March 1996 • VOL. 2 NO. 3
US $11.50

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

in this issue

Customizing the vi editor 1
Creating macros in vi 4
Oh, no—I forgot the root password! 7
Don't panic! 9
Starting Solaris x86 in single-user mode 11
Netscape 2.0n ported to Solaris v2.5 15
Locking out a user 15

Customizing the vi editor

By Al Alexander

For years, UNIX users and administrators have had a love/hate relationship with the `vi` editor—in general, they've loved to hate it. Many of the problems people have with the `vi` editor can be alleviated with a few configuration options that can automatically be set during startup.

Most users never learn about this startup option and how much easier to use it can make the `vi` editor. In this article, we'll explore some of the most useful commands you can use to customize `vi`.

The vi startup sequence

When you start the `vi` editor, it searches for the environment variable EXINIT and uses its contents as a set of configuration commands. The EXINIT variable, however, is limited in the commands you can place in it and, as such, isn't the best option to use.

If EXINIT is not defined, `vi` looks for the *.exrc* file in your home directory (i.e., *~/.exrc*) and uses its configuration commands. If one of the configuration commands in *~/.exrc* is the `:set exrc` command, `vi` looks in your current directory for a file named *.exrc*, and if it exists, `vi` executes the commands in that file.

In this manner, you can have a default configuration for `vi` as well as a different configuration for each directory you're working in. In this article, we'll modify the *.exrc* file in our home directory, which will affect all of our `vi` editing sessions.

Which customizations will make using vi simpler?

It's nice to know that `vi` provides such a flexible startup procedure. But for now, that's begging the question. Just what sorts of customizations does `vi` provide for?

It turns out that there are quite a few things you can do to make `vi` simpler to use. Some commands modify the way `vi` operates, and other commands allow you to create useful macros. We'll show a sampling of the more useful commands we've found.

Showing the current mode

One of the biggest complaints about `vi` is that you never know what mode you're in—there's nothing onscreen to indicate whether you're in command mode or insert mode. You can easily cure this complaint by issuing the `:set showmode` command.

The `:set showmode` command forces `vi` to give a visual indication of what mode `vi` is in when you're typing—insert mode, append mode, etc. If you're in command mode, as when you first enter `vi`, you won't see anything different. But as soon as you enter insert mode, the words *INSERT MODE* appear in the lower-right corner on your screen. Similarly, `vi` displays *APPEND MODE*, *CHANGE MODE*, and *OPEN MODE* when appropriate.

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

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 is pending in Louisville, KY.

Send address changes to:

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

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

Masking screen garbage

When you're in a full-screen editor, one of the last things you want is another user or process sending messages to your screen. (Another user can use the `talk` command to ask you a question, for example.) When messages are sent to your screen, they're mixed with the `vi` screen, making it difficult to see what you're editing.

Hence, many `vi` users learn (and frequently use) the `^L` command, which redraws the screen. Since the `^L` command is short and simple, many people use it each time their screen is jumbled instead of searching for a better way. The better way is to tell `vi` not to allow other users or processes to send you messages. That is precisely what the `:set nomsg` command does.

Setting tab stops

By default, `vi` sets tab stops to eight characters. When you're programming, this can be annoying, as deeply-indented structures can scroll off the right side of the page. That's why programmers often use tab stops of three or four characters. With the smaller tab stops, you still get good indentation, but you don't run out of space so quickly.

You can tell `vi` how many characters wide a tab stop should be with the `:set tab stop=#` command. All you need to do is replace the `#` symbol with the number of character positions you'd like to use. Thus, for programming, you might use:

```
:set tab stop=4
```

Showing line numbers

Another feature that helps programmers is knowing the line number that you're on. When you compile a program and encounter errors, the compiler tells you which line numbers the errors are on so you can find them more quickly.

You can tell `vi` to display the line numbers in a source file by using the `:set number` command. When you want to turn line numbering off, simply use the `:set nonumber` command.

The EXINIT environment variable

If you use the EXINIT environment variable to customize `vi`, you're limited to using the set commands. (If you want to see a list of all the set commands, invoke `vi` and type *:set all*. When you do, `vi` will show you a list of all available set commands.)

In order to use the EXINIT method of customizing `vi`, you simply set the EXINIT environment variable with a string that represents the options you want. The first word of the string should be `set`, and the rest of the string is the list of set commands with their parameters, if any. Once you've set the EXINIT environment variable, you must export it so that `vi` will see it when it starts.

Suppose that you want to show the current insert mode, you want tab stops to be set to 4, and you want line numbering turned on. Using the EXINIT environment variable, you could do so with the following commands:

```
EXINIT='set showmode tab stop=4 number'
export EXINIT
```

Please note: In this example, we used the Korn shell, which is the most standard shell. The Bourne shell uses the same syntax. If you're using the C shell, you would instead use the single command:

```
setenv EXINIT 'set showmode tab stop=4 number'
```

The .exrc file

If you want your default `vi` configuration to have predefined macros, you'll need to use the *.exrc* file instead of the EXINIT environment variable. The *.exrc* file can contain comments, colon commands (those entered on the last line of the `vi` editor and beginning with :, such as `:set showmode`), and macro definitions. (For more information on macros, see the accompanying article "Creating Macros in vi," on page 4.)

Comments in the *.exrc* file are those lines that begin with a " character (double-quote character). `vi` ignores all characters after the ". You can include as many comments in the *.exrc* file as you like. One warning—don't put any blank lines in the *.exrc* file. `vi` doesn't care for blank lines in the configuration file very much and may ignore your configuration options.

A sample .exrc file

Now let's put all of this information together into a single example. Figure A shows an *.exrc* file that sets up some features that make `vi` very easy to use.

Let's examine this *.exrc* file to see just what it does for us. The first three commands shouldn't be any surprise, as we covered them already. They simply tell `vi` to show us which mode we're in, to block messages from external sources, and to allow further customization in *.exrc* files found in the current directory.

In our next command, we map a command to [F1]. After the `:map #1` portion of the first command, you see the character sequence `:!more ~/.vi_help^M`. If you're familiar with `vi`, you may know that the `:!` sequence allows you to run any UNIX command while you're in the `vi` editor. For instance, while you're in command mode in `vi`, you can type `:!ls -al` (followed by the [Enter] key) to get a long listing of all files in your current directory.

This is a great feature because you don't have to exit and re-enter `vi` just to run the `ls -al` command, saving you a lot of

Figure A

```
" Issue the 'showmode' command to tell vi to display the current editing mode
:set showmode
"
" Block messages from other users to keep my display clean
:set nomesg
"
" Tell vi to also read the .exrc file in the current directory for further customization
:set exrc
"
" Define F1 to show a customized "help" file
:map #1 :!more ~/.vi_help^M
"
" Define F2 to save current changes to file
"  (uses the current filename)
:map #2 :w^M
"
" Define F3 to display line numbers on screen
:map #3 :set number^M
"
" Define F4 to remove line numbers from screen
:map #4 :set nonumber^M
"
" Define F5 to prepare for programming by turning on the autoindenting feature and setting the tab stops to 4
:map #5 :set autoindent^M:set tab stop=4^M
"
" Define F6 to set the tab stops to 4 characters
:map #5 :set tab stop=4^M
"
" Define F10 to display "long help" for vi
:map #0 :!man vi^M
"
" Set some useful abbreviations
:ab s4 Solaris v2.4
:ab s4x Solaris v2.4 for x86
:ab s5 Solaris v2.5
:ab s5x Solaris v2.5 for x86
```

Here's the .exrc file we use in our home directory to prepare `vi` for serious use.

keystrokes. We simply created a file in our home directory named *.vi_help* that we use to hold some quick notes on using our customized version of `vi`, as well as some frequently-used commands. Our *.vi_help* file is shown in Figure B. Now, by pressing [F1] we can call up this file to the screen to refresh our memories.

We've created a few other macros as well in this customization file. We've defined [F2] to write the file, so we don't lose any changes. [F3] and [F4] turn the line numbers on and off, respectively. We defined [F5] to turn on the autoindent mode and set tab stops to 4 to make `vi` more convenient for programming.

While the short help file is useful, sometimes we need to search for a command that we don't often use. That's why we defined [F10] to open the manual entry for `vi`. Here we can search for any command we want. To do this, we again started a shell and executed a command, only this time we executed the `man vi` command.

Since we write about Solaris often, we created a few abbreviations. Thus, when we write articles, we won't have to type *Solaris v2.5 for x86* whenever we need to reference it. Instead, we can type *s5x*, and let `vi` do the substitution for us.

That's all there is to it

You've now seen how easy it is to customize `vi` to suit your preferences. While you have some limited ability to customize `vi` with the EXINIT file, you'll probably want to create your own *.exrc* file because of the extra flexibility.

Figure B

```
************************** Short Help for Vi ****************************
F1-Short help, F2-Write File, F3-Show Line #s, F4-Hide Line #s,
F5-Autoindent ON+Set Tabs=4,  F10-Long help

s4  -> Solaris v2.4
s4x -> Solaris v2.4 for x86
s5  -> Solaris v2.5
s5x -> Solaris v2.5 for x86

Delete: x=curr char, d$=del to end of line
        dd=current line, #dd=next # lines

Insert: a=after curr char, i=before curr char, o=new line after this one
        A=after end of line, I=at start of line, O=new line before this one

Move:   0 (zero)=move to start of line, $=move to end of line
        b=back one word, w=forward one word
        G=end of file, #G=goto line #

Misc:   :q!=Quit, no save, :wq=Save & Quit, /xxx=search for xxx
        !cmd=execute command cmd in shell
```

This is the text of our *.vi_help* file that we use to remind ourselves of the commonly-used vi commands and our macro definitions.

Conclusion

In this article, we used the *.exrc* file to display the current working mode, keep other users' messages off our display, and create a set of useful macros. You, too, can make your life with the `vi` editor easier and more productive. You can now apply the techniques contained in this article to customize `vi` to include your own working preferences. ❖

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 provided UNIX and Internet training to over 400 clients in the last three years.

application tips

Creating macros in vi

In an ideal world, our lives would be filled with a variety of interesting things. At work, however, we must be content to do whatever is required to complete our jobs. Often, this means performing endless repetitive tasks.

The good part is that we deal with computers. Instead of performing these repetitive tasks ourselves, we can make our computers do them. That is, after all, why computers were invented.

In this article, we're going to show you how to use the macro facilities in `vi`. Using this technique, you'll be able to perform complex tasks with only a couple of keystrokes.

The :map and :unmap commands

The :map command lets you customize the vi editor by allowing you to redefine the meaning of a key or combination of keys when you're in command mode. The :unmap command lets you tell vi to forget a specific mapping. You can create a macro using the following syntax:

```
:map lhs rhs
```

You replace *lhs* with the key sequence you want to use to trigger the macro, and you replace *rhs* with the key sequence you want to feed to vi. When you're done with macro *lhs* and you want to remove it, just type

```
:unmap lhs
```

For example, if you often need to turn on and off the line numbering facility in vi, you'll need to use the :set number and :set nonumber commands repeatedly. You might like to have a couple of shorthand commands, say X and Y, to turn on and off the numbering facility. To do so, we can create the following maps:

```
:map X :set number
:map Y :set nonumber
```

With these macros defined, you should be able to turn on the numbering by using the X command and turn it off again with the Y command. Right? Well, almost. If you press X, vi waits for you to press the [Return] key before displaying line numbers. We need some way to insert the carriage return into the macro definition.

Adding control characters to macros

As you may know, the vi editor has a way for you to embed control characters in your files. You can use the same method to embed control characters in your macros.

All you need to do to put a control character in your text (or macro) is to precede it with [Ctrl]V. Then vi inserts the keystroke that follows into the text without processing it as a command. So, to insert the [Return] key into your macro, just press [Ctrl]V and then [Return].

Now we can redefine our macros to turn on and off line numbering so they don't pause until you press [Return]:

```
:map X :set number^M
:map Y :set nonumber^M
```

When you press [Ctrl]V, vi displays the ^ symbol on your screen. When you press [Return], it displays the M, giving you ^M. In case you don't already know, a ^M ([Ctrl]M) character is the same as the [Return] key. If you're at a prompt, type ls[Ctrl]M, and you'll see a directory listing.

Which macros have I defined?

If you ever need to see which macros you've defined, you can simply use the :map command with no parameters. When you do so, vi will display all the macros you've defined. You'll see a list like the one shown in Figure A.

Notice that vi displays seven macro definitions instead of only two. That's because vi doesn't know about the [Insert] and cursor control keys, so Sun Microsystems modified vi to map them to the appropriate commands for you.

Figure A

```
inschar ^[[2~   i
up      ^[[A    k
down    ^[[B    j
left    ^[[D    h
right   ^[[C    l
U       U       :set number
V       V       :set nonumber
[Hit return to continue]
```

You can see the macros you've defined in vi by issuing the :map command.

Which keys can I map?

Just what can you map in vi? You can map any printable character that you want. If you do so, and vi uses that character for a command, you won't be able to use that command from the keyboard. Usually that's not a problem, as few people use *all* the commands that vi provides.

Further, vi won't let you map multiple-character sequences unless they start with a nonprinting character. This prevents you from accidentally invoking a macro. For example, if you could create this macro

```
:map et :q!
```

then you couldn't use any of the set commands, or vi would quit without saving any changes to your current file.

If you want to map a function key, you can map the first ten by using #1 through #9 for the first nine function keys and #0 for [F10]. In addition, there are many [Ctrl] and [Alt] combinations that you can use for macro definitions.

Special note about :map

Please note that macros have the ability to trip you up: When you execute a macro in vi, it first finds the string it's about to execute. *Then it repeatedly tries to perform any macro substitutions it can in the string.* When it can find no more substitutions, it then executes the macro.

Suppose, for example, that you define the two macros

```
:map p dd
:map d xxxx
```

Here, the first macro definition tells vi to delete the current line of text (using vi's dd command) when you press *p*. Then the second macro tells vi to delete four characters when you press *d*. However, this changes the effect of the first macro! After you define the second macro, vi deletes eight characters when you press *p*.

This happens because when you press *p*, vi loads the macro buffer with dd. Then it maps each *d* key to *xxxx*. The result is that *p* executes *xxxxxxxx*. If you use a lot of macros and some of them intersect like this, you will see unexpected results.

If you experience this problem, you might want to consider telling vi not to successively process macros. The command to do this is `:set noremap`. If you want to restore the default behavior, use `:set remap`.

What about using macros in insert mode?

The macros you create using the `:map` command work only while you're in command mode. You can't create a macro in insert mode that will execute commands for you. But all is not lost! The vi editor has two special facilities for use in insert mode that can still make your life easier.

The first of these is the `:map!` command, which operates much like the `:map` command, except that it operates in insert mode. If you want to execute commands, you must first make your macro process the [Esc] key. You can also use it to perform text replacements. Just as you'd suspect, the `:map!` command by itself lists all the insert-mode macros, and the `:unmap!` command can erase a macro.

Thus, using `:map! ^E Codfish` tells vi to insert the string *Codfish* into your text when you're in insert mode and press ^E. Similarly, using the `:map! ^G ^[:w^Ma` command allows you to save your file while you're in text mode. It works by first pressing [Esc] to get back to command mode, then issuing the `:w` command with carriage return, and finally issuing the `a` command to get back into insert mode.

The `:map!` command has the same caveat as the `:map` command: If you have strings that clash, then they'll be mixed together. However, you can use multiple-letter macros that start with printable characters. Suppose you create the following three macros:

```
:map! fi endif
:map! od enddo
:map! ^E Codfish
```

If you actually type the ^E while in insert mode, instead of getting *Codfish*, as you would want, you'll get *Cenddoendifsh*. What we really need is a command that creates macros that are replaced only if the key sequence is a separate word. Fortunately, vi provides that command.

A better macro facility for insert mode

The second macro facility for insert mode is the `:abbreviate` command. By the name, you may have guessed what an abbreviation does. If you type the abbreviation in insert mode, vi will replace it with the full text of the phrase.

For example, since I'm the author of *Inside Solaris*, I might want to abbreviate the text string *Solaris v2.5* to *s5*. Then, as I enter text, I could just type *s5* anytime I intend to type *Solaris v2.5*.

The `:abbreviate` command operates just like the `:map` command. To make an abbreviation, just type:

```
:abbreviate lhs rhs
```

replacing ***lhs*** with the abbreviation you want to use and ***rhs*** with the string you want vi to type in its place. It has always amused me that the `:abbreviate` command is so long. Fortunately, vi abbreviates the `:abbreviate` command, so you can use `:ab` instead of its longer counterpart.

Suppose I create the abbreviation:

```
:ab s5 Solaris v2.5
```

and I want to type "I am using Solaris v2.5." In insert mode, I can type "I am using s5," and you'll see "I am using s5" on the screen. Then when I type a space, [Return], or a punctuation character, vi will replace the text *s5* with *Solaris v2.5*.

This is the feature that we like about the `:ab` command. Using the `:map!` command, if I really meant to type *s4* and typed *s5*, vi would have replaced the text with *Solaris v2.5*. In order to get back to *s4*, I'd have to delete the phrase and retype it. With the

:ab command, vi waits until you complete the word before expanding the phrase. This prevents the macro mixup (remember *Cenddoendifsh*?) we described previously.

Another benefit of this behavior is that vi allows you to have multiple abbreviations with the same beginnings. Waiting for an extra key allows vi to detect when you've completed the abbreviation.

As an example, here are three abbreviations you might use to write about Solaris:

```
:ab s5 Solaris v2.5
:ab s5s Solaris v2.5 for SparcStations
:ab s5x Solaris v2.5 for x86
```

When I type *s5*, vi doesn't know for sure which abbreviation I mean. If I add an *s*, I could still change my mind and replace it with an *x* or just remove it.

Just as with the :map command, you can view your abbreviations by using the :abbreviate command without any parameters. Also, you can remove an abbreviation with the :unabbreviate (or :unab) command.

Conclusion

The vi editor provides two types of macros. Using the :map and :map! commands, you can create powerful macros that can perform complex operations: the first while vi is in command mode, and the second when vi is in insert mode. When vi is in insert mode, the :ab command lets you type abbreviations and have vi automatically expand them to full phrases. ❖

system administration

Oh, no—I forgot the root password!

Just before you went on a Hawaiian vacation, you cleaned up your office. Now you've returned, expecting a nice day of answering mail and playing phone tag. Unfortunately, you see several people camping out in your office: The computer is down, and they're waiting for you to fix it.

After a moment of fright, you realize that the request is a simple one. Just log in as *root* and tweak a parameter, and you're home free. You sit down and type *root* at the login prompt. Then you type the password. The system rejects your attempt.

All of a sudden, you have a cold feeling in the pit of your stomach. The password is wrong. What did you change it to? Where did you put that slip of paper that you use to remind yourself? After a half-hour of frantic searching, you realize you're doomed. You can't log in as *root*. What do you do?

In this article, we'll show you how to regain access to the root account after you've lost your password. Be forewarned that it takes a bit of time and effort. Fortunately, the method works and isn't a security nightmare.

Let's get going

First you have to put your computer in single-user mode. (If you have Solaris x86, you'll need to refer to the article "Starting Solaris x86 in Single-User Mode" on page 11.) Now that we're at the shell prompt, we can do the dirty work. First, we'll need to mount the file system that contains the */etc* directory. Then, we'll edit the *shadow* file to tell Solaris that the root account has no password. Finally, we'll unmount the file system and reboot Solaris.

Mounting the root file system

If you know which file system holds the */etc* directory, this isn't a major problem. Just issue the mount command

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

replacing *filesys* with the appropriate file system name. If you've installed Solaris on the first IDE drive, for example, the file system name will be c0d0s0, or slice 0 of the first disk on the first controller card. Similarly, if the file system is on the third SCSI disk on the second controller, you'd

use c1t3d0s0. If you know which file system to mount, you can skip the next section.

What if I don't know which file system to mount?

If you don't know which file system to use, you'll have to try them all, one at a time. (I'd try c0d0s0 and c0t0d0s0 first, just in case someone set the system up as simply as possible.)

To try using a file system, issue the `mount` command and see if the system complains. If it complains, your screen will look something like this:

```
# mount /dev/dsk/c1t0d0s0
mount: /dev/dsk/c1t0d0s0 is already mounted,
/a is busy,or allowable number of mount
points exceeded
```

If it doesn't complain, then use the `ls` command to list the directory to see if the */etc* directory is there. If the drive successully mounts but doesn't contain the */etc* directory, issue the `umount` command and try the next file system. This process looks like this:

```
# mount /dev/dsk/c0d0s6
# ls /a
5bin    dict     lib      oasys    sadm
tmp     adm      dt       lost+found
old     sbin     ucb      aset
games   mail     openwin  share    ucbinclude
bin     include  man      opt      snadm
ucblib  ccs      kernel   net      preserve
spool   vmsys    demo     kvm      news
pub     src
# umount /a
```

Removing the root password

Now we have to remove the password entry from the */a/etc/shadow* file. To do so, type ***vi /a/etc/shadow***, and you'll see a screen like the one shown in Figure A.

The password is encrypted so you can't read it. It's the jumble of characters between the first two colons on the root line (the first line, in this case). All you need to do to remove the password is to delete the characters between the first two colons on the root line. In other words, change root:sXuu63aJkkTml: to root::. Now save the file. Since it's read only, you must use the `:w!` command. If you try the `:w` command, you'll receive the error message *"/a/etc/shadow" File is read only.*

Now all you need to do is remove the floppy from drive A, unmount the file system, and reboot the computer. To do so, just type

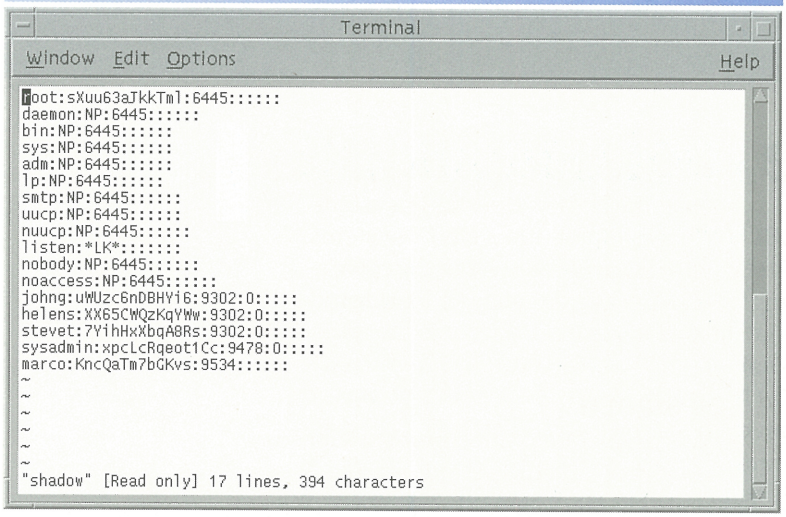
```
umount /a
reboot
```

Be sure to change the root password once the system boots up. You don't want to leave the system open after all this work! (Also, be sure to remember the password this time.)

If you'd like to avoid the headache of using trial and error to find the */etc* directory, just print a copy of the */etc/vfstab* file and place it somewhere safe. Figure B shows the */etc/vfstab* file on my work machine.

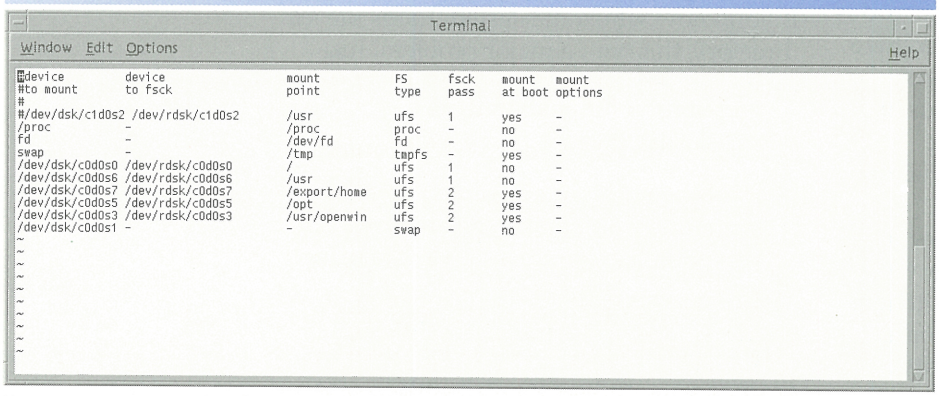
As you can see, there's no explicit entry for */etc*, so it's located in the root directory, which is on c0d0s0. Use the mount point

Figure A



Removing the characters between the first two colons on the root line clears out the password.

Figure B



If you set up your machine in the standard fashion, you'll see the */etc* directory on the file system that holds /.

column to determine which part of the directory tree is held by a file system.

What are the security risks?

That's all there is to it. The technique is tedious, but not terribly complex. But now there are a few thousand more people who know how to do it. From a security standpoint, that's not too bad.

First of all, many UNIX administrators already know this technique. The technique is tricky enough that most people won't attempt it casually. Second, you need access to the Solaris installation disk, the CD-ROM drive, *and* the console. These things are pretty easy to keep under control.

There's no chance of someone removing the root password remotely, as long as the permissions on the */etc/passwd* and */etc/shadow* files aren't changed. For your information, the permissions on these two files are normally set to

```
-rw-r--r--  passwd
-r--------  shadow
```

Conclusion

Obviously, you want to keep your system secure, and you shouldn't forget your password. But at least you now have peace of mind: If you ever lose the root password, all is not lost. With a bit of patience and work, you can still get back into your system. ❖

book review

Don't panic!

By Marco C. Mason

In the days before the Internet became popular, there were few UNIX books in the bookstores. Now, however, there are quite a few to choose from. While browsing in my local bookstore recently, one in particular caught my eye: *Panic! Unix® System Crash Dump Analysis*, by Chris Drake and Kimberly Brown. If you've ever seen your computer hang and give you a panic message and wondered what was going on inside UNIX to cause this problem, this is the book for you.

Should you get this book?

This book demonstrates some of the techniques used to decipher the state of a machine after it experiences a fatal system crash, known as a *panic*. The authors give a brief introduction to adb, the assembly language debugger, and show you how to search the include files for symbol information.

Since the topic is very technical, you probably won't want to read this book if you aren't familiar with programming. If you've ever done any assembler or C programming, you have the requisite experience.

On the other hand, if you're really interested in becoming a UNIX guru, this book can give you some invaluable insight into the inner workings of Solaris because the authors use the Solaris operating system. Thus, while users of other UNIX systems can benefit, Solaris users have the advantage that the examples should work properly the first time.

Getting started

The book is divided into three major sections. The first section, called "Getting Started," helps you learn the basics about panics. First, you'll learn a lot of background information such as:

- The difference between a panic and a coredump
- Why panics occur
- How to save the panic information in preparation for analyzing it
- What to do when your system panics
- How to force your system to panic (This technique won't make you any friends if you panic a system that others are using!)

Then comes the meat of the section: a tutorial on the features of adb that you'll use to analyze crash dumps. The authors show you the basic adb commands and then

show you how to extract certain pieces of information that will help you understand the system, such as the boot time, how long the system was running, the machine type, the version of Solaris, etc.

Next, you'll learn how to get the symbol information from the kernel and coordinate it with the information found in the header files on your system. You learn how to use this information to decode more complex information that pertains to your system crash.

Finally, you'll learn how to build adb macros that will help you perform complex tasks with just a few keystrokes. This is the part of the book I found most interesting. The authors teach you step-by-step how to build adb macros and then present some problems for you to solve. The problems range from fairly easy to difficult enough to really make you think.

Advanced studies

Once you're familiar with the tools and techniques required to debug your system, the authors dive in and teach you some of the really hard (but fascinating!) material. They start with an introduction to assembly language, and this is the only place I feel that the book is weak. The introduction is accurate but so brief that it's useful only as a refresher for those who've forgotten assembly language.

If you're really interested in becoming a UNIX guru, this book can give you some invaluable insight into the inner workings of Solaris.

The rest of this section is very good. The authors describe the methods you use to trace the stack, and they cover all kinds of information about the internals of Solaris, such as:

- Memory management
- Process scheduling
- Threads
- File systems
- Device drivers
- Interprocess communications
- Streams

If you're running Solaris on an x86 machine, you'll find that the information on stack tracing isn't nearly as useful as the stack frame structure changes on x86 machines. Any x86 user interested in this book should probably also get a good book on the x86 internals. (One of my favorites is the *Pentium Processor User's Manual, Volume 3: Architecture and Programming Manual*.)

Even if you're not particularly interested in crash dump analysis, this section of the book is a great read. It gives you an appreciation for all the activity that's going on "under the hood."

Case histories

Once you've gotten this far, you should be able to figure out why a kernel panics. It would be hard to decipher though, because you don't have any experience. In this section, the authors try to impart some of their experience to the reader by analyzing eight different panics. After you read this section, you'll be better prepared to give it a try on your own.

It was refreshing to see that they didn't hold anything back. They showed one case, "A Stomped-on Module," where they were unable to fix the problem. In this case, they were able to find that something was trashing some code, but they weren't able to pin down the culprit. A realistic example like this can help manage your expectations: Not even UNIX wizards are always going to figure out what causes some panics.

Conclusion

This book is a must if you're interested in adb, panics, and/or the Sparc architecture. It's published by SunSoft Press, ISBN: 0-13-149386-8. Next month, I'll discuss *another* great book I found while wandering the bookstore. ❖

Starting Solaris x86 in single-user mode

By Marco Mason

Don't you hate it when you can't get the computer to run properly? Perhaps you've made a system configuration change, and now the system refuses to boot properly. Maybe the file system's corrupt and the kernel won't load. Perhaps you just forgot your root password and need to get into your system. (See the article "Oh, No—I Forgot the Root Password" on page 7.)

When these types of situations occur, you'll need to start up Solaris in single-user mode. In this mode, Solaris gives you full access to your computer, enabling you to repair file systems, edit files, etc. In this article, we'll show you how to get Solaris going in single-user mode, so you can do what needs to be done.

Before you start

You need to have a CD-ROM installed on the affected machine. You'll also need the Solaris installation CD-ROM (and boot disk if you're using Solaris x86). Finally, you'll need some time to work through the technique. On our system it took about 10-15 minutes to get to the single-user mode prompt. We checked out this method on a Solaris v2.4 system, so be warned that your mileage may vary on a different version.

The objective of this technique is to run through the installation program until the first opportunity to quit presents itself. Once we quit the Solaris installation program, we'll be in single-user mode, and we'll be able to perform any required system maintenance.

Let's get going

First, put the Solaris Installation CD-ROM in the drive, and shut down the system as well as you can. Then insert the boot diskette in drive A, and boot the computer. Eventually, the computer should display the Solaris Boot screen similar to the one shown in Figure A. Don't walk away from your computer during this first part, because this screen waits only 30 seconds for your response and then it will just boot Solaris from the disk device.

On our system, we enter *10* for the CD-ROM. On your system, the CD-ROM may have a different device code. Select the one that corresponds to the CD-ROM on your computer. When you do, the system will work for a while and then present you with the secondary boot screen, shown in Figure B.

Figure A

```
Solaris for x86 - FCS MDB                              Version 1.23

                  Solaris/x86 Multiple Device Boot Menu

            Code  Device Vendor   Model/Desc        Rev
            ========================================
             10)  CD     SONY     CD-ROM CDU-541   2.6a
             11)  NET    EtherExp I/O=300 IRQ=5
             80)  DISK   First IDE drive (Drive C:)

            Enter the boot device code: 10
```

You should select the CD-ROM device from the Solaris/x86 Multiple Device Boot Menu screen.

Figure B

```
Solaris 2.4 for x86              Secondary Boot Subsystem, vsn 2.11

                  <<< Current Boot Parameters >>>
Boot path: /isa/aha@330,0/cmdk@0,0:a
Boot args: /kernel/unix

Select the type of installation you want to perform:

        1 interactive
        2 custom JumpStart

Enter the number of your choice followed by the <ENTER> key.
```

At this point, you have five seconds to tell the installation kernel which boot method you want to use.

Figure C

```
Booting /kernel/unix...
SunOS Release 5.4 Version generic [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1994, Sun Microsystems, Inc.
WARNING: clock gained 552 days -- CHECK AND RESET THE DATE!

Configuring the /devices directory

Configuring the /dev directory

Stand By...
```

This part of the installation takes awhile because it's trying to boot Solaris from a slow CD-ROM.

Figure D

```
The Solaris Installation Program

You are now interacting with the Solaris installation program.  The
program is divided into a series of short sections.  At the end of each
section, you will see a summary of the choices you've made, and be given
the opportunity to make changes.

As you work with the program, you will complete one or more of the
following tasks:

  1 - Identify peripheral devices
  2 - Identify your system
  3 - Install Solaris software

About navigation...

  - The mouse cannot be used

  - If your keyboard does not have function keys, or they do not respond,
    press ESC; the legend at the bottom of the screen will change to show
    the ESC keys to use for navigation.

    F2_Continue       F6_Help
```

The first screen you encounter in the Solaris installation program is this introduction screen.

Figure F

```
Identify Graphics Devices

On the next screens, you must identify one or more of the following
peripheral devices that are attached to your system. This is necessary to
configure your window system for use during the Solaris installation
program.

      - Graphics card
      - Pointing device

You will not be asked to identify devices which have been identified
automatically.

> If you do NOT wish to configure your window system at this time, press
  F4. This will cause the Solaris installation program to run in a
  non-graphics mode. You will have another chance to configure the window
  system when the system reboots after installation, if you choose this
  option.

> To begin identifying devices, press F2.

F2_Continue     F4_Bypass Configuration     F6_Help
```

You'll press [F4] at the Identify Graphics Devices screen so you can get to the exit point more quickly.

Here, you want to select 1, for an interactive installation. You have only five seconds to answer, but there's no real hurry, as it defaults to 1.

Now the system will grind away for a few minutes, updating the screen with the information shown in Figure C. Don't be alarmed if the system seems to hang. As long as it's periodically reading from the CD-ROM, it's plugging along.

The italicized lines in Figure C don't show up immediately. Instead, you first see a slowly turning spinner. Some time later, the next message appears, and the process continues. Once the installation kernel finishes loading, the Solaris installation program starts.

Figure E

```
Keyboard Language

On this screen you must specify the language your
keyboard supports.

> To make a selection, use the arrow keys to high
  light the option and
  press Return to mark it [X].

    Keyboard language
    -----------------
 ^  [ ] German
 |  [ ] Italian
 |  [ ] Japanese(106)
 |  [ ] Japanese(J3100)
 |  [ ] Korean
 |  [ ] Norwegian
 |  [ ] Spanish
 |  [ ] Swedish
 |  [ ] Swiss-French
 |  [ ] Swiss-German
 |  [ ] Taiwanese
 |  [ ] UK-English
 -  [X] US-English

  F2_Continue     F3_Go Back     F6_Help
```

You use this screen to select the keyboard language.

Figure G

```
Confirm Information

> Confirm the following information.  If it is
  correct, press F2;
  to change any information, press F4.

      Keyboard type: AT keyboard
  Keyboard language: US-English

  F2_Continue     F4_Change     F6_Help
```

Now that you've described the relevant hardware in the system, you're asked to confirm it.

The first screen the Solaris Installation program presents is the introduction screen shown in Figure D. You may want to read it before pressing [F2] to continue.

Now we have to navigate through some more screens. Fortunately, they're pretty simple. First, we have to select the keyboard language, as shown in Figure E. Just press [F2] to advance to the next screen.

Next, the installation program prompts us to identify the graphics devices, as shown in Figure F. Just press [F4] to bypass the configuration. This allows us to skip a few screens.

Since we bypassed the graphics device identification phase, we immediately go to the first Confirm Information screen, shown in Figure G. Press [F2] to advance to the system identification phase.

Now the computer goes through a bit of activity, and the screen shows information as it processes. After a few moments, the computer presents the screen shown in Figure H. When you see it, press [F2] to continue.

Now, using the screen shown in Figure I, the installation process wants to find the host name of the computer. We really don't care what name is used as long as the name contains at least three characters, so we enter *Cobb* and press [F2] to continue.

The installation program next displays the Network Connectivity screen, shown in Figure J. On this screen, we'll indicate that there's no network connected. This allows us to bypass another screen. Just press the down arrow to highlight the No box, press [Return] to mark it with an X, and press [F2] to go to the next screen.

Now we're presented with the second Confirm Information screen, shown in Figure K. Just press [F2] to accept the information and advance to the Time Zone screen shown in Figure L.

Press [F2] to accept the United States setting, and the installation program will present the second Time Zone screen used to further refine the time zone your computer is in. We'll just press [F2] at this second screen to bypass it.

Now you set the time and date on your computer on the Date and Time screen shown in Figure M. Press [F2] to advance to the next screen.

Now we're at the final Confirm Information screen, shown in Figure N. Press [F2] to move to the next screen.

Finally, we're at the screen we've been waiting for, shown in Figure O. Here, we'll

Figure H

```
Identify This System

On the next screens, you must identify this system as networked or
non-networked, and set the default time zone and date/time.

If this system is networked, the software will try to find the information
it needs to identify your system; you will be prompted to supply any
information it cannot find.

> To begin identifying this system, press F2.

  F2_Continue     F6_Help
```

With this screen, the system identification phase starts.

Figure I

```
Host Name

On this screen you must enter your host name, which identifies this system
on the network.  The name must be unique within your domain; creating a
duplicate host name will cause problems on the network after you install
Solaris.

A host name must be at least two characters; it can contain letters,
digits, and minus signs (-).

  Host Name: _

  F2_Continue     F6_Help
```

It doesn't matter what host name you use as long as it's longer than two characters.

Figure J

```
Network Connectivity

On this screen you must specify whether this system is connected to a
network.  If you specify Yes, the system should be connected to the network
by an Ethernet or similar network adapter.

> To make a selection, use the arrow keys to highlight the option and
  press Return to mark it [X].

    Networked
    ---------
    [X] Yes
    [ ] No

  F2_Continue     F6_Help
```

You'll tell the installation program that there's no network so you can bypass another installation screen.

Figure K

```
Confirm Information

> Confirm the following information.  If it is correct, press F2;
  to change any information, press F4.

    Host name: Cobb
    Networked: No

  F2_Continue     F4_Change     F6_Help
```

The second Confirm Information screen allows you to confirm that the system identification information is correct.

Figure L

```
Time Zone

On this screen you must specify your default time zone.  You can specify a
time zone in three ways:  select one of the geographic regions from the
list, select other - offset from GMT, or other - specify time zone file.

> To make a selection, use the arrow keys to highlight the option and
  press Return to mark it [X].

   Regions
   --------------------------------
^  [ ] Asia, Eastern
¦  [ ] Asia, Western
¦  [ ] Australia / New Zealand
¦  [ ] Canada
¦  [ ] Europe
¦  [ ] Mexico
¦  [ ] South America
¦  [X] United States
¦  [ ] other - offset from GMT
-  [ ] other - specify time zone file

  F2_Continue    F6_Help
```

The first Time Zone screen allows you to modify the computer's time and date if necessary.

Figure M

```
Date and Time

> Accept the default date and time or enter new values.

Date and time: 02/07/96 16:54

  Year   (4 digits) : 1996
  Month  (1-12)     : 02
  Day    (1-31)     : 07
  Hour   (0-23)     : 16
  Minute (0-59)     : 54

  F2_Continue    F6_Help
```

You'll bypass this screen that lets you set the time and date on your computer.

Figure O

```
Upgrade System?

This system is upgradable. Choosing the upgrade option means any bundled
Solaris software will be updated to the new release, and as many local
modifications as possible will be saved.

While your system is upgradable, you can choose the initial otion; however,
files on your disk will be overwritten and data will be lost.

      CAUTION: If you choose the upgrade option, it is especially
      important to back up your system. However, backing up is also
      recommended for the initial option if there is any data on the
      disk that you want to save.

> To start the upgrade option, choose F2.

> To start the initial option, choose F4.

  F2_Upgrade    F4_Initial    F5_Exit    F6_Help
```

Here's the screen you were waiting for—the first one offering the Exit option.

press [F5] to exit the installation. When we do so, the program will ask you if you're sure. Just press [F2] again, and you'll exit the installation program.

You've done it!

Now you're done. You should see a # prompt on your screen showing you that the single-user version of Solaris is operational. You can now proceed to complete any maintenance tasks you need. ❖

Figure N

```
Confirm Information

> Confirm the following information.  If it is
  correct, press F2;
  to change any information, press F4.

      Time zone: US/Eastern
  Date and time: 02/07/96 16:54:00

  F2_Continue    F4_Change    F6_Help
```

You're finally at the last Confirm Information screen.

Marco C. Mason is a freelance computer consultant and author based in Louisville, KY. He's worked on cow feeding systems, automated destructive equipment testing, and the largest computer-controlled sound system in the world.

Netscape 2.0n ported to Solaris v2.5

The Netscape World Wide Web (WWW) browser, probably the most popular browser available for the DOS/Windows environment, is being ported to Solaris v2.5. The final version is now available from Netscape's home site.

Why should I be interested?

Right now, there are many manufacturers jockeying for market share in the WWW browser world. Currently, Netscape has the lead in the DOS/Windows world. One of the reasons is that Netscape has some proprietary extensions to HTML to improve the appearance of documents.

Since Netscape currently has the largest market share, you'll find that many Web pages use Netscape extensions and therefore won't look their best on other Web browsers. These pages usually have the tag phrase "This page looks best when viewed with Netscape." Thus, if the sites you're interested in visiting often have that tag line, you'll be interested in Netscape's browser.

Also, Netscape has been adding features as quickly as they can. For example, Java support, once the exclusive domain of Sun Microsystems, is now being supported by Netscape as well.

If you're running an x86 version of Solaris, you're out of luck. There's no x86 version of Netscape available, and it doesn't appear that one is coming soon.

Where can I get more information?

If you're interested in checking out Netscape's browser, you can visit their home page at http://home.netscape.com/. From there, you can track the progress of the latest version of their Web browser software. (If you're interested in Netscape's Web server software, you can also reach that information from the same page.) ❖

Locking out a user

Periodically, we hire consultants to set up applications and write programs for us. When the contract expires, we need to lock them out. How do we do this so they can't access the machine anymore?

On one occasion, we believe that we locked out the consulting account, but saw it logged in on a log. Did we just forget to lock it out?

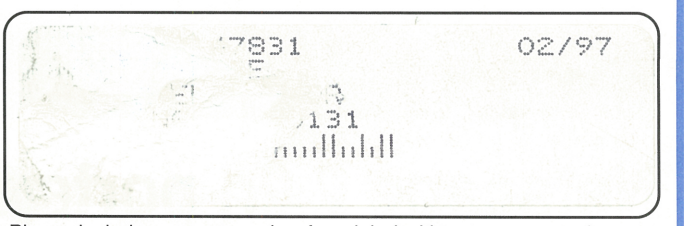
Ray Benjamin
Clearwater, Florida

It's not too difficult to lock out an account. First, log in as *root*, or issue the `su` command. If you're running OpenWindows, start a terminal screen. When you're at a shell prompt, enter the following command:

```
passwd -l consult
```

But that's not all there is to it. This simply prevents someone from logging in using the login prompt. If someone is logged in under that account and knows you're locking him or her out, he or she can simply run `passwd` and enter a new password. So you must first make sure that person isn't already logged in.

That's still not good enough. If you're on a network, the user may have access from a remote host. Therefore, you need to check



the user's home directory and either remove or rename the *.rhost* file if there is one.

To find the user's home directory, go to a shell prompt and list the */etc/passwd* file as shown in Figure A.

The highlighted text shows the consultant's home directory. So go to the */export/home/consult* directory and look for the *.rhost* file. Since the file begins with a period, you'll need to use the `-a` switch with `ls`, like this:

```
#ls -a
.               .Xauthority     .dt
..              .ab_library     .dtprofile
.rhost          .sh_history
```

In this example, the *rhost* file exists. Since we may call the consultant back for other projects, we don't want to delete the *.rhost* file; we'll just rename it to *old.rhost* so we can restore it when we fix the consultant's account, like this:

```
#mv .rhost old.rhost
```

Now we need to plug the last hole. Did the consultant leave any executable files or shell scripts that can set a new user ID? If so, then executing that program can provide the same rights he or she once had from another account. So let's use the `find` command to find all the files the consultant owns that have this permission bit set:

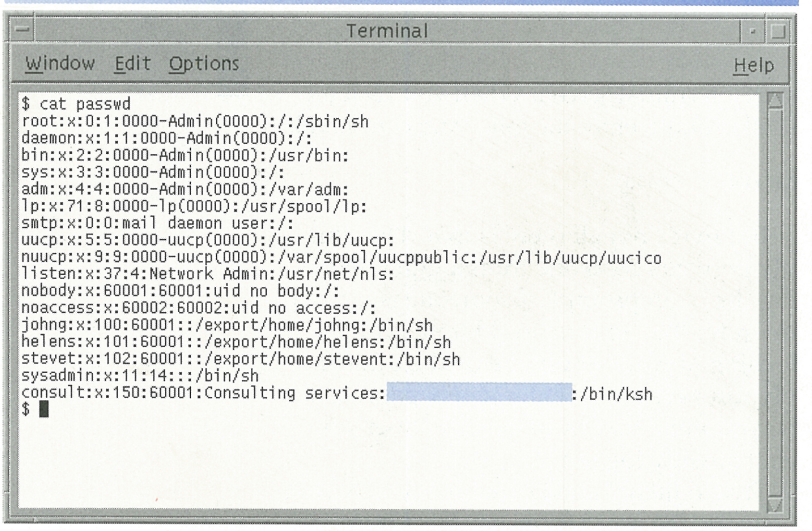
```
#cd /
#find -perm -04000 -user consult -print
```

Here, we start at the root directory looking for any files that have the setUID permission bit set (specified by the *-perm -04000* arguments) that are owned by the consult account (specified by *-user consult*), and we'll print them on the screen.

In this particular case, there were none. If they did exist, you'd need to decide whether to remove the commands or leave them alone, depending on whether you use them or not.

As you can see, we don't know whether you forgot to lock the password. If the consultant was a remote user, you might have left the *.rhost* file alone, enabling the consultant to log in remotely. If the consultant *really* wanted to get in, he or she could have left in a backdoor program or script file to grant access at a later time. ❖

Figure A



You can find any user's home directory by looking at the sixth field in the */etc/passwd* file.

We'd love to hear from you

If you've come across an interesting Solaris tip, have questions about articles you've seen in *Inside Solaris*, or have ideas for topics youd like us to cover in future issues, you can send mail to the

Editor-in-Chief, Inside Solaris
The Cobb Group
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Or you can reach us via the Internet at inside_solaris@merlin.cobb.zd.com.

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