it starts a shell and begins sending the characters to the shell as if they were typed. Each time it encounters a % symbol, it instead sends a newline, (unless the % symbol is escaped by a preceding \, as described previously).

Therefore, if you have a command that expects information from the standard input stream, you can simply append % to your command. Doing so executes the command and starts the next line of input. At this point, enter the data you want to give to the program, using a % wherever you'd normally use a newline.

Here's a contrived example: We'll use the cat command to add two lines of data (*abc* and *def*) to a file called */tmp/junk* five minutes past every hour. To do so, our crontab entry looks like this:

```
5 * * * * cat >>/tmp/junk%abc%def
```

Editing a crontab

The safest way to edit your list of cron jobs is to use the crontab -e command. When you do so, the crontab command creates a temporary file and invokes the editor for you. Then, when you finish your changes and exit the editor, crontab checks the validity of each line. If it finds a problem, it will echo the line to the screen and issue an error message, like so:

```
$ crontab -e
Bad line in crontab
crontab: errors on previous line; unexpected
➥character found on line.
crontab: errors detected in input, no crontab
➥file generated.
```

Here, we entered the text *Bad line in crontab* into the editor, then we saved and quit. The `crontab` command echoed this line back to us to show which line failed, then printed its error messages.

Clearing a crontab

If you want to clear your `crontab` and you want to use `crontab -e`, be aware that you can't just erase all the lines in the file. If you do so, `crontab` emits the message:

```
crontab: temporary file empty
```

and it won't change your `cron` jobs. Since the message doesn't tell you that it's not going to change your `cron` jobs, you'll get a nasty surprise when your next `cron` job executes. You can avoid this surprise by putting a comment in your `crontab`. This way, although the `crontab` file won't be empty, it won't do anything destructive either.

Conclusion

It often amuses me how one discovery leads directly into another. Because of my problem with `cron`, I discovered a feature I didn't even know existed. While investigating the problem, I used a test account. When finished, I erased the `cron` job. Or so I thought. The next time I logged into the test account, I was swamped with mail from `cron` complaining about the problem with the `cron` job. ❖

Quick Tip

If you use the `crontab -e` command to edit your `cron` table, be aware that it'll use the editor specified by the `EDITOR` variable to perform your edits. So make sure that you set `EDITOR` to your favorite editor, or you may have to struggle with an editor you aren't proficient in or don't care for. If you're using CDE, the default setting for `EDITOR` is */usr/dt/bin/dtpad*.

QUICK TRICK

What's the difference between floppy0 and disk1?

Have you ever noticed that when you load a new floppy disk, two directories appear in the */floppy* directory? One always seems to be named *floppy0*; the other is often unnamed, *disk1*, or some other designation.

Whenever you load media into a removable drive that the Volume Manager controls, the Volume Manager mounts the media under two names. One of these is the actual name of the media (i.e., its disk label, if it has one), and one is a symbolic link, as shown in Figure A.

The symbolic link is always the media type with a suffix of the drive number for the media. So, the first floppy drive is always floppy0, and the second is floppy1. Similarly, the first CD-ROM is cdrom0, the second is cdrom1, etc.

This naming scheme is a convenience that the Volume Manager offers. Sometimes you don't care about the actual media name, and sometimes you do. When you don't care what the media name is, such as when you're copying a file to a floppy for your boss,

Figure A

```
bash$ ls -al /floppy
total 20
drwxr-xr-x   3 root     nobody       512 Oct  8 07:22 .
drwxr-xr-x  27 root     root        1024 Oct  7 13:04 ..
drwxrwxrwx   1 root     other       7168 Dec 31  1969 boot
lrwxrwxrwx   1 root     nobody         6 Oct  8 07:22 floppy0 -> ./boot
```

The Volume Manager mounts removable media under two names.

you don't really need to know the floppy disk's name. Thus, you can use the generic name of the floppy, like this:

```
bash$ volcheck
bash$ cp file /floppy/floppy0
bash$ eject floppy0
/vol/dev/rdiskette0/boot can now be manually
ejected
```

At other times, however, you want to be certain that a particular volume is mounted before you proceed. This happens often in shell scripts that access a particular CD-ROM. In this case, your shell script can check for the existence of the CD-ROM by examining the */cdrom* directory for a directory of the required name, like this:

```
if [ ! -d /cdrom/MyCD ]; then
   echo "Please mount MyCD and try again"
   exit
fi
```

Conclusion

If you're always copying files to and from floppies for people, then by all means ignore the label name and use the generic name. It's much simpler. But keep in mind that when you must deal with specific files on specific volumes, checking or using the actual volume name will help prevent costly mistakes. ❖

QUICK TIP

How well do you know mkdir?

Everyone knows how the `mkdir` command works, right? You just type

```
# mkdir zebra
```

and you create the subdirectory *zebra*. What could be simpler? Many people think that's as far as it goes, and they don't learn the other options. After all, who reads the `man` pages for the commands he already knows?

But the `mkdir` command has two options that many overlook: First, it can create multiple directories at once. It also allows you to specify the permissions on the directories you create.

Creating multiple directories

Suppose you want to create the directories *abc*, *abc/def*, *abc/ghi*, and *xyz*. You can do so with a single `mkdir` command:

```
# mkdir -p abc/def abc/ghi xyz
```

Hmm. I set out to create four directories, but only specified three. Did I miss one? No, not only can you specify multiple directories on the same command line, but you can also specify the `-p` option which tells `mkdir` to create any parent directories required. So when `mkdir` tried to create *abc/def* and noticed that *abc* didn't exist, it also created *abc* before creating the *def* subdirectory.

Specifying permissions

Another `mkdir` command option allows you to specify the permissions on your directories as you create them. This feature can be very handy when you want to give people access to some files, but don't want to open up your entire directory tree to them. You can create a directory and limit access to it in one command.

For example, you may want to create a directory to distribute some information to the other members of your group, but you don't want anyone outside the group to have access to it. To do this, you could always create the directory, then use `chmod` to alter the permissions. But it would be even simpler to specify the permissions as you specify the directory name, like this:

```
# mkdir -m 750 distrib
```

With this one command, you've created the *distrib* subdirectory with the desired permissions. You, of course, get full access, members of your specified group may access and read the directory, and no one else may access it. ❖

Using find to locate unneeded files

by David S. Herron

A continuing problem with any computer system is the lack of free space on critical file systems. It seems as though there's never enough—and where did all these files come from anyway? While this article doesn't offer the ultimate solution to this problem, it does offer a tool to help manage one aspect of the problem—detecting and deleting junk files littering your file systems.

What are junk files?

There are many sources for junk files: applications or processes that dump core, temporary files from programs that are no longer running, backup copies of files, and so forth. These files take up space, and, while they might be useful for a short time, they're usually a waste of disk space. Even as disk drive prices drop, available disk space is always at a premium.

The fundamental problem is that these files *do* have a useful lifetime, which is why they're being generated. After all, you wouldn't let your programs generate core files if you weren't going to look at them, would you? (See "Stopping Core Dumps" for more details.) However, these files quickly lose their value, but people are often too busy to delete these files when they no longer have value. Either they forget about them, in the case of backup copies of files, or they never knew about them, such as when some programs generate temporary files that aren't cleaned up.

Find these obsolete files

The basic technique that we'll present here is to use `find` to locate the files we wish to delete, then delete them. In this article, we'll lay out the foundation for demonstrating the technique. Next month, we'll build a fancy shell script to do all the work.

The first problem is to identify which files are obsolete and may be safely deleted. Some files are easy: How useful is a *core* file after it hasn't been touched in a week? It's a safe bet that we may safely remove it.

But how long is an `emacs` backup file? Once we get past the obvious few cases, it turns out that other files depend on the site. Backup files here have a limited useful lifetime since we perform nightly backups. We can always retrieve the original document from tape, if we must. So in our case, backups more than a couple of days old are safe to delete.

Complex search clauses for find

We've used the `find` command to locate files in various articles, so it should come as no surprise that we'll use it here. However, so far we haven't used `find` with complex search clauses.

Let's say that the host we're working on is for the software development team, and they use `emacs` for editing. After some thought and digging around on the file system, we find that we want to delete these files from the */home* file system:

- Any core file that hasn't been accessed in three days.
- Any file named *a.out* not accessed in the last week.
- Any file named *junk* or *ttt* that hasn't been accessed in five days.
- Any backup file (*~ for emacs, and *.bak for some other applications) not accessed in the last four days.

We could locate all the files we want to delete with the six commands shown here in Figure A. However, we don't really want to do it this way. It's far too resource-intensive. Not only does the process take too much time, but we wind up going over the file system six times instead of only once, thus increasing the number of I/O operations, RAM usage, and the number of processes.

Figure A

```
$ find /home -name core -atime +3 -print
$ find /home -name a.out -atime +7 -print
$ find /home -name junk -atime +5 -print
$ find /home -name ttt -atime +5 -print
$ find /home -name '*~' -atime +4 -print
$ find /home -name '*.bak' -atime +4 -print
```

These six statements will find and delete the files we want, but at too high a cost.

The `find` command allows us to specify a complex expression that describes the files we want to locate. So, we'd be better off learning how to create these more complex `find` statements. Not only would it help with the current task at hand, but we'll find it useful in other system administration tasks as well. You're probably aware that when you string clauses together with `find`, the `find` command treats them as if the clauses had the word AND between them.

That's why the six commands we showed you will work: Not only must they satisfy the name requirement, but they must also satisfy the access time restriction (`-atime`), or `find` won't print the filename. For example, the clause `-name core -atime +3` specifies files whose names match core and which were last accessed more than five days ago.

The `find` command allows us to specify a choice with the `-o` (or) option, negation with the `!` (not) option, and grouping with parentheses `( )`. Using these operators, we can rewrite our six statements as a single statement, as shown in Figure B.

Figure B

```
$ find / \( \( -name core -atime +3 \) \
>         -o \( -name a.out -atime +7 \) \
>         -o \( -name junk -atime +5 \) \
>         -o \( -name ttt  -atime +5 \) \
>         -o \( -name '*~' -atime +4 \) \
>         -o \( -name '*.bak' -atime +4 \) \) \
>         -print
```

This `find` statement will do the same job, but much more efficiently.

Here we need the parentheses we want to specify more than just the filenames: In this case, we also want to match the file's name and age. Without parentheses, the expression would run together like "find files whose name is core and last accessed more than three days ago or name is a.out and last accessed more than seven days ago or ..." and it wouldn't be clear exactly what the or clauses referred to. With parentheses, the statement is clear: "... (name is core and was last accessed more than three days ago) or (name is a.out and last accessed more than seven days ago) or ...".

We must write the parentheses as `\(` and `\)` because parentheses are special characters to the shell (i.e., they're metacharacters). If we don't escape them, the shell would interpret them, and `find` would never see them. This way, the shell ignores them, and `find` does see them. If we're going to use the negation operator (`!`), we should be aware that the C shell uses the exclamation mark for its command-line history mechanism, so we'll want to quote that character as well.

Once we start playing with complex file-selection criteria for the `find` command, we may want to look more closely at the palette of commands that `find` offers, so we can see what we can do with them. Some of the more useful `find` options are shown here in Table A.

The `find` command has many more options, so be sure to consult the `man` page for them. This should give you a flavor of what `find` can accomplish. It's worthwhile to spend an hour or two investigating the flexibility of the `find` command.

Table A: *Useful find options*

Find Option	Description
`-depth`	Performs a depth-first search by searching subdirectories before processing the parent directory.
`-mount , -xdev`	Restricts the search to the current file system.
`-local`	Excludes remote file systems (as defined in */etc/dfs/fstypes*).
`-fstype type`	Only include file systems of type `type`.
`-newer file`	Only show files newer than the file `file`.
`-atime n, -ctime n, -mtime n`	Compare today's date with the last access time (`-atime`) of the file, the creation date (`-ctime`), or the last date the file was modified (`-mtime`).
`-gid n`	Yields true if the file's group ID matches `n`.
`-group groupname`	Yields true if the file's group name matches `groupname`.
`-uid n`	Yields true if the file's owner's user ID matches `n`.
`-user username`	Yields true if the file's owner's name matches `username`.
`-nouser, -nogroup`	Yields true when the file is missing a valid user ID (i.e., user doesn't exist in */etc/passwd*) or group ID (as found in */etc/group*).
`-type c`	Yields true if the file is of the specified type: (`d`=directory, `f`=file, `s`=symbolic link, etc.).

Deleting the files

Once we identify a list of files to delete, all that's needed is to delete them. While we could use `find`'s `-exec` option to invoke `rm` to delete the files, it starts a new process for each command, which can consume quite a bit of CPU and I/O time. Instead, we use `xargs` to gather all the filenames and pass them at one time to the `rm` command, as shown in Figure C, so we run only three commands once each: `find`, `xargs`, and `rm`, rather than the `find` process and one `rm` command per file to delete.

Figure C

```
$ find / -name core -print ¦ xargs rm -rf
```

We pass filenames one at a time to `rm`.

The `xargs` command collects all the words it finds on its standard input and saves them. Once the standard input is empty, it starts the specified command (with argument list) and adds the list of words it collected to the end of the argument list. So the `find` statement in Figure C locates all files named *core* and sends them to `xargs`. Once `xargs` gets the names of all *core* files, it starts the command `rm -rf` followed by the entire list of commands to delete.

Together `find` and `xargs` make for a very flexible tool. There are a number of options to tune their performance to the vagaries of the exact command you want to use: You get to do the job, and save time, RAM, and I/O operations.

Putting it all together

What we've shown you so far is how to use `find`, `xargs`, and `rm` together for the purpose of cleaning away old files. Now we'll show you a complete example.

First we must determine, for each file system that we're responsible for, which files are to be treated as disposable junk. What's considered a junk file varies, depending on the file system.

As we said, when developing the statement in Figure B, we tailored it to the */home* directory of a machine used for software development. Each set of tools has its own characteristic junk files that are left behind (e.g., `emacs` leaves backup files with a ~ at the end). We must think about how we use every file system on each computer, and tailor our `find` statement accordingly. You'll probably want to do this for each host computer separately because you don't want to run `find` running across NFS-mounted partitions.

Let's say we must clean up the file systems */*, */usr*, and */home*. Each has a separate list of files to take care of, and we don't want to have the `find` that starts at / to wander into */usr* or */home*. We also don't want `find` to cross over into NFS-mounted partitions. So we might make a shell script, like `cleanFS`, shown in Figure D.

Figure D

```
#!/bin/sh
#
# cleanFS -- remove unwanted files from the
# /, /usr, and /home file systems
#
find / -xdev -name core -print ¦ xargs rm
find /usr -xdev -name core -print ¦ xargs rm
find /home -xdev \( \( -name core -atime +5 \) \
             -o \( -name '*~' -atime +10 \)    \
             -o \( -name '*.bak' -atime +10 \) \
             -o \( -name junk -atime +3 \) \)  \
      -print ¦ xargs rm
```

The cleanFS script removes typical useless files from your file system.

Once we've tested the `cleanFS` script, we can safely automate it. The more you automate things, the less you have to worry about. The

Stopping core dumps

Core dump files can take a significant amount of disk space, especially if you don't notice them, and they start to litter your hard drive. If you never intend to examine these core files, you can modify your shell's startup file to prevent any program you run from generating a core dump.

If you're running in the C shell, just add the command

```
Devo% limit coredumpsize 0
```

Bourne and Korn shell users can achieve the same effect with this command:

```
$ ulimit -c 0
```

In either case, you're setting the size limit of the core dump file to 0 blocks, preventing the program from generating any core dump file at all. If you don't want any of your system processes to generate core files either, be sure to modify the *.profile* file for the root account as well.

simple way to automate this task is to have cron execute this script every day at, say, 3:00 a.m. You can do this by starting a root shell and running the command:

```
# EDITOR=vi crontab -e
```

This tells crontab to use vi as your editor and to load the current cron schedule into vi for editing. Now, add the following line of text to the end of the current cron schedule:

```
* 3 * * * /usr/jrnl/bin/cleanFS
```

Finally, save the file and quit vi. Now you'll run the cleanFS script every morning at 3:00. Please note that we're assuming here that you place scripts from the journal in a directory named */usr/jrnl/bin*.

Conclusion

It's very important to remove useless files from your hard drives. Although hard drives are becoming cheaper every day, you still don't want to shut down your computer to add hard drives, do you? While shell scripts using find and xargs are very flexible, the result isn't simple to read or maintain. Since disk space management is such an important topic, we'll build a shell script that will use a configuration file to tailor which files will automatically be deleted, simplifying this laborious task. Look for the script in an upcoming isssue. ❖

DISK ADMINISTRATION

Getting megabytes of free space, absolutely free!

It happened again. You just popped in a new 9GB hard disk and just a few weeks later, it's 95 percent full. Would you like to make another 800MB appear on that disk? You can do it! The trick is to know that when you initialize a file system with newfs using the default parameters, Solaris sets aside some of the space for working space (inode maintenance, etc.) The newfs command uses the minfree value to tell it how much space to reserve—10 percent by default.

However, the default value of 10 percent was decided a long time ago—when hard disks were much smaller and partitions didn't spread across multiple disks. While Solaris needs some disk space to manage the system, 10 percent of an 8121MB disk tallies up to 812MB—an expensive price to pay. In fact, it's too large a price! On any large partition, a 1 percent minfree value is more than enough to do the job.

Change your file system

You can change the amount of free space when you create the file system (with the -m parameter for newfs) or afterwards by using the tunefs command, as shown in Figure A.

Figure A

```
challenger{root}: umount /home1
challenger{root}: tunefs -m 1 /dev/md/rdsk/d0
minimum percentage of free space changes from
10% to 1%
should optimize for space with minfree < 10%
challenger{root}: mount /home1
```

It takes but a moment to change the minfree parameter on a file system.

After you unmount the partition with the umount command, it takes only a second for tunefs to change the minimum free space parameter. Then you can mount the partition again and begin using it immediately. Please note that when you change the free space on your file system using tunefs, you can safely ignore the message about optimizing for space.

Figure B on page 16 shows the difference between two identical 8121MB drives. The */home1* file system has a minfree value of 10 percent, while */home2* is set to 1 percent. The same is true with the two identical 25GB partitions */home3*, and */home4*.

SunSoft Technical Support

(800) 786-7638

PERIODICALS MAIL

147831 06/98

[redacted]

Please include account number from label with any correspondence.

Figure B

```
challenger{root}: df -k
Filesystem          kbytes   used    avail capacity  Mounted on
/dev/dsk/c0t4d0s7   8316189     9  7484570     1%    /home1
/dev/dsk/c0t5d0s7   8316189     9  8233019     1%    /home2
/dev/md/dsk/d0     24943121     9 22448802     1%    /home3
/dev/md/dsk/d1     24943121     9 24693681     1%    /home4
```

The /home2 and /home4 file systems have more space available to you than do their counterparts.

Figure C

```
#!/bin/csh -f
# Show the minfree value of each local partition

echo "Partition minfree     lost           total"

foreach PARTITION ('df -k -F ufs¦grep -v Filesystem¦awk '{print
$6}'')

   set USED='df -k $PARTITION¦grep $PARTITION¦awk '{print $3}''
   set AVAIL='df -k $PARTITION¦grep $PARTITION¦awk '{print $4}''
   set TOT='df -k $PARTITION¦grep $PARTITION¦awk '{print $2}''
   set LOST='echo $TOT $USED $AVAIL¦awk '{print $1-($2+$3)}''
   set MINFREE='echo $LOST $TOT¦awk '{printf "%3.0f", $1*100/
$2}''

   echo " " \
   ¦ awk '{printf "%-10s%3.0f%%\t  %5.0f MB out of %6.0f MB\n",
partition, minfree, lost/1024, tot/1024}' \
     partition=$PARTITION minfree=$MINFREE lost=$LOST tot=$TOT
end
```

This `showminfree` script will show you an approximation of minfree on all your file systems.

Figure D

```
challenger{root}: showminfree
Partition minfree      lost            total
/           10%      5 MB out of      47 MB
/usr        10%     32 MB out of     325 MB
/var        10%     33 MB out of     329 MB
/export     10%      8 MB out of      82 MB
/opt         5%     95 MB out of    1907 MB
/home1      10%    812 MB out of    8121 MB
/home2       1%     81 MB out of    8121 MB
/home3      10%   2436 MB out of   24359 MB
/home4       1%    244 MB out of   24359 MB
/home5       5%     95 MB out of    1907 MB
/home6       1%     80 MB out of    8030 MB
/home7       5%     90 MB out of    1791 MB
```

The resulting script shows the size of the reserved area for each file system, as well as the minfree parameter.

How do you know the value of `minfree`? Solaris doesn't have any command to show which `minfree` value is used on any given partition. However, you can determine `minfree` by doing some math with the values `df` presents. Figure C shows the `showminfree` script that displays the approximate value of `minfree` for all file systems on your host.

We built the script by using `df -k` to generate the numbers we need. If you add the amount used and the amount available for use, you'll find that you always come up short of the total amount. This difference is the space set aside by Solaris as dictated by the `minfree` value. So we calculate the percentage by dividing the amount reserved by the total size of the file system. Figure D shows an example of the output from our `showminfree` script.

Conclusion

When you have large file systems on your machine, you owe it to yourself to eke out some extra space on your hard drives. After all, you've paid for those hard disks, and Solaris isn't using nearly so much as the default value of `minfree` reserves. ❖

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