May 1996 • VOL. 2 NO. 5
US $11.50

INSIDE™ SOLARIS™
Tips & techniques for users of SunSoft Solaris

in this issue

An introduction to awk

By August Mohr and Marco C. Mason

The awk language is a powerful part of the UNIX system. This language allows you to do many things that would otherwise require you to write a C program. In fact, it's often used for prototyping programs to test ideas before converting them to C or another compiled language. If you're wondering about the funky name, that's because it's named for its authors: Aho, Weinberger, and Kernighan.

If you've never used awk, this article may inspire you to try it. If you've used only its default output capabilities, this article gives a starting point for taking fuller advantage of this powerful tool. Be sure to read the man page for more information on awk's capabilities.

Despite the power, awk programs can be amazingly simple—especially when compared with the equivalent C programs. This simplicity comes in part from its ability to make assumptions about the format of its input that a generic programming language can't. It presumes that its input is ASCII text, that the input can be organized into lines or records, and that the records can be organized into fields.

Simple idioms, complex tasks

The simplicity possible with awk allows it to be used within pipelines for tasks such as extracting fields from a line of input. Here's an example of a common idiom used in shell scripts. By default, awk will use a blank space in an input line (spaces and tabs) to separate fields of non-blank characters. The following awk command will print the first field of every line in the file *test*:

```
$ awk '{ print $1 }' test
```

The quoted part of the command line tells awk what to do with the file *test*. In this case, it tells awk to print the first field in each line.

The command we just showed you illustrates the basic structure of an awk command. The part in quotes specifies a list of action statements telling awk what to do, and the part after the quotes specifies the list of filenames to process.

Records and fields

When awk processes a file, it reads it line by line. Each line is considered to be a record and is treated by itself. Each record is a collection of fields, separated by white space.

In your action statements, you can refer to a field by the construct $*n*, where *n* is the field number you're interested in. In an awk program, you may have multiple action statements, each in the form

```
pattern { action }
```

where you can omit either the pattern or the action part. If you omit the pattern part, awk executes the specified action on every line. In our example, we omitted the pattern part and used just the action part, which simply prints the first field in the record.

INSIDE SOLARIS™
Tips & techniques for users of SunSoft Solaris™

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

Prices	Domestic	$115/yr ($11.50 each)
	Outside US	$135/yr ($16.95 each)
Phone	US toll free	(800) 223-8720
	UK toll free	(0800) 961897
	Local	(502) 493-3300
	Customer Relations fax	(502) 491-8050
	Editorial Department fax	(502) 491-3433
	Editor-in-Chief	(502) 493-3204

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

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

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

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

Postmaster Second class postage paid in Louisville, KY.

Send address changes to:

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

Advertising For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, extension 430.

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, Discover, or American Express, or we can bill you.

Staff	Editor-in-Chief	Marco C. Mason
	Contributing Editors	Jeff Warner
		Al Alexander
		August Mohr
	Editors	Linda Recktenwald
		Martha Bundy
	Production Artists	Marguerite E.Winburn
		Amy Scott
	Circulation Manager	Mike Schroeder
	Editorial Director	Linda Baughman
	Publisher	Lou Armstrong
	President/CEO	J. Thomas Cottingham

Let's try it out. First, go to your root directory, type ls -C, and examine your output:

```
$ ls -C
TT_DB     dev       export    lib    opt        sbin     usr     xfn
bin       devices   home      mnt    platform   shlib    var     xxx
cdrom     etc       kernel    net    proc       tmp      vol
```

Now, if we pipe the output of the `ls -C` command to our example `awk` command, we should see the following:

```
$ ls -C | awk '{ print $1 }'
TT_DB
bin
cdrom
```

For each line, `awk` reads the record, splits it into fields, and prints the first field.

Pattern matching

The pattern part of an action statement tells `awk` when to execute the action. If the pattern doesn't match, then `awk` doesn't execute the corresponding action in curly braces.

If you want to operate only on lines that contain a particular string, you can use the syntax `/x/` to represent the string, where `x` is the string you're looking for. Thus, if you wanted to print out each line that contained the word Solaris, you could use the action statement:

```
/Solaris/ { print }
```

The `awk` program has two special patterns that make report writing easier. These are the `BEGIN` and `END` patterns. The action for the `BEGIN` pattern is automatically run before `awk` reads any records. The action for `END` is run after `awk` processes the last record.

Variables and mathematics

In order for you to do some serious processing, `awk` allows you to create variables and do mathematics. If you want, you can perform calculations with values found in a specific field. To create a variable, simply assign it a value like this:

```
var = 5
```

Here, we've assigned the value 5 to the variable *var*. You can treat the fields as numeric values and access them in your calculations.

The `awk` program provides you with many mathematical operators, like addition (+), subtraction (-), multiplication (*), and division (/). Putting together what we've learned so far, you could add all the numbers in a column like this:

```
awk 'BEGIN { total=0 } { total += $1 } END { print "Total=",
   ➥total }' test
```

As you can see in this example, there are three action statements. The first has a pattern of `BEGIN`, and we use it to set a variable named *total* to 0. The second doesn't have a pattern, so it's executed for every line; this action statement adds the value of the first column to *total*. The third action statement has an `END` pattern, so `awk` executes it after it's processed all the records. In this case, the last action statement prints *total*, which at this point contains the total of all values found in the first column in the file *test*.

Decisions and looping

We've already shown you how you can execute a pattern based on a decision: Using a pattern like `/Solaris/` tells `awk` to process the action statement only if the word Solaris is found in the record. However, `awk` provides much more sophisticated decision-making capability.

You can use an `if` statement to execute a statement if a certain condition is met. To do so, you use the syntax

```
if (condition)
   stmt1
else
   stmt2
```

Thus, if the specified condition is true, the `if` statement executes the statement labeled *stmt1*. On the other hand, if the statement is false, the `if` statement executes the statement labeled *stmt2*. It will execute one or the other statement, but not both. Please note that the `else` and *stmt2* part of the `if` statement are optional. For example, the statement

```
if ( var == 2 )
   print "Var equals 2"
```

prints "Var equals 2" if, and only if, the value of `var` is 2. Otherwise it does nothing. Pretty simple, isn't it?

One of the looping constructs is known as a `while` loop. This construction looks like this:

```
while (condition)
   stmt
```

When you get to the `while` statement, `awk` checks the condition. If it's true, then it executes the statement `stmt`. Then `awk` executes the `while` statement again. Thus, as long as the condition is true, the statement will be repeatedly executed. For example, if you execute the statement

```
while ( var > 2 )
   var -= 1
```

when *var* is 10, then `awk` will notice that the condition is true. Then it will subtract one from *var*, leaving it at 9. Then `awk` will execute the `while` statement again leaving *var* set to 8. `awk` will repeatedly execute the `while` statement until finally `awk` subtracts one from *var*, and *var* is less than 2. Then `awk` stops processing the `while` statement and goes on to the next statement.

Compound statements

As you may have noticed, a single statement in a `while` loop isn't terribly useful. There's just enough room for you to change the variable your condition is based on. You don't really have enough room to do any other work.

As you may expect, there's a way around this: `awk` provides compound statements. In other words, `awk` allows you to treat an arbitrarily large number of statements as a single statement by enclosing them in a pair of curly braces and separating them with semicolons (;). You can put a compound statement anywhere you're allowed to place a statement. As an example, let's take our `while` loop and let it print the value of *var* as it executes. Go ahead and type in the following command:

```
awk 'BEGIN { var=10; while (var>2) { print
  ➥var; var -= 1 } }'
```

When you press [Enter], you'll see a column of digits from 10 to 3. You won't see a prompt, however. The `awk` program is waiting for records to process. Since we didn't specify an input file, it's waiting for you to enter the records from the keyboard. Simply press [Ctrl]D to tell `awk` that there aren't any more records.

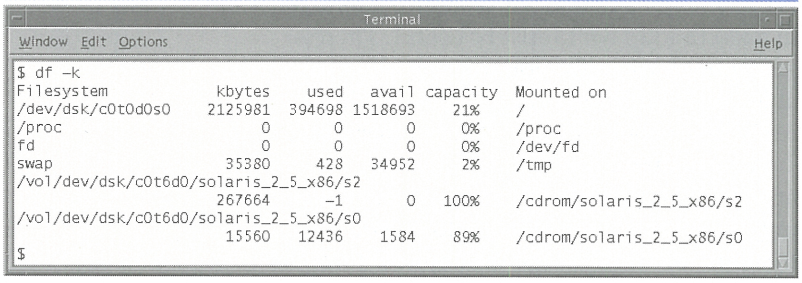
A simple sample

As a real example of the power of `awk`, we'll create a script that will print the amount of disk space on your machine in megabytes and show the total amount of space you've used. To do so, we'll take the output of the `df -k` command, which is close to what we want, convert the values from 1K blocks to megabytes, and format the output into multiple columns with headers.

Let's look at the output we get from the `df -k` command, shown in Figure A on the next page. In manipulating this input with `awk`, we'll look at several useful features of the language. First, the input fields are separated by any amount of white space (spaces or tabs). The output of `df -k` will always have white space separating the fields.

The script we'll use to manipulate and display these values is shown in Figure B. Please note that the line numbers are just for reference, they're not part of the script.

Figure A



```
$ df -k
Filesystem           kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t0d0s0    2125981  394698 1518693    21%    /
/proc                      0       0       0     0%    /proc
fd                         0       0       0     0%    /dev/fd
swap                   35380     428   34952     2%    /tmp
/vol/dev/dsk/c0t6d0/solaris_2_5_x86/s2
                      267664      -1       0   100%    /cdrom/solaris_2_5_x86/s2
/vol/dev/dsk/c0t6d0/solaris_2_5_x86/s0
                       15560   12436    1584    89%    /cdrom/solaris_2_5_x86/s0
$
```

This is an example of output from the `df -k` command. It shows disk usage for mounted file systems in 1K blocks.

Figure B

```
 1) #!/bin/ksh
 2) # Based on df -k, show disk usage in megabytes.
 3) PATH=/bin:/usr/bin
 4) df -k | awk -e '
 5) # Print the column headers
 6) BEGIN {
 7)    printf "%-18s%8s%8s%5s%8s%s\n","File", "Max","Used",
              ➥"Used","Free","Mount"
 8)    printf "%-18s%8s%8s%5s%8s %s\n","System","MByte",
              ➥"MByte","%","MByte","Dir"
 9)    TotalM = TotalUsedM = 0
10)   }
11)
12) # Process each line of df -k output
13) NR >= 2 {
14)    FileSys=$1; MaxK=$2; UsedK=$3; FreeK=$4; MntPt=$6
15)    # Ignore unmounted partitions, i.e., those with 0 byte size.
16)    if ( MaxK > 0 ) {
17)       # If the filename is too long, print it on its own line
18)       if ( length(FileSys) < 19 )
19)          printf "%-18s", FileSys
20)       else
21)          printf "%s\n%18s", FileSys, " "
22)       # Volume is mounted, display stats
23)       MaxM = MaxK/1024; UsedM=UsedK/1024; FreeM=FreeK/1024;
24)       UsedPct = UsedK*100/MaxK;
25)       printf "%8.2f%8.2f%5.1f%8.2f %s\n", MaxM, UsedM, UsedPct,
              ➥FreeM, MntPt
26)       # Accumulate totals
27)       TotalM += MaxM; TotalUsed += UsedM
28)       }
29)    }
30)
31) # Print the ending summary
32) END {
33)    print  "---------------------------------"
34)    printf "Total disk space : %8.2f MBytes\n", TotalM
35)    if ( TotalM > 0 )
36)       printf "  Percent in use : %5.1f%%\n", TotalUsed*100/
              ➥TotalM
37)    }
38) '
```

The `dfv` script uses `awk` to calculate and format a table of the megabytes used on all mounted file systems.

Line 14 of Figure B shows how we can assign these field values to variables. Note that assigning them to variables is for clarity and ease-of-use only; we can make calculations and output using the field-number variables directly. Also note that line 16 checks to see if we might divide by 0 (this could occur if a file system listed in */etc/vfstab* isn't mounted). If *MaxK* is zero, we won't process the line any further. Lines 23, 24, 27, and 36 of the *dfv* script show how we can use variables in calculations and how to assign those values to new variables.

The overall structure of this `awk` program consists of three action statements. The first, beginning on line 6, uses the `BEGIN` pattern to specify the actions we want to perform before any input lines are read or processed. In this case, we're printing our header and zeroing out our *TotalM* and *TotalUsed* variables.

The pattern that selects the second and subsequent lines uses the built-in *NR* variable, which contains the count of the current input record or line. Here, if the *NR* variable is 2 or larger, we process the action statement. This has the effect of skipping the first line output by the `df -k` command, which is a column header, as you can see in Figure A.

The last action statement, starting on line 32, simply prints the total amount of disk space and the percent used.

Formatting with printf

We use the `printf` commands in lines 7 and 8 to format our column headers. In line 25, we use another `printf` statement to display the results of our calculations. In the `printf` commands, a pattern beginning with a percent sign (%) describes the formatting of each field. After defining the output pattern, the `printf` routine substitutes the input values into the pattern in the order they appear.

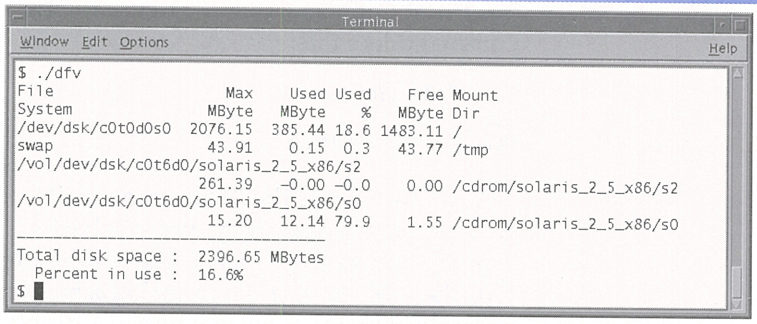
In the header commands, each field is defined as some number of string (`s`) characters. The minus sign (-) that precedes the first and last fields indicates that the field is to be printed left-justified instead of the default right-justified. Therefore, the format string `%8s` specifies a right-justified string that takes eight columns.

In lines 7 and 8, we described all fields using the `s` character, and we gave all input values as character strings. In line 25, only the first and last fields are strings; the rest are either base-10 (decimal) numbers (`d`) or floating point numbers (`f`).

As with the string values, the number given is the number of character places in the output format. For floating point fields, the number to the left of the period indicates the total width of the output field, while the number to the right of the period indicates how many of those characters should be reserved for digits to the right of the decimal point. We used the same field widths in our column headers as our numeric output to keep our columns aligned.

Note the `printf` statement in line 36. All the text appearing outside of a field definition is printed literally. In this case, `awk` will print *"Percent in use : "* followed by a number five digits wide (one digit after the decimal point), followed by a % symbol. Note that we had to use two % symbols to get a single one, since the % symbol tells `printf` to start a field definition. Figure C shows what the output of the *dfv* script looks like on one of our machines.

Figure C



```
$ ./dfv
File                   Max    Used Used    Free Mount
System               MByte   MByte    %   MByte Dir
/dev/dsk/c0t0d0s0  2076.15  385.44 18.6 1483.11 /
swap                 43.91    0.15  0.3   43.77 /tmp
/vol/dev/dsk/c0t6d0/solaris_2_5_x86/s2
                    261.39   -0.00 -0.0    0.00 /cdrom/solaris_2_5_x86/s2
/vol/dev/dsk/c0t6d0/solaris_2_5_x86/s0
                     15.20   12.14 79.9    1.55 /cdrom/solaris_2_5_x86/s0
----------------------------------
Total disk space :  2396.65 MBytes
  Percent in use :  16.6%
$ 
```

This is the output of the dfv *script in Figure B when applied to the same file systems as shown in Figure A.*

Conclusion

We didn't show you everything about `awk`. It's too big and complex a language. We do hope that we got your curiosity going, though. The basics are pretty easy to learn, and you can use them to do some fairly complex jobs. As we mentioned, you'll want to read the `man` page to get some more detail on `awk`. If you'd like some more detailed information on the `awk` language, you might want to refer to the following books:

The AWK Programming Language
Alfred V. Aho, Brian W. Kernighan, & Peter J. Weinberger
Addison-Wesley Pub. Co., 1988
ISBN 0-201-07981-X

sed & awk
Dale Dougherty
O'Reilly & Associates, Inc., 1990
ISBN 0-937175-59-5 ❖

system administration

Removing a strangely named file

Solaris

By August Mohr and Marco C. Mason

Sometimes you or your computer accidentally creates a file that has unusual characters in its name. Depending on the nature of the error and the application that created it, such a file might have control characters, quotes, tabs, or even new line characters within the name. Cleaning up such an error by removing the file can prove difficult, but it should be possible using one of the following techniques.

Filenames containing special characters

As you know, some characters have special meaning in the shell. These include the wildcard characters * and ?, quotation marks, parentheses, and the ampersand. If the filename contains characters like these that are special to the shell, all the characters of the filename may be visible in an `ls` listing, but you can't remove the file simply by typing the name as displayed.

Suppose you have a file named *T&Bar\Vee* in your directory that you want to remove. Your first impulse would be to use the rm command like this:

```
$ rm T&Bar\Vee
338
BarVee: not found
```

You can remove the file (or rename it with the mv command) by typing a backslash (\) before each special character. This tells the shell to treat the next character as a normal character. (This is why the \ didn't appear in the error message above; \V told the shell to treat the V as a V. In effect, the \ was ignored.) Since the backslash is a special character as well, a filename containing a backslash will need two of them. For example, to remove a file named *T&Bar\Vee*, you'll need to put a backslash before both the special characters & and \ in your rm command:

```
$ rm T\&Bar\\Vee
```

Names beginning with a dash

One common, but easy to deal with, case is a filename that begins with a dash (-), such as *-foo*. If you just try the obvious rm command, an error will occur:

```
$ rm -foo
rm: illegal option--0
rm: illegal option--0
usage: rm [-fiRr] file ...
```

This is because a dash introduces rm options, in this case, -f, -o, and another -o. There is a -f option to rm, but no -o, so rm complains. Even if it didn't, it still wouldn't remove your file named *-foo*. It would only set the specified options in preparation to deleting the file(s) specified, in this case, none.

There are many possible solutions, but the easiest one in this case is to precede the filename with ./. This simply specifies that the file is in the current directory. While this is what rm defaults to, adding the ./ to the start of the filename keeps rm from processing the filename as a set of switches because the dash is no longer the first character in the filename:

```
$ rm ./-foo
```

Names with non-printing characters

If the filename contains non-printing characters, such as control characters or spaces, that condition may show up in an ls listing that doesn't properly align in columns. If you're really unlucky, the filename contains the character sequence that clears your screen (e.g., the single character [Ctrl]L on the console). Every time the filename is printed, your screen clears.

One way of identifying the erroneous characters is to put the output of ls into a file and then examine the file with vi. Using the :set list command in vi will show where the lines end with a $ and let you identify embedded tabs and trailing spaces.

There are two options to the ls command that can also be helpful in identifying files with non-printing characters in their filenames. These are the -b and -q options. The -b option displays non-printing characters as three-digit octal values preceded by a \ character, and the -q flag causes them to be displayed as question marks. When you know where the non-printing characters are in the name, you can match those characters with shell wild card characters.

For instance, in a directory containing a file with the name *foo^Lbar* (with an embedded [Ctrl]L character), an ls command would just clear the screen, leaving just "bar" in the upper-left corner. Using ls -b would display it as *foo\014bar* and using ls -q would produce foo?bar. Knowing there is only one erroneous character, you could then remove the file with the command rm foo?bar (assuming no other files matched the foo?bar pattern).

You can also use the -i (interactive) flag for the rm command. Using the command rm -i * will prompt you for a y or n response for each file in the current directory. You simply respond with an n to each file except the ones you want to remove. You can use leading and trailing characters with the * to narrow the choices (e.g., rm -r f*r will offer you only files that begin with f and end with r).

You can copy all the valid files to another directory (not a subdirectory of the one with the problem). Then you can remove the troublesome file with the rest of the original directory using the command rm -r *directory_name*. Then you can recreate the directory and move the remaining files back.

If you're running OpenWin or the CDE environment, you can use the File Manager to delete the files. First, you must select the file(s) to delete. Then, in the OpenWin environment, select Delete from the Edit menu. In the CDE environment, choose Put in Trash from the Selected menu.

You can also use the `find` command to remove files using its `-inum` and `-ok` arguments. Using `-inum` followed by an inode number will match the file with that number. The `-ok` argument to `find` is similar to the `-exec` argument, except that it prompts you for confirmation before executing each command. Note that this won't work well with filenames that have characters such as [Ctrl]L that affect the display.

To use `find` with the `-inum` option, first type `ls -i`. This will give you the inode numbers of all the files in the directory. Find the inode number of the problem file. Now, to delete the file, enter the command

```
$ find . -inum number -ok rm {} \;
```

where *number* is the inode number of the file you want to remove. You'll then be prompted to confirm whether you want to remove the file.

Names containing a slash

Another very special case occurs when the file contains an actual slash (/) in its name. This isn't supposed to happen—slashes separate the components of a name and are never supposed to occur within a name. In fact, you can't create such a file—the system won't let you. Nor can you remove such a file—the slash, no matter how you type it in, will be interpreted as a component separator.

Filenames with embedded slashes can only occur as a result of a catastrophic system failure. Extreme action is needed to remove them using the `clri` command. You can't use `clri` on a mounted file system, so this technique won't work on the root file system. See the next section, "Files Containing a / in Your Root File System," for additional instructions if your file exists in the root file system.

First, find the inode number using the `ls -i` command. *Write this number down carefully.* Next use the `shutdown` command to enter system maintenance mode. You should double-check that the file system in question has been unmounted. If it hasn't, use `umount` to unmount it. Clear the inode using the `clri` command.

Suppose for a moment that we have a file named *gesoren/platz* in our */var/x* directory. Suppose further that the */var* directory is the mount point for our */dev/dsk/c0t0d0s6* file system. Using `ls` -i, we find that the inode number of *gesoren/platz* is 2345. We put the system in system maintenance mode and unmount the */var* file system. Now we can clear the inode with the command

```
# clri /dev/dsk/c0t0d0s6 2345
```

This command deallocates the inode and puts it on the free list. However, `clri` doesn't remove the entry from the directory. The directory entry that you couldn't remove is still pointing to the inode that's now on the free list. This is a potentially dangerous situation. Before you do anything else, run `fsck` on the affected file system. For the example above, run

```
# fsck -F ufs /dev/dsk/c0t0d0s6
```

and `fsck` will detect and repair the file system structure inconsistencies caused by our running the `clri` command. When it finds the directory entry that we cleared with `clri`, it will show some information like this:

```
# fsck -F ufs /dev/dsk/c0t0d0s6
** /dev/dsk/c0t0d0s6
** Currently Mounted on /a
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
UNALLOCATED  I=2345  OWNER=root MODE=0
SIZE=0 MTIME=Dec 31 19:00 1969
NAME=/gesoren/platz

REMOVE?
```

Here, `fsck` is telling you that it found a directory entry for file */gesoren/platz* using the unallocated inode numbered 2345. It wants to know if it should remove the directory entry. At the *REMOVE?* prompt, type *y* and press [Enter]. The `fsck` command will delete the directory entry and continue to scan your disk drive. Near the end, you'll get another warning:

```
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
FREE BLK COUNT(S) WRONG IN SUPERBLK
SALVAGE?
```

Now `fsck` is telling you that the number of free blocks calculated doesn't match the number of free blocks in the superblock. (The `clri` command didn't update the free block count in the superblock when it cleared the block.) It's asking if it can fix the count in the superblock. Again, type *y* and press [Enter]. Then fsck will finish fixing the errors on your drive:

```
22925 files, 394588 used, 1731393 free (3137
frags, 216032 blocks, 0.1% fragmentation)

***** FILE SYSTEM WAS MODIFIED *****
```

Warning: Running clri in multi-user mode, running clri on a mounted file system, or failing to run fsck immediately after running clri can cause file system corruption, leading to data loss.

Files containing a / in your root file system

Since the clri command works properly only on an unmounted file system, we need a way to run Solaris without mounting our root file system. For more information on this, see the article "Starting Solaris x86 in Single-User Mode"in the March issue of *Inside Solaris*.

You can use the basic technique described in that article to boot Solaris in maintenance mode from the CD-ROM. From there, you can execute the clri command to clear the specified inode and then run the fsck command as described in the previous section.

Conclusion

As with many things in Solaris, there are techniques you can use to accomplish a seemingly impossible task. In this article, we've tried to show you several ways you can solve the problem of removing files with unexpected characters in their names. ❖

system configuration

Configuring the Devices and Systems files for cu

By Al Alexander

If you've read the article "Using the cu Utility with Solaris 2.x" in our December 1995 issue, you may have started using the cu command to communicate with remote systems via a modem. In this article, we'll examine the process of configuring your *Devices* and *Systems* files to make using the cu program a little easier.

Background

The command you'll use to dial out is called cu, for Call UNIX. This command is a part of a much larger communication facility known as UUCP, or UNIX to UNIX CoPy. The entire UUCP suite of tools allows you to perform remote logins to other systems, copy files to and from other sites, and send E-mail to remote users—all via modem connections.

UUCP is now standard on all UNIX platforms. On UNIX System V Release 4 systems, such as Solaris, it is also referred to as the Basic Networking Utilities, or BNU. Configuring your system for dial-out capability with the cu command is the beginning of the entire UUCP configuration.

Configuring the Devices file

In our original article, we didn't cover system configuration. Thus, if your system wasn't already configured for use with cu, you couldn't dial out. For a minimal configuration for dial-out capability, you only need to modify one UUCP file—*/etc/uucp/Devices*—to connect to the outside world. The */etc/uucp/Devices* file tells the UUCP utilities which devices they can use to contact remote machines.

You can edit the *Devices* file with your favorite text editor. Assuming that you have a Hayes-compatible modem connected to the first serial port on your computer, add the following lines to the end of the *Devices* file:

```
ACU ttya - Any hayes
Direct ttya - Any direct
```

Make sure there are no blank spaces at the beginning of these lines. Any line within the *Devices* file that begins with a blank will be considered to be a comment. Lines beginning with a # symbol are also considered comments.

Table A provides the definitions of the fields in the *Devices* file. The first line we added to the file specifies that an Automatic Calling Unit (e.g., a modem) is connected to the serial line */dev/ttya*, that connections can be made at any speed, and that a Hayes-compatible dialer program will be used to issue commands to the modem.

If you're using a Hayes-compatible modem on your serial port, this configuration line lets you submit a phone number to the `cu` command, and the dialer program takes care of the rest of the work. For example, the following command dials the phone number 568-6250:

```
# cu 5686250
```

The second line we added to the *Devices* file specifies that we also have a direct connection to the serial line */dev/ttya*. You use this configuration line to allow direct access to the modem, in case you need to issue special commands, such as `at` or `atz0`.

Testing the Devices file changes

Once you've made these changes to the *Devices* file, you can connect to the modem directly with the command

```
# cu -lttya
```

where the `-l` command-line switch tells the `cu` program the device to which you want to establish a connection.

If this command succeeds, you'll see the message *Connected* from the computer system. You can now type commands from the Hayes command set, such as AT, and your modem should respond with the usual *OK* message, as shown below:

```
# cu -lttya
Connected
at
OK
```

For clarity, we've displayed what you type in color and what your computer displays is in black.

If you see these messages, congratulations! You've done everything you need to dial out from your system. If you make an error configuring your *Devices* file, `cu` may give you an error message like this:

```
# cu -lttya
Connect failed: NO DEVICES AVAILABLE
```

If you don't see these messages displayed on your screen, it might be because your modem isn't turned on, your modem may be connected to the wrong port, the port may not be enabled properly, or you may be using an incorrectly-configured cable. Please read your modem's installation manual before proceeding any further.

Once you successfully configure your system, you can either try dialing out to another system with the ATDT or ATDP commands, or you can disconnect from the modem. To disconnect, type the special character sequence ~. (a tilde followed by a period) and press [Enter]. The system should respond with the *Disconnected* message and return you to your UNIX prompt.

The entire connection sequence should look like this:

```
# cu -lttya
Connected
at
OK
~voyager.
Disconnected
```

Take a look at the fifth line. While you typed only the character sequence `~.` the `cu` program displayed the system name between the characters you typed.

Modifying the Systems file

With your current setup, you can call any computer in the outside world by either specifying its phone number or connecting to the *ttya* device and then issuing an ATDT or ATDP modem command. If you change an

Table A

Field #	Description
1	Device type (either ACU or Direct)
2	Device to use for the connection
3	Dialer-line (this field is archaic; just use a "-")
4	Connection speed
5	Dialer-token (`hayes` uses the Hayes command set for dialing; Direct assumes a connection is already made and this field is not required.)

These are the definitions for the fields in the /etc/uucp/Devices *file.*

additional UUCP configuration file, you can call remote systems by name, such as

```
# cu compuserve
```

The file you'll need to change is called */etc/uucp/Systems*. In this file, you specify the names of computer systems to which you may want to connect, as well as some additional information, like the phone number and connection type.

For instance, if your local CompuServe phone number is 568-6250, you can add the following line to your *Systems* file:

```
compuserve Any ACU Any 5686250
```

Table B describes the field definitions for the *Systems* file.

Table B

Field #	Description
1	Name of the remote site
2	Calling schedule—`Any` allows calls at any time of day; `Never` disallows callouts using this entry.
3	Device type (ACU or Direct)
4	Connection speed
5	Phone number of the remote site
6	Login conversation script (not used in this article)

These are the definitions for the fields in the /etc/uucp/Systems *file.*

Please note that only the first seven letters of the remote site name are expected to be unique. Therefore, you must be careful when setting up the *Systems* file to ensure that no two site names start with the same two letters.

After you configure your *Systems* file, you can easily dial in to CompuServe by typing

```
# cu compuserve
```

Conclusion

Once you've configured the *Devices* and *Systems* files, you and every user on your workstation can dial out to remote computer sites to run remote applications and transfer files. An additional benefit is that you've begun to configure your entire UUCP system. If your business has several geographically separated offices, you can continue the UUCP setup to further enable scheduled remote logins, file transfers, and site-to-site E-mail. ❖

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

foreign file systems

Solaris

Microsoft's DMF and MDF floppy disk formats

Many of you use Wabi so you can run the same Windows applications that others in your company use, such as Microsoft Word. With the older versions of Microsoft products, there usually isn't a problem, because Microsoft distributed its applications on floppy disks with the standard DOS file system on them.

However, Microsoft ships some products in huge volumes. If it can find a way to ship you the same software using fewer diskettes, it stands to save quite a chunk of change. Thus, Microsoft started using a disk format called DMF, and then more recently, MDF. In this article, we'll describe the ramifications of these two file systems on a Solaris installation.

How is a floppy formatted?

At this point, you may be wondering how Microsoft is packing the same amount of information on fewer disks. Obviously, it uses file compression as much as possible. But Microsoft still wanted more.

As you may know, a standard 3.5" DOS diskette is formatted to use 80 tracks, with

18 sectors per track, and both sides of the diskette. Since each sector is 512 bytes, the diskette holds 80 (tracks) * 18 (sectors) * 2 (sides) * 512 (bytes), or 1,474,560 bytes. Then DOS uses a few sectors to hold the root directory, the FAT tables, etc., giving you about 1.44MB to use.

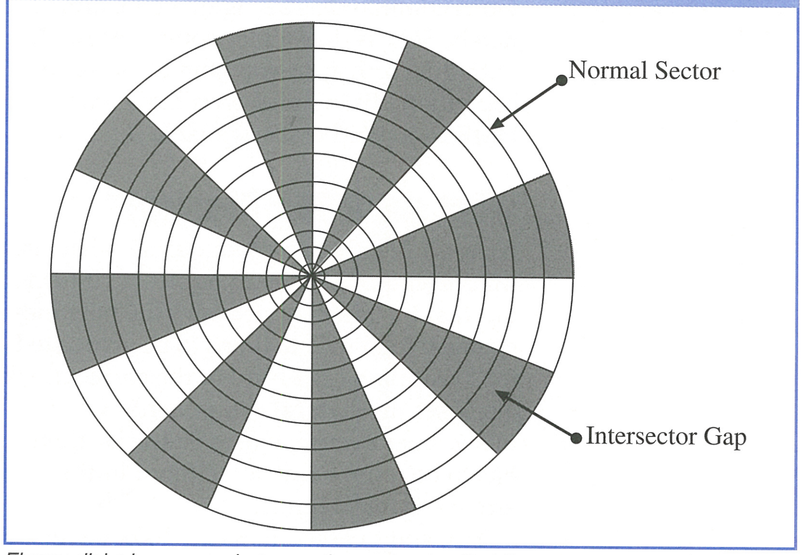
What you may not know is that sectors aren't just butting up against each other. A floppy drive is a mechanical device, and mechanical devices can't (economically) be built with the same exacting standards as computer chips. The head positions can change slightly, motor speeds can vary, etc. In order to allow floppy disks to be transported between machines, you have to build tolerances into them.

The end result is that floppy disk formats have an intersector gap that's used as a buffer zone between sectors, as shown in Figure A. This gap allows differences in motor speeds to be accommodated.

For example, suppose you formatted a floppy without an intersector gap on a machine with a motor speed that was slightly too slow. This would make your sectors slightly smaller than normal, since the disk controller writes the data at the same speed each time. Then you put your floppy in a different computer and wrote a sector to the disk. Unfortunately, this computer has a motor that runs a bit too fast. When you finished the write, your disk would look something like the one shown in Figure B.

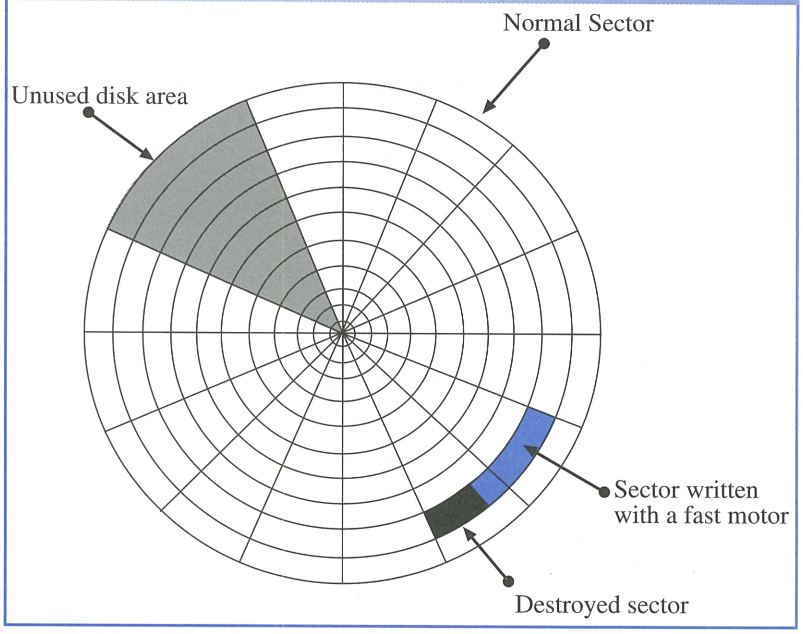
See the blue sector? This is the one written to by the machine with the slightly fast motor. As you can see, it's longer than the others. Long enough, in fact, to damage the following sector, shown in black. This is why we have the intersector gaps.

Figure A



Floppy disks have gaps between the sectors so that they can accommodate differences in motor speeds.

Figure B



This disk was formatted with no intersector gaps and used in two machines with different motor speeds.

What is DMF?

Microsoft's engineers evidently thought that the intersector gap was very conservative. They decided to shrink it in order to fit more sectors around the disk. Rather than the 18 sectors used with a standard high-density floppy disk, they squeezed the intersector gap enough to add three more sectors per track. This boosted the total to 21 sectors per track, for a total of 1.72MB. After removing the overhead for the root directory and FAT tables, you get 1.68 usable MB per diskette.

Fortunately, some versions of Solaris, such as Solaris v2.5, will mount these floppies properly as a DOS floppy, allowing you to install software from them, as well as read and write to them. You can even put a ufs file system on them, using the command

```
/usr/sbin/newfs -v -i 4096 -o space -b 4096
  ➥-c 4 -d 1 -f 512 -m 1 -s 3360 -t 2 /dev/rfd0
```

Thus, if you want to reuse your DMF disks after you upgrade a Microsoft product, you don't have to reformat them. You can simply use them as a DOS file system or install the ufs file system and use them normally. (Please note that you should enter the previous command as a single line.) Unfortunately, Solaris doesn't let you format disks with this format, so you won't be able to copy these disks unless you have spare DMF-formatted disks available.

What about MDF floppies?

Now you're probably wondering what else Microsoft could have up its sleeve. It turns out that the motor speed isn't the only mechanical tolerance that has to be accounted for. Positioning the head to the correct track also uses mechanical components. Obviously, since everyone uses 80 tracks, there has to be room for all 80 without hitting any mechanical limits.

Yes, you guessed it. Since disks have to be transportable, drive manufacturers decided to add a bit more space to move the heads. Microsoft then decided to add two more tracks and got an easy 2 (tracks) * 2 (sides) * 21 (sectors) * 512 (bytes), or 43KB of space. It's not much, but if it lets you squeeze out one disk from a package, and you're going to sell 100K copies, you can save a cool million bucks.

While some versions of Solaris will mount the DMF floppies, the MDF floppies are a different story altogether. The Solaris drivers won't let you read the last two tracks, so you can't use MDF floppies in your Solaris machine.

Workarounds

If you want to install a Microsoft product that's distributed on MDF media, there are several ways to do so. Each method requires a PC on your network running Windows and PC-NFS or PC-NFSpro. The basic idea is to read the floppies from the PC and write them to the Solaris machine's hard drive. Then you can install the software. Some Microsoft software may install differently than others, so you may have to try one or more different workarounds to install a package.

First, try creating a base directory for the software product, like */install*. Then simply copy all the diskettes into that directory. Then run the applications installation program (usually *setup.exe*) from that base directory. Some applications sold on CD-ROM and floppies will work from this method, as the CD-ROM often is just a dump of the floppy images, usually stored as a bunch of *.CAB* files.

Other packages won't work this way. In this case, a CD-ROM installation doesn't bother compressing the files since there are over 600MB available on it. Your next step is to decompress all the *.CAB* files in your */install* directory to make your */install* directory look like this kind of CD-ROM. To do so, go to the */install* directory on the DOS machine and execute the following DOS command line:

```
for %i in (*.CAB) do extract /e %i
```

This will decompress all the files in the *.CAB* files into the directory. Now you can run the installation program. If the installation program likes this method, it won't even bother asking you to change disks.

If this doesn't work, your final fallback is to create a subdirectory for each installation disk under your base directory, named *disk1*, *disk2*, etc. Then when the installation program prompts you for the next disk, just alter the path to point to the next directory. ❖

letters

Blocking foreign IP packets from accessing your internal network

I've set up a computer running Solaris as a firewall. I have all of my internal systems on one side, and an Internet connection is on the other. The firewall is the gateway between the Internet and my internal systems.

I've disabled the `in.routed` daemon, but I want to make sure that packets with a defined route can't get through the firewall. The only way I want packets to get to my internal network is for the packet to go to an application on the firewall. If my application decides to, it may forward the packets to my internal network. In summary, I want to configure a Solaris system with two network controllers such that it will *never* route a packet.

Todd Matthews
via the Internet

You've described a situation that's becoming more common all the time as the Internet continues its explosive growth. You want your users to be able to get out to the Internet, but at the same time you want to keep the rest of the world out of your LAN.

In the scenario you describe, shown in Figure A, you have a Solaris machine acting as a gateway with two network interfaces: one interface connected to your internal LAN and a second interface connected to the Internet. Your internal LAN may be a small 5-user LAN or a 5,000-user corporate network.

Using static routing tables

A classic solution to this problem has been to kill the routing daemon on the Solaris workstation acting as the router and then manually create a static routing table with the `route` command. By doing this, you create static routes to specific hosts on the Internet you want to be able to communicate with but disable all other routes.

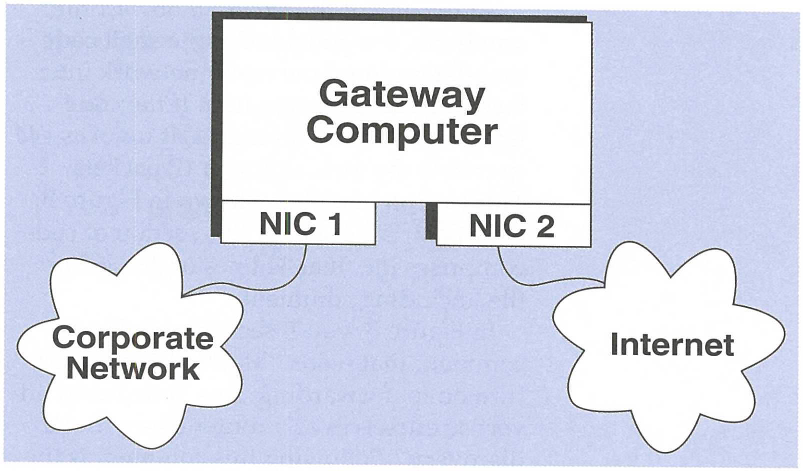
When you manipulate the routing table in this manner, it doesn't matter how a remote site sets up its routing, because it can't communicate with your host if your host doesn't have a route back to the remote site. While it sounds simple, as a practical matter, this type of firewall can require a considerable amount of ongoing administration, depending on your businesses' use of the Internet.

Using firewall software

A better alternative is to use special firewall software that will decide, on a packet-by-packet basis, whether the network traffic is safe. This solution has the advantage that it tends to be more secure, if only because the software vendors spend a lot of time worrying about the security of their firewall software. Firewall software also has the advantage of being much more flexible than a static routing table.

In your case, you already have an additional firewall software product installed on your Solaris server. Your security objective is

Figure A



The gateway computer is the single point of communication between the internal and external networks.

to disallow all TCP/IP traffic from passing through your workstation at the operating system level. Instead, you want all TCP/IP traffic to flow through your firewall software.

The problem you're probably running into has to do with an assumption that the Solaris developers made and embedded in the system startup files. Their assumption is that if, during boot time, Solaris detects that two network interfaces are present on the server, all TCP/IP network packets will be forwarded from one interface to the second. In many cases this is a fair and useful assumption, but it's not appropriate in the scenario you describe.

Please note that the software actually checks three conditions:

1. Are there more than two network interfaces?
2. Is there a point-to-point interface configured?
3. Does the */etc/gateways* file exist?

For the purposes of this article, we're assuming that your problem is due to your two network interfaces and not due to a point-to-point interface or the presence of an */etc/gateways* file.

In this case, the solution is to modify the proper startup file so that all forwarding of TCP/IP packets by the Solaris operating system is disabled. The proper startup file to modify is */etc/rc2.d/S69inet*. This file contains several lines containing the `ndd` command.

The `ndd` command is used to manage the TCP/IP driver configuration parameters. The same `ndd` command that enables IP forwarding can also be used to disable IP forwarding to resolve your problem.

At the end of the */etc/rc2.d/S69inet* file, you'll find a section of Bourne shell code that determines how many network interfaces are on your computer. If the code finds two or more interfaces, it invokes `ndd` to enable the forwarding of IP packets. This section of code is shown in Figure B. On a Solaris 2.4 server, this section of code comprises the final 34 lines of the *S69inet* file, including comments.

In Figure B, you'll see a highlighted comment that reads "Machine is a router: turn on ip_forwarding, run routed, and advertise ourselves as a router using router discovery." Following this comment is the code that performs these three steps:

Figure B

```
if [ -z "$defrouters" ]; then
   #
   # Determine how many active interfaces there are and how
      ➥many pt-pt
   # interfaces. Act as a router if there are more than 2
      ➥interfaces
   # (including the loopback interface) or one or more
      ➥point-point
   # interface. Also act as a router if /etc/gateways exists.
   #
   numifs=`ifconfig -au | grep inet | wc -l`
   numptptifs=`ifconfig -au | grep inet | egrep -e '-->'|wc -l`
   if [ $numifs -gt 2 -o $numptptifs -gt 0 -o -f /etc/gateways
      ➥]; then
         # Machine is a router: turn on ip_forwarding, run routed,
         # and advertise ourselves as a router using router discovery.
         echo "machine is a router."
         ndd -set /dev/ip ip_forwarding 1
         if [ -f /usr/sbin/in.routed ]; then
            /usr/sbin/in.routed -s
         fi
         if [ -f /usr/sbin/in.rdisc ]; then
            /usr/sbin/in.rdisc -r
         fi
     else
         # Machine is a host:if router discovery finds a router then
         # we rely on router discovery. If there are not routers
         # advertising themselves through router discovery
         # run routed in space-saving mode.
         # Turn off ip_forwarding
         ndd -set /dev/ip ip_forwarding 0
         if [-f/usr/sbin/in.rdisc]&&/usr/sbin/in.rdisc-s;then
            echo "starting router discovery."
         elif [  -f /usr/sbin/in.routed ]; then
            /usr/sbin/in.routed -q;
            echo "starting routing daemon."
         fi
   fi
fi
```

The last few lines of /etc/rc2.d/S69inet *turn on packet forwarding if you have two or more network interface cards.*

1. It executes the `ndd` command to tell TCP/IP to route packets through your computer.
2. It starts the `in.routed` daemon to allow dynamic routing configuration and force the computer to supply routing information to other computers requesting it.
3. It starts the `in.rdisc` daemon that allows the system to advertise routing information to the internal and external networks.

To solve your routing problem, you can modify the *S69inet* startup file. How you elect to do this depends on your programming style. We prefer to allow the program to come into this portion of code and then echo some text that states that "Multiple network interfaces have been found, but IP forwarding is turned off and all routing daemons are disabled." You'll see this text appear on the console at boot time. Then make the appropriate changes to the `ndd`, `in.routed`, and `in.rdisc` commands.

The changes we made to this file are shown in the highlighted section of Figure C. As you can see, we first echo some text to the console when the script runs to tell anyone watching what we're doing. Next, we changed the `ndd` command line so that the last argument is a 0 instead of a 1. This disables the forwarding of TCP/IP packets from one network interface to the other.

Note that the script no longer starts the `in.routed` and `in.rdisc` daemons. We do this because the Internet already has the routing information required to find your site, and your internal network should already have all the routing information it needs. By not providing these two daemons on the firewall computer, we're helping to prevent internal routing information from being seen on the Internet. This may make your system less attractive to hackers.

Determining whether you want to run these daemons will probably be dictated by your firewall software. If you *do* need to run these daemons, you'll want to run them in host mode instead of router mode. In that case, change the lines that start the daemons to read

```
/usr/sbin/in.rdisc -s
```

and

```
/usr/sbin/in.routed -q
```

The `-s` option to the `in.rdisc` daemon populates the host's routing tables at boot time but does not advertise itself to other computers as a router. The `-q` option on the `in.routed` daemon prevents it from supplying routing information to other computers.

Please note that any time you make changes to a script file that's used when your system boots up, you must test these changes *before* rebooting your computer. Otherwise, a simple error in the script may cause your computer to hang during system bootup. This can be difficult to recover from.

In order to prevent a system hang, copy *S69inet* into a different directory, such as */test*, and make your changes to that copy. You can then execute the script from the command line by typing

```
# sh /test/S69inet
```

This command tells Solaris to start a new Bourne shell and that the Bourne shell should run your modified *S69inet* script file. After your script file runs, you'll be returned to your system prompt. If the script doesn't run correctly, you'll need to verify the changes you made. After you've tested your changes and verified that they work properly, copy the *S69inet* script back into the */etc/rc2.d* directory. The next time you boot your computer, it will automatically execute your script and quit forwarding IP packets indiscriminately.

You can find more information about TCP/IP routing by referring to the `man` pages on these topics: `in.routed`(1M), `in.rdisc`(1M), `route`(1M), `routing`(4), `ip`(7), `netstat`(1M), and `ndd`(1M).

Figure C

```
if [ -z "$defrouters" ]; then
  #
  # Determine how many active interfaces there are and how many
     ➥ pt-pt
  # interfaces. Act as a router if there are more than 2 interfaces
  # (including the loopback interface) or one or more point-point
  # interface. Also act as a router if /etc/gateways exists.
  #
  numifs=`ifconfig -au | grep inet | wc -l`
  numptptifs=`ifconfig -au | grep inet | egrep -e '-->' | wc -l`
  if [ $numifs -gt 2 -o $numptptifs -gt 0 -o -f /etc/gateways
     ➥]; then
    # --------------  CHANGES START HERE  --------------------
    echo "Machine has 2 or more network interfaces, but IP"
    echo "forwarding is disabled and networking routing and"
    echo "router discovery daemons are disabled."
    ndd -set /dev/ip ip_forwarding 0
    # if [ -f /usr/sbin/in.routed ]; then
    #        /usr/sbin/in.routed -s
    # fi
    # if [ -f /usr/sbin/in.rdisc ]; then
    #        /usr/sbin/in.rdisc -r
    # fi
    # --------------  CHANGES STOP HERE  --------------------
  else
    # Machine is a host: if router discovery finds a router then
    # we rely on router discovery. If there are not routers
    # advertising themselves through router discovery
    # run routed in space-saving mode.
    # Turn off ip_forwarding
    ndd -set /dev/ip ip_forwarding 0
    if [ -f /usr/sbin/in.rdisc ] && /usr/sbin/in.rdisc -s; then
      echo "starting router discovery."
    elif [  -f /usr/sbin/in.routed ]; then
      /usr/sbin/in.routed -q;
      echo "starting routing daemon."
    fi
  fi
fi
```

This modified /etc/rc2.d/S69inet *file turns off packet forwarding and doesn't start the routing daemons.*

How do you perform system maintenance on the root file system?

About your article in the March issue of *Inside Solaris*, "Oh, No—I Forgot the Root Password." The tricky part is to get the shell prompt in single-user mode without giving the root password. I have Solaris 2.4 and I can't get permission to enter the shell in init level 3, nor can I use `boot -s` without giving the root password. Maybe you know how to do this?

Marita Oman
via the Internet

Marita, you can do so by following the same general procedure described in the article "Starting Solaris x86 in Single-User Mode" on page 11 of the same issue. Please note that the prompts may be different for the SPARC installation. Just follow the installation procedure, and exit from the installation at the first opportunity.

When you exit the installation program, you're left at single-user mode, with the CD-ROM mounted as your root file system.

SunSoft Technical Support

(800) 786-7638

SECOND CLASS MAIL

[illegible] 7831 02/97
[redacted]
[redacted]
[illegible] GA 30131

Please include account number from label with any correspondence.

You can then mount your root file system on your boot drive at the */a* directory with the command

```
mount /dev/dsk/filesys /a
```

and proceed to recover your root password. (Note that *filesys* is the name of your boot partition, such as c0t0s0 if it's the first slice of the first IDE drive on the first controller.)

It's important to note that this trick is much more useful than just allowing you to recover a lost password. If you've ever needed to perform some system maintenance commands on your root file system, such as `fsck` or `clri`, you'll find that some commands only work on a file system that isn't mounted. In this case, you can use the same trick to mount the CD-ROM as your root file system, so you can leave your normal root file system unmounted. Since the CD-ROM is already set up with a */bin* directory with all the file system maintenance commands, you're all ready to go, without having to install a bootable Solaris partition on a second hard drive.

File locking under Wabi

Help! I have a database application that I'm running under Wabi, and I'm seeing some strange behavior. I can open tables on a normal Solaris partition without any problems. If I try to open a database table on a DOS partition, however, it fails. What's going on?

Isaac Curtis
Los Angeles, California

That's a strange problem. After some investigation, we found that Sun is aware of the situation. It turns out that Solaris doesn't support file locking on DOS partitions (the ones you mount as type pcfs).

When an application requests to open a file with locking, Solaris tells the program that file locking isn't available. This causes your program to think that the open operation failed, which is causing your problem. This isn't a concern for most Windows applications, since few of them open files with locking.

Sun recently published a workaround for this problem so you can use your database. Just set the environment variable WABI_NOLOCK to 1 before you start Wabi. This tells Wabi not to lock any files, even when requested.

If you're using the C shell, you can do this with the command:

```
setenv WABI_NOLOCK=1
```

For the Bourne and Korn shells, you use the following commands:

```
WABI_NOLOCK=1
export WABI_NOLOCK
```

After you change the environment variable, when an application requests a file open with locking, Wabi reports that the file is open and locked. Then you can use your tables on your DOS partition.

Keep in mind, however, that your files aren't locked, so if another person uses these tables at the same time, you can lose data, even if they're on a standard Solaris (ufs) partition. If that's a problem, you should probably move your data tables to a standard Solaris (ufs) partition instead of using this technique. ❖

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