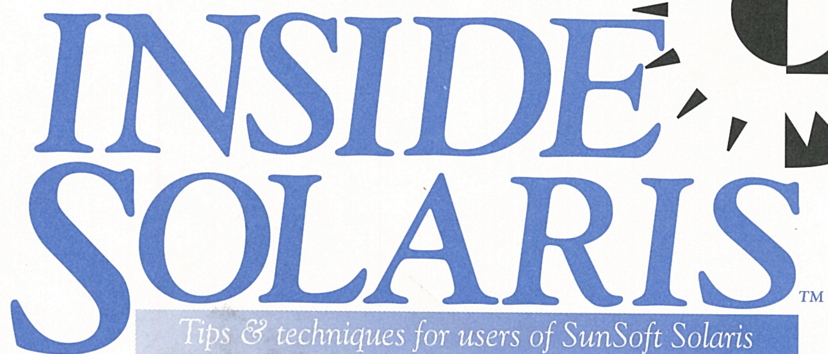
March 1998 • $11.50
Vol. 4 No. 3



IN THIS ISSUE

1
Installing and using Ghostscript

6
Inserting program output into your documents with vi

7
An easy way to modify 1,000 files

11
Making your shell prompt stand out

14
Running Wabi on 24-bit video displays

15
Attention Solaris x86 users!

16
Using du to determine the size of a subdirectory tree

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

Installing and using Ghostscript

No matter what we do, we can never get enough money into our budgets, can we? On the other hand, our users continually demand more and more from us system administrators. So how do we solve this dilemma? Why, find ways to economize, of course.

When we reviewed our budget, we searched for ways to get a job done at minimal cost. Two areas in which we could economize required that we find a way to render PostScript files. If we used a PostScript-rendering program, we could buy cheaper printers for some applications. We could also use less-expensive notebook computers.

Saving money with a PostScript renderer

How do you save money with a PostScript renderer? Sun puts its documentation in PostScript, which makes the documentation look good when you view it with AnswerBook, and it also prints nicely. The downside is that if you want to print any documentation, you'll need a printer that understands PostScript. Also, the AnswerBook depends on your X server providing Display PostScript.

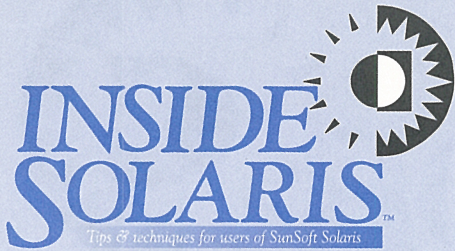
Printers with a built-in renderer are normally more expensive than ones without. So, if we could render PostScript on the computer, rather than making the printer do it, we could use less-expensive printers or printers with other features (such as color).

If you're buying SPARC-based laptops, the X server provided with Solaris should support the display. If you want to economize, you might want to use Solaris x86 on Intel-based laptops. If you do so, know that Solaris' X server doesn't support all possible laptop displays. For example, some Toshiba models seem to be best-supported under Solaris x86, while some other brands aren't supported at all.

If you'd like a larger range of laptop choices, you can use a different X server for Solaris. XFree86 is a popular X server used by the Linux and FreeBSD communities. This server supports many chipsets and boards, so it may be able to give you X on a laptop that's not supported by the Solaris X server.

As you've probably guessed, XFree86 doesn't support Display PostScript. Therefore, you can't view the AnswerBook documentation on your laptop. However, if you have a PostScript renderer, you can view your AnswerBook documentation—without Display PostScript.

ZD ZIFF-DAVIS a SOFTBANK company



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

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

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

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

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

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

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

Staff
Editor-in-Chief .... Marco C. Mason
Contributing Editors .... Al Alexander
Print Designer .... Marguerite Winburn
Editors .... Karen S. Shields
Joan McKim
Michael E. Jones
Publications Coordinator .... Linda Recktenwald
Product Group Manager .... Michael Stephens
Circulation Manager .... Mike Schroeder
Publisher .... Jon Pyles

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

Postmaster
Periodicals postage paid in Louisville, KY.
Postmaster: Send address changes to

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

Copyright
Copyright © 1998 The Cobb Group, a division of Ziff-Davis Inc. The Cobb Group, its logo, and the Ziff-Davis logo are registered trademarks of Ziff-Davis Inc. All rights reserved. Reproduction in whole or in part in any form or medium without express written permission of Ziff-Davis is prohibited. The Cobb Group reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for personal and commercial use. Information furnished in this newsletter is believed to be accurate and reliable; however, no responsibility is assumed for inaccuracies or for the information's use.

Inside Solaris is a trademark of Ziff-Davis Inc. Sun, Sun Microsystems, the Sun logo, SunSoft, the SunSoft logo, Solaris, SunOS, SunInstall, OpenBoot, OpenWindows, DeskSet, ONC, and NFS are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

What is a PostScript renderer?

What *is* a PostScript renderer, anyway? PostScript is actually a language that describes the information on a page. A PostScript-rendering program is a program interpreter that takes the PostScript program and turns it into an image of the page as an array of colored (if you've enabled color) dots.

One disadvantage of running a PostScript renderer on your host is that rendering a PostScript page is a CPU- and RAM-intensive process. When you buy a PostScript printer, you're letting the printer do all the work, saving your CPU and RAM for your computing tasks. For light use, a renderer is usually acceptable, but if you're thinking of using the PostScript renderer often on a heavily-loaded machine, you probably ought to go ahead and get a PostScript printer.

Where do you get it?

By the title of the article, you've no doubt surmised that we're going to use Aladdin Ghostscript as our PostScript-rendering package. Why did we select it? Since economy was our objective, the fact that Aladdin Ghostscript is freely available on the Internet was the deciding factor.

At the time of this writing, version 5.10 is the latest and greatest offering. You can get the latest source code from the primary FTP site *ftp.cs.wisc.edu*. in the */ghost/aladdin/current* directory. You can find more information about Ghostscript—as well as mirror FTP sites—at *www.cs.wisc.edu/~ghost*.

You'll need to get the source code archive for Ghostscript, the font archive, and the code for the JPEG, PNG, and zlib libraries. Our visit to the FTP site went like this:

```
/work> ftp ftp.cs.wisc.edu
Connected to piglet.cs.wisc.edu.
    •    •    •
User (piglet.cs.wisc.edu:(none)):
➥anonymous
331 Guest login ok, send your
➥complete e-mail address as password.
Password:
    •    •    •
230 Guest login ok, access
➥restrictions apply.
ftp> cd /ghost/aladdin/current
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data
➥connection for file list.
ghostscript-5.10.tar.gz
ghostscript-5.10jpeg.tar.gz
ghostscript-5.10libpng.tar.gz
ghostscript-5.10zlib.tar.gz
ghostscript-fonts-other-5.10.tar.gz
ghostscript-fonts-std-5.10.tar.gz
ghostscript-5.10gnu.tar.gz
ghostscript-5.10pc.tar.gz
    •    •    •
ftp> get ghostscript-5.10.tar.gz
ftp> get ghostscript-5.10jpeg.tar.gz
ftp> get ghostscript-5.10libpng.tar.gz
ftp> get ghostscript-5.10zlib.tar.gz
ftp> get ghostscript-fonts-std-
➥5.10.tar.gz
ftp> quit
221 goodbye.
```

We need only five of the files (the ones we italicized). For a larger selection of fonts, you can pick up *ghostscript-fonts-other-5.10.tar.gz* as well, but Ghostscript will work fine without the extra fonts.

How do you install it?

Now that you have the files, you need to prepare them for use. First, put them in a working directory so they don't conflict with any files already on your system.

Quick Tip: If you plan to download software from the Internet and build it, you'll probably want to have a permanent work area for just this purpose. Then, you can work with the files in a standard location. When you're finished, just clean out your work directory, and you're ready for the next project. We have a small file system mounted at */work* for this.

From the file extensions, you can see that the files are tar archives compressed with GNU's gzip. So, you must first expand the files and use tar to create the directory trees. On our system, we did it as shown in Figure A. (If this command seems obscure, take a moment to read the article "An Easy Way to Modify 1,000 Files" on page 7.)

Now we have the five tar files and five directory trees. It's time to get to work. Let's go into the *gs5.10* subdirectory and prepare to configure, compile, and install Ghostscript.

Building Ghostscript

Ghostscript runs on just about everything, so it has several makefiles that govern the build process. While we'll show you how to build Ghostscript, you may want to read the file's *Readme* for an overview of the system. Also check out *devices.txt* to know which printers Ghostscript supports, *make.txt* for information about how you can build Ghostscript on each different platform, and *use.txt* for information about how to use Ghostscript once you've compiled and installed it.

We've already gone through the effort of deciphering *make.txt* and installing Ghostscript on Solaris. Now, we'll present a condensed procedure for building and installing Ghostscript on your machine.

Which makefile should you use?

To configure, build, and install Ghostscript, you must first decide which makefile to use. To do this, you need to know how you're going to compile it. If you'll use Sun's Visual Workshop C++ or Workshop C/C++, you'll want to use *unixansi.mak*. If you're using GNU gcc, then choose *unix-gcc.mak*. To make things simpler later on, create a link from the makefile you selected to the name *makefile* like this:

```
/work/gs5.10> ln -s unix-gcc.mak makefile
```

In the next steps where we configure Ghostscript, we'll modify one of the files from which the makefile is built, then we'll rebuild the makefile. If you selected *unixansi.mak* as your makefile, go ahead and open *ansihead.mak* in your favorite editor. If you selected *unix-gcc.mak*, then open *gcc-head.mak* instead.

Check the third-party library paths

Ghostscript was written to use the JPEG, PNG, and zlib libraries provided by other parties.

Figure A

```
/work> for J in *; do
> gunzip $J
> tar xf ${J%.*}
> done
/work> ls
fonts                              ghostscript-fonts-std-4_03_tar
ghostscript-5.10.tar               gs5.10
ghostscript-5.10jpeg.tar           jpeg-6a
ghostscript-5_10libpng_tar         libpng-0.96
ghostscript-5_10zlib_tar           zlib-1.0.4
ghostscript-fonts-std-5.10.tar fonts
```

The first step in installing Ghostscript is to expand and extract the files.

Because they're not written by the Ghostscript team, the libraries aren't all placed inside the same tar file (the code in these libraries changes independently of Ghostscript). But Ghostscript needs them in order to compile, so we must link these directories to the location where Ghostscript expects to find them.

You can find out the names of the expected subdirectories by locating the definitions of the JSRCDIR, PSRCDIR, and ZSRCDIR macros. For example, in version 5.10, these macros are defined as

```
JSRCDIR=jpeg-6a
PSRCDIR=libpng
ZSRCDIR=zlib
```

Please note that these macros aren't defined right next to each other, so you'll have to use your editor's search mechanism to find each one. Referring back to the directory listing shown in Figure A, you'll notice that the names of the subdirectories containing the libraries don't exactly match these names. We could just link the existing directories to the names that Ghostscript expects, like this:

```
/work/gs5.10> ln  -s  ../jpeg-6a  jpeg-6a
/work/gs5.10> ln  -s  ../libpng-0.96  libpng
/work/gs5.10> ln  -s  ../zlib-1.0.4  zlib
```

Or, since we're in the editor, we can change the macro definitions to point to the appropriate directories, like this:

```
JSRCDIR=../jpeg-6a
PSRCDIR=../libpng-0.96
ZSRCDIR=../zlib-1.0.4
```

Configure the makefile for Solaris

The next step is to customize the makefile to work with Solaris. Solaris is based on UNIX System V Release 4, so we tell Ghostscript about it by locating the definition for CFLAGS and add

-DSVR4 to the end, as we do here:

```
CFLAGS=-O $(GCFLAGS) $(XCFLAGS) -DSVR4
```

Since Solaris stores the X server libraries in a non-standard (with respect to other UNIX systems) location, you must modify the XINCLUDE, XLIBDIR, and XLIBDIRS macros to reflect their locations on your system, as shown here:

```
XINCLUDE=-I/usr/openwin/share/include
XLIBDIR=/usr/openwin/lib
XLIBDIRS=-L/usr/openwin/lib
```

Various versions of UNIX have differing versions of the install command, so you need to change the INSTALL macro to use the old SunOS 4.x version found in */usr/ucb*:

```
INSTALL = /usr/ucb/install -c
```

Important note for GNU gcc users: If you're using gcc versions 2.7.0 through 2.7.2, you must modify the GCFLAGS macro to add -DCONST= and remove the -Wcast_qual and -Wwrite_strings directives, like so:

```
GCFLAGS=-Dconst= -Wall -Wpointer-arith
➥-Wstrict-prototypes
```

You may also want to add the directive -fno-builtin to XCFLAGS, or you'll see plenty of warning messages such as this one:

```
/usr/local/lib/gcc-lib/i486-sun-solaris2.5.1/
➥2.7.2.2/include/string.h:32: warning:
➥conflicting types for built-in function 'strcpy'
```

Which devices will you use?

Finally, select the output devices you want to use. If you choose the defaults, Ghostscript provides a diverse set of output devices, such as X Windows displays, HP DeskJets, LaserJets, PaintJets, Canon BubbleJets, Fax images, TIFF, and many others.

If you decide not to support some formats, you're also free to remove devices or add any devices that are on the list. (The documentation included with Ghostscript even provides information so you can write drivers for new devices, if you're so inclined.)

For the details on including and excluding particular drivers, refer to the file *devs.mak*, which contains a complete list of all available drivers, and *drivers.txt*, which contains documentation for some of the drivers. To include any particular driver, just add the driver name to one of the macro definitions starting with DEVICE_DEVS. Similarly, to exclude a driver, simply remove it from the DEVICE_DEVS line it's on.

As an example, suppose you don't want to include the Canon BubbleJet drivers. To exclude them, you'd find the appropriate DEVICE_DEVS macro and remove the drivers. In this case, DEVICE_DEVS6 contains the references to the BubbleJet drivers:

```
DEVICE_DEVS6=bj10e.dev bj200.dev bjc600.dev
➥bjc800.dev
```

Just remove the driver(s) you don't want. Since we didn't want any of them, we changed the line to the following:

```
DEVICE_DEVS6=
```

Compiling and installing

Now that the hard part is over, you can build and install Ghostscript. You can compile it with the command

```
/work/gs5.10> make
```

When you do so, your computer will grind away, compiling and linking until it either completes successfully or halts with an error. You shouldn't have any problems if you've followed these steps carefully.

We've tested this procedure on versions 4.03 and 5.10 and experienced no difficulty. On a lightly-loaded computer, it took slightly more than eight minutes to compile, so expect to wait a little while for the results.

At this point, test Ghostscript to see if it compiled correctly—and whether or not it's working. If you're running in an X windows session, that's as easy as typing the command

```
/work/gs5.10> ./gs tiger.ps
```

If all went well, Ghostscript should display a picture of a tiger on your screen, as shown in **Figure B**. (We also built a copy of version 4.03 and tested it with the file *golfer.ps*.) After Ghostscript draws the tiger, it will ask you to press the [Return] key, which tells it to go to the next page. Since the tiger is a one-page document, the program will end, leaving you at the Ghostscript prompt: GS>. Just type *quit* to return to your shell prompt. At this point, you're ready to install Ghostscript.

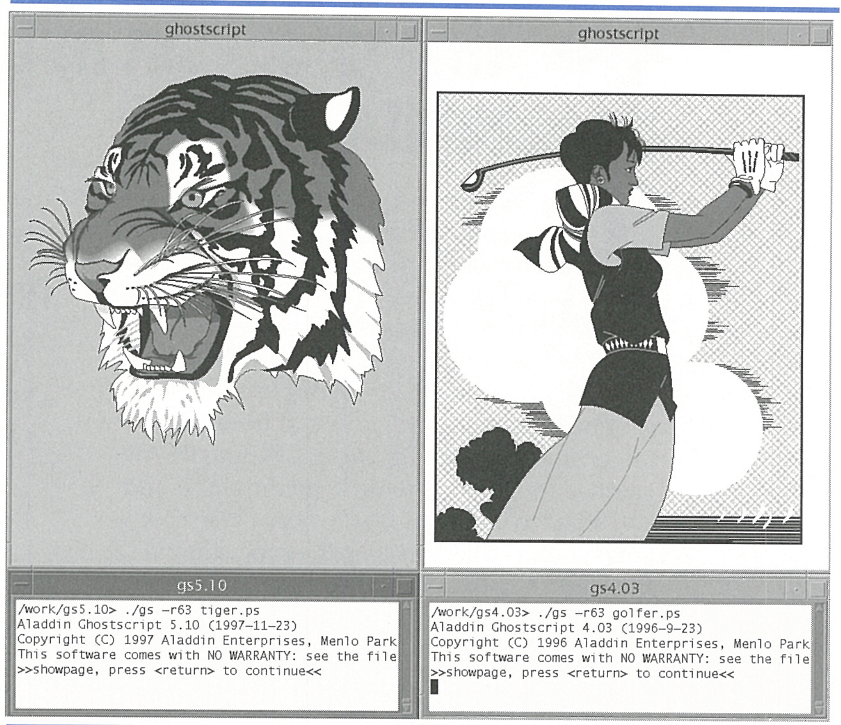
After you acquire root permissions, then you can install the fonts and install the program. You must place the fonts in the */usr/local/share/ghostscript/fonts* directory so that Ghostscript knows where to find them.

If you've installed X on your machine, then you'll also want to replace Ghostscript's default *Fontmap* file with the one that's customized for Solaris, named *Fontmap.Sol*. This way, you'll get access to the higher-quality fonts provided as part of the Display PostScript portion of the Solaris' X server. Finally, you can install Ghostscript just by telling `make` to build the install target with the following lines:

```
/work/gs5.10> su
Password:
# mkdir -p /usr/local/share/ghostscript
# mv ../fonts /usr/local/share/ghostscript
# mv Fontmap.Sol Fontmap
# make install
```

Installation takes a little while as the program copies executables, `man` pages, setup files, etc. to various locations in the */usr/local* directory. That's all there is to it.

Figure B



You can see the tiger if you built Ghostscript successfully.

Using Ghostscript

Now that we have Ghostscript installed, how do we use it? If you want to view a PostScript document, you can do so by telling Ghostscript the document name, as we did when we used it to draw the tiger. Just press the [Return] key to advance to the next page.

Printing a document is almost as simple. You follow the same process, with the addition of a few command-line switches, like so:

- Specify the driver: `-sDEVICE=`*`driver`*
- Send image to: `-sOutputFile=`*`out`*
- Exit when done: `-dBATCH`
- Don't wait for [Return] key after each page: `-dNOPAUSE`

So, if we have an HP LaserJet 4 printer connected to */dev/lp1*, we use the driver ljet4. Then, we can send the output directly to the printer with this command:

```
/work/gs5.10> gs -sDEVICE=ljet4
-sOutputFile=/dev/lp1 -dBATCH
-dNOPAUSE tiger.ps
```

Admittedly, the user interface isn't as good as it could be. However, Ghostscript is intended to be a PostScript renderer, not a documentation viewer or print manager, even though it can do both jobs by itself.

Next month, we'll show you how to install Ghostscript as a print filter. Then you can print your documents as if you were using a native PostScript printer. We'll also show you how to install GhostView to more easily navigate through PostScript documents on the screen.

Conclusion

At our site, we needed both a color printer and another laptop computer. However, it would've been hard enough to work just a laptop into the budget, much less a color laser printer. When you're browsing through the computer stores, you can't help but notice all the inexpensive color inkjet printers available, some for less than $200. We discovered that with Ghostscript and an inexpensive color inkjet printer, we could add color output for minimal cost. Be sure to read next month's issue where we tie our new inkjet printer into Solaris' print system. ❖

Inserting program output into your documents with vi

Have you ever been editing a file with `vi` and wanted to insert another document into the one you're currently working on? If so, you probably learned about the `:r filename` command, which reads the file *filename* and inserts it at the current location in your document.

Did you know that you can use this same command to run a program and insert the output of that program into your document? All you need to do so is to precede the command you want to run with `:r !`, where the exclamation symbol tells `vi` to start the program in a shell, and use the program's standard output stream as the file to insert.

You'll notice that `vi` inserts the resulting text into your document after the end of the line on which your cursor is currently located. Suppose you're writing a procedure for a user, and you want to show exactly the sort of results the user can expect to see on the screen. Using this little trick, you can do exactly that.

As an example, let's say you're editing a file, which contains the following two lines (where the blue square represents your cursor location):

```
Now is the time for all good men . . .
The quick red fox jumped over . . .
```

If you enter the command

```
:r !ls
```

then the file should now look something like

```
Now is the time for all good men . . .
vitip.doc
Ghostscript.doc
HTML_update.doc
for_loop.doc
The quick red fox jumped over . . .
```

You'll notice that the cursor is left at the beginning of the last line `vi` inserted for you.

The pièce de résistance

As it so often happens, there's another way to do the job. `vi` provides another command, `!!`, which executes the command you specify and replaces the current line with the program's output. However, you must have a line you're willing to sacrifice. (Often, that's no big deal).

While both of those commands are great, wouldn't it be nifty to have a command that would let you take part of your document, feed it to another command to process, and replace that part of your document with the results? It turns out that the `!!` command is really a special case of the more general command `!`, which has the form

```
! range command
```

where *range* specifies a range of lines, and *command* is the command to execute.

The *range* parameter isn't as flexible as it is in the colon commands, largely because `vi` will hand entire lines of text to your command—not just parts of lines. So while the colon commands allow you to specify ranges that select text on character boundaries, the range parameter for the ! command restricts you to line boundaries.

You may specify one of the basic cursor-movement commands, some of which you may precede with a count. Just for reference, some of the cursor-movement commands are shown in Table A, where *n* represents a number that defaults to 1 unless otherwise noted.

When you execute the ! command, `vi` starts a shell to execute `command`, then sends the text from your current cursor location to the location specified by *range* to that shell's standard input stream. The command may then do anything it likes with the text. `vi` captures the standard output stream of the program and replaces the selected text with it.

As you type in the command, `vi` gives you no visual indication of what you're doing until you complete the `range` part of the command. Once you complete a valid range, `vi` prompts you with a ! on the status line. Suppose you

Table A: *Cursor-movement commands*

Command	Description
/*p*, ?*p*	Search forward '/' or backward '?' for pattern *p*
*n*G	Go to line *n* (last line if not specified)
*n*j, *n*k	Go down 'j' or up 'k' *n* lines
n{, *n*}	Go back '{' or forward '}' *n* paragraphs
n(, *n*)	Go back '(' or forward ')' *n* sentences

have a table of 23 lines in your document that you'd like sorted. Just place your cursor on the first line of the table and enter *!22j*.

When you press the j, `vi` will prompt you for the command to enter—we'll use `sort`. At this point, `vi` takes the 23 lines you specified (the one you're on and the 22 following it), and sends them to the `sort` command, which then sorts the lines. After completing the sort, `vi` replaces your table with the sorted values.

Conclusion

The `vi` command has many surprising features hidden in its nooks and crannies. As you discover new features and become familiar with them, you can be that much more productive. `!` is a handy command you can use in many ways, such as writing user documentation, inserting `awk` reports into your documents, and including program output with a bug report. ❖

MASS PRODUCTION

An easy way to modify 1,000 files

by Alvin J. Alexander

With so many Internet and intranet servers in the world today, I've noticed an interesting phenomenon: The sheer number of HTML files on these servers plays right into one of the great strengths of Solaris systems—the ability to manipulate large quantities of text files en masse. Powerful Solaris batch-mode, command-line, text-editing tools, such as `sed`, `grep`, `awk`, `tr`, `cut`, and `paste`, make it easy to modify dozens, hundreds, or even more HTML files with a single command sequence.

You may be thinking "Why would I want to edit a thousand HTML files with one command?" Let's consider these situations:

- The name of a directory structure on your server changes.
- A business product line is renamed.
- The contact E-mail address on all your pages changes.
- You must remove unnecessary meta tags from HTML documents.
- You want to convert your HTML files to plain text format.
- File transfers from DOS/Windows systems to your Solaris system leaves ^M characters in your files.

In each of these situations, you'll need to make the same change—or series of changes—to a large number of files on your Web site. Using Solaris' powerful command-line tools, you can modify all of the files in one fell swoop. In this article, we'll show you how to automate the mass-editing process by using `sed` commands inside of a `for` loop to modify HTML files.

An overview of the process

From a high-level perspective, we'll perform the mass-editing of HTML files by placing our text-editing commands inside a continuous loop. The loop executes once for each file we want to modify. So if there are 1,000 HTML files to modify, the loop executes 1,000 times.

Inside the loop, we'll write our `sed` text-editing commands to operate on one HTML file at a time. Each time the `sed` program finishes modifying one file, the loop gives it another file to work on until all 1,000 files are modified.

There are two secrets to the success of this approach. The first is knowing how to create the continuous programming loop. Next, you must know how to use a good command-line, text-editing utility, such as `sed`, `awk`, or `Perl`. Let's examine the need for a good loop first. Then, we'll look at a text-editing utility.

A simple for loop

Many people use the shell's control-flow commands only in script files and don't think about using the control-flow commands on the command line. This is a double mistake: Since you can use these constructs on the command line, you can save yourself needless typing. You also

become more proficient with constructs as you use them, which gives you good practice for your shell scripts.

Just in case you're not familiar with the `for` statement, let's take a brief look at it. The `for` statement's purpose is to repeatedly perform a list of operations. The only difference between each set of operations is the value of a parameter—the loop variable. In the Bourne and Korn shells, the syntax of the `for` statement is

```
for identifier [ in word... ] ; do list ; done
```

Here, `for` tells the shell that you're starting the loop, and *`identifier`* is the loop variable. The next part, `[ in word... ]` lets the shell know which values to use for the variable *`identifier`*. (Please note that the brackets don't actually show up in the `for` statement; they indicate that the `in word...` clause is optional.) The `do` and `done` words tell the shell when the command list starts and stops, respectively, so *`list`* is the list of commands to execute.

Now, let's examine a simple problem that the `for` loop can help you solve. Assume that you have 100 HTML files in a directory, and you want to modify each file. Using the Bourne or Korn shell, you could use a command statement like this to make changes to every HTML file in the current directory using a `for` loop:

```
for htmlFile in `ls -d *html`
do
   echo "Converting $htmlFile ..."
   tmpFile=/tmp/$htmlFile.tmp
   sed -f cmds.sed $htmlFile > $tmpFile
   mv $tmpFile $htmlFile
done
```

Rather than specify the name of each file we want to manipulate, this statement first generates in the current directory a list of files that end with the characters `html`. It does this by including the `ls -d *html` command within the grave (`) characters at the end of the first line. (As you may remember, if you enclose a command within grave accents, the shell executes this command first, replacing the text between the grave accents with the results of the command.)

Please note: We used the `-d` option of the `ls` command to instruct `ls` not to list the contents of any subdirectories that match this search pattern. If you omitted this option and kept a backup copy of all your HTML files in a directory named backup.html, then our command would've changed the HTML files in the current directory *and* in your backup directory as well!

Next, the `for` loop works on each of these files, one at a time. Each time through the loop, the current filename (e.g., *index.html*) is stored in the variable `htmlFile`. For the convenience of whoever uses this code, an `echo` statement displays the name of the file currently being worked on.

Next, a temporary filename is created and stored in the variable `tmpFile`. As an example, if the variable `htmlFile` contains the string `index.html`, then `tmpFile` is assigned the string `index.html.tmp`.

In the fourth line of code, the `sed` interpreter reads the `sed` editing commands from the file `cmds.sed` and applies these commands to the input file `$htmlFile`. The `sed` command output is stored in the intermediate file `$tmpFile`. (Remember that `sed` never modifies the original file; it just writes its changes to standard output. Therefore, we're directing standard output to the filename stored in the variable `tmpFile`.) As the last line of the loop, you use the `mv` command to overwrite the original HTML file with the new temporary file.

This approach works great when all of your HTML files are contained in one directory. However, you'll usually want your changes to be propagated to every HTML file on your Web server. Since this can include a large number of subdirectories, we can't use the `ls -d *html` command to generate filenames any more. Instead, we'll generate a list of HTML filenames using the `find` command.

Combining the find command with the for loop

The `find` approach is almost identical to the previous code that used the `ls` command. The first change to the code is to use the `find` command to generate the list of HTML filenames, as shown in Listing A.

The `find` command checks the current directory, then every subdirectory, looking for text files that end with the extension html. The output of the `find` command is stored in the file named `list_of_html_files`.

The second change is the addition of the variables `numFiles` and `i`, and the modification of the `echo` statement. These changes are really just for the benefit of the end user, providing better status information as the program runs.

The next change to the code is the use of the command `cat list_of_html_files` to generate the list of files to change. Because the filenames are stored in this file with one filename per record,

the variable `htmlFile` is still assigned only one filename at a time.

The final change to the code is the creation and use of the new variable, `baseFileName`. Because `find` locates and prints files with their entire path prefixed to the filename, we must use the `basename` command to extract just the filename from the full path.

As an example, the `find` command might locate a file named *./products/software/Solaris/myApp.html* in your Web server directory tree. Inside the `for` loop, this full name—with the path included—is assigned to the variable `htmlFile`. Using the `basename` command, we extract just the base filename (*myApp.html*) and assign it to the variable `baseFileName` before using this name to create the `tmpFile` variable.

Creating a sed program

Now that we've created the necessary `for` loop, let's create the `sed` text-editing code. In this article, we'll keep our `sed` commands in a file named `cmds.sed`. Then we'll run the `sed` commands like this:

```
sed -f cmds.sed inputFile
```

Using this syntax, `sed` reads the commands in the `cmds.sed` file and applies these commands to the file named `inputFile`. It's important to remember that `sed` doesn't modify *`inputFile`*. The `sed` command just reads *`inputFile`* and prints the changes to standard output. It's the programmer's responsibility to redirect that output.

For our HTML programming example, let's assume that one of our company products, XYZ Widgets, was just renamed ACME Gadgets. In our large Web site, the name XYZ Widgets is included in hundreds of individual HTML files. Manually editing hundreds of files to make this change is definitely *not* on our top 10 list of things to do today!

A `sed` script to convert XYZ Widgets to ACME Gadgets must include two possible cases. In the first case, the full name XYZ Widgets will be on one contiguous line of an HTML file. In the second case, the character string XYZ will be at the end of one line, and the string Widgets will begin the next line.

The following line of code shows the `sed` command to replace the character string XYZ Widgets, when the string occurs on one contiguous line:

```
s/XYZ Widgets/ACME Gadgets/g
```

Listing A: *fix_html.sh*

```
#!sh
# Filename:  fix_html.sh
# Purpose:   Run a sed command on every HTML
#            file on our server.
# NOTE: Make sure you "cd" to your HTML
# "document root" directory here,  like:
# "cd /usr/local/etc/httpd/htdocs"

find . -name "*html" -type f -print >
   list_of_html_files
numFiles=`cat list_of_html_files|wc -l`
numFiles=`echo $numFiles|tr -d ' '`
i=0
for htmlFile in `cat list_of_html_files`
do
   i=`expr $i + 1`
   echo "Editing file $i of $numFiles:  $htmlFile ..."
   baseFileName=`basename $htmlFile`
   tmpFile=/tmp/$baseFileName.tmp
   sed -f cmds.sed $htmlFile > $tmpFile
   mv $tmpFile $htmlFile
done
```

This command changes the string XYZ Widgets to ACME Gadgets on every line of the given input file. Not only that, but the `g` at the end of the command tells `sed` to perform this change globally across every line. So if XYZ Widgets appears twice on one line of an HTML file, it will be replaced both times.

In a mass-editing situation such as this, we must also consider other possibilities. We can improve our search-and-replace technique in at least two ways. First, we can account for the possibility that there may be more than one space between the words XYZ and Widgets, generally resulting from a typing error. We can easily account for this possibility with the following change:

```
s/XYZ *Widgets/ACME Gadgets/g
```

Here, we've added an asterisk (*) after the space in the search string. An asterisk in a `sed` search string tells `sed` to look for zero or more occurrences of the character preceding the *. Therefore, this `sed` string will search for the characters XYZ, followed by zero or more blank spaces, followed by the characters Widgets. (This also covers the possibility of XYZWidgets, with no spaces between the two words.) When `sed` finds any string that matches this pattern, it replaces that string with the new string, ACME Gadgets.

The second improvement to our search string is to recognize that the search string may

also be used in its singular or possessive forms, like this:

```
XYZ Widget
XYZ Widget's
```

After looking at these possibilities, it's apparent that we're better off changing XYZ Widgets to XYZ Widget, and ACME Gadgets to ACME Gadget. This subtle code change—omitting the letter s at the end of each string—lets us account for the singular, plural, and possessive forms of our search string.

In summary, these two changes turn our sed command from this:

```
s/XYZ Widgets/ACME Gadgets/g
```

into this:

```
s/XYZ *Widget/ACME Gadget/g
```

When the new command is applied, it will make the following changes to the search strings it finds:

```
XYZ Widget becomes ACME Gadget
XYZ    Widget becomes ACME Gadget
XYZWidget becomes ACME Gadget
XYZ Widgets becomes ACME Gadgets
XYZ Widget's becomes ACME Gadget's
```

Multiline pattern-matching

The next circumstance to account for occurs when the string XYZ appears as the last string on one line of an HTML file, and the word Widget appears as the first word on the next line. This is a common occurrence, especially with HTML code-generators, that you must account for. The sed command to manage this multiline possibility looks like this:

```
/XYZ *$/ {
N
s/XYZ *\n *Widget/ACME Gadget/g
}
```

This important command accounts for several multiline possibilities. First, it searches for the string XYZ followed by zero or more blank characters at the end of one line. (The $ character symbolizes the end of the line.) Then, if this string is found, code execution moves inside the curly brackets.

Within the curly brackets, the N, or *Next*, command is called. The N command reads the next line of text from standard input and appends it to the current line of text in the buffer. As an example, if the line of text currently in the buffer contains the string

```
this is line 1 - XYZ
```

and the second line contains the text

```
this is line 2
```

the N command would merge both of those lines in the current buffer, like this:

```
this is line 1 - XYZ\nthis is line 2
```

You'll notice that the N command leaves the newline character (\n) in the text. After the N command performs this merge operation, our search pattern accounts for the embedded newline character by including the special \n symbol.

The completed sed code

Listing B shows the final sed program. It combines the two search-and-replace commands we just developed with a few comments for good measure.

Listing B: *cmds.sed*

```
# cmds.sed
#
# 1. search-and-replace single-line occurrences
#
s/XYZ *Widget/ACME Gadget/g
#
# 2. search-and-replace multi-line occurrences
#
/XYZ *$/ {
N
s/XYZ *\n *Widget/ACME Gadget/
}
```

Now, when you're ready to clean up your HTML files, you can run the Bourne shell program, shown in Listing A, like this:

```
fix_html.sh
```

The Bourne shell program calls the find command to generate the list of files you want to modify, invokes the for loop, and runs your sed program on each file in the list.

A few other sed examples

The next two sections provide a few other sed batch-mode, text-editing examples designed to stimulate your thought processes. We'll look at

two examples. The first shows how you can change the E-mail response addresses (typically prefaced with a tag like mailto:), and the second shows you how to delete extraneous HTML tags that you don't want in your documents.

Example 1: Modifying mailto: addresses

The following `sed` command shows how you can make wholesale changes to mailto: addresses when the need arises. Assume for a moment that you need to change a mailto: E-mail address from *fred@xyzcorp.com* to *webmaster @xyzcorp.com*. Maybe Fred left the company, or you think this anonymous approach is better. In any case, you must change the mailto: URL from fred to webmaster:

```
s/mailto:fred@xyzcorp\.com/
➥mailto:webmaster@xyzcorp.com/g
```

Depending on the other wording on your Web site, you may also need to change additional occurrences of *fred@xyzcorp.com* to *webmaster@xyzcorp.com*. Here, you might want to make your search-and-replace strings a little less specific:

```
s/fred@xyzcorp\.com/webmaster@xyzcorp.com/g
```

Notice that you should use the backslash character before the dot character in the .com portion of the search string. Because the dot character is a `sed` wild card that matches any *single* character except the newline character (similar to the ? character with filename pattern matching), your search string will work without the backslash. However, you leave the door open to match something else, too. In reality, it won't hurt you in this example, but be careful in other less-specific commands.

Example 2: Deleting meta tags

One administrator I met liked to delete from his HTML files the meta tags that advertised the name of the software program (Netscape, FrontPage, etc.) used to generate the file. He didn't care for this form of free advertisement, so we devised a `sed` program to delete these particular meta tags:

```
/^<meta name="GENERATOR" .*>$/d
/^<META NAME="GENERATOR" .*>$/d
```

Note that we use two `sed` commands—one to eliminate lowercase usage and another to eliminate uppercase usage. We can also accomplish the same thing with this code:

```
/^<[Mm][Ee][Tt][Aa] [Nn][Aa][Mm][Ee]="GENERATOR"
➥.*>$/d
```

Conclusion

Obviously, if you're not comfortable with `sed`, `awk`, or `Perl`, using them can be as dangerous as it can be helpful. As always, *back up your original files before making any wholesale changes*. Next, always test your `sed` program thoroughly on a few sample HTML files before running it on your entire Web site! As HTML text files continue to proliferate on Web servers, the powerful text-editing commands Solaris provides can save you a great deal of time when you must make repetitive changes to a large number of files across your entire Web site. ❖

PROMPT TRICKS

Making your shell prompt stand out

Why would you want to change your shell's prompt? Well, you might dislike the default value. Or, maybe you're working with large, busy screens, and it's too easy to lose your prompt. Perhaps you work on multiple machines, and you'd like your prompt to indicate which machine you're on. It's also conceivable that you don't remember exactly which directory you're in, and you'd like your prompt to reflect the location.

As you can see, it's often a blessing that you can easily change your prompt to suit your needs. In this article, we'll briefly describe how to change your prompt to fit your requirements.

Customizing your prompt

The authors of the various shells remembered that people like to customize their working environments. For this reason, all the popular

shells offer a simple method to change your prompt. For the Bourne shell and its derivatives, you simply set the PS1 environment variable to the prompt you want. With the C shell and its derivatives, you use the prompt environment variable.

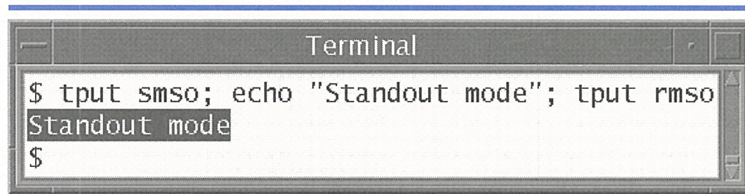
Therefore, to change your prompt, you just set the appropriate variable and export it if you wish child shells to use the same prompt. In the Korn shell, we can change our shell with this code:

```
$ PS1="`hostname`> "; export PS1
Devo>
```

Using terminal features

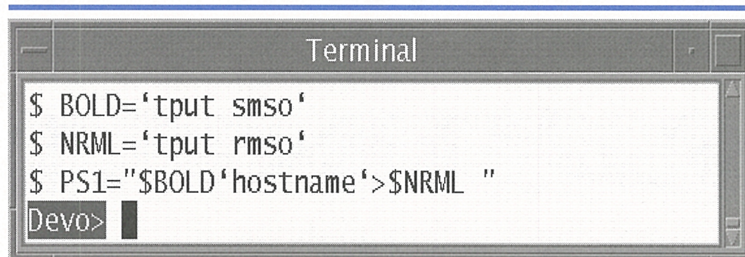
To make your prompt stand out, you can take one of two paths. If your work includes moving from terminal to terminal, you can go for portability. On the other hand, if you remain at the same terminal, you might customize your prompt to take advantage of any special features offered by your terminal.

Figure A



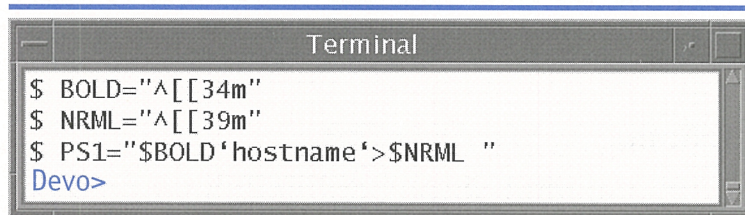
The tput command offers a standard set of terminal customization commands.

Figure B



You can make your prompt stand out using escape sequences garnered from tput.

Figure C



Taking advantage of the special character sequences in the dtterm window results in a blue prompt.

If you want portability, you should familiarize yourself with the tput command, which allows you to put your terminal in one of several standard modes. While these standard modes may appear different on various terminals, at least they'll be handled rationally.

The standout mode is one standard mode. The tput command provides the set standout mode command (abbreviated smso) and the reset standout mode command (rmso). Figure A shows a brief example of using tput to print text in standout mode.

To build a customized prompt, we just get the proper escape sequences with tput and use them to build our prompt. Thus if we're using the Bourne shell and we want to make our prompt contain our host name and stand out, we can use the statements shown in Figure B.

Instead of making a single, overly complex statement, we used three steps. First, we put the escape sequence to put the terminal in bold mode into the variable BOLD. Then, we placed the escape sequence to return the terminal to normal mode in the variable NRML. Once we've completed those two steps, the third step is clear: We use the BOLD and NRML variables to bracket our prompt. At this point, we built our new prompt with these variables and used the hostname command to find the name of the computer where we're currently connected.

The disadvantage of this technique is that in order to be as standard as possible, the tput command doesn't necessarily offer you all the features available in your terminal. If you really want some fancy customizations, you can abandon any portability concerns, whip out the manual for your terminal, and make any fancy prompt you want.

For example, if you use the CDE dtterm window as your terminal, you might want to use colors on the screen. By reading Section 5 of the dtterm manual, we learn that the escape sequence for making text blue is

```
[Escape] [ 34 m
```

and for returning the standard text color to the default is

```
[Escape] [ 39 m
```

where [Escape] is the ASCII value ESC (0x1b). Thus, we can use the commands shown in Figure C to make the prompt blue, thereby making it easier to find.

Please note that to get the [Escape] (^[) in the BOLD and NRML variables, you first press [Ctrl]V followed by the [Escape] key. Otherwise, you won't get the desired results.

Using shell features

The standard Bourne shell provides no special features for making your prompt dynamic. However, the Korn shell does provide one such feature: It expands the prompt string before printing it. This allows the Korn shell to display any environment variables that change during normal use. For example, if you include the string $PWD in your prompt as shown in the following code, then the Korn shell will replace it with your current working directory:

```
$ PS1='$PWD> '; export PS1
/export/home/marco>
```

Please note: If you use this trick, you *must* quote your prompt with single quotes, or the shell will expand the environment variable *before* assigning it to your prompt. And that won't work at all. For example, in the Korn shell, the variable PWD tells you the current directory that you're in. So if you want to create a prompt that shows you the current directory, you'll want to put the variable $PWD into your prompt, as follows:

```
$ PS1="$PWD> "; export PS1
/export/home/marco> cd /
/export/home/marco> pwd
/
/export/home/marco>
```

Hey! What happened? Here, we used quotes so the space at the end of the prompt was preserved. However, since we used double quotes, the Korn shell expanded the PWD variable *before* assigning it to PS1. So, in effect what we've done is execute the statement

```
$ PS1="/export/home/marco> "; export PS1
```

If we used single quotes, then PS1 would have a reference to the PWD variable in it. The Korn shell would expand it to give us the current directory at every prompt as a result of the lines

```
$ PS1='$PWD> '; export PS1
/export/home/marco> cd /
/>
```

Alternate prompts

When you're configuring your prompt, don't forget about the other prompts. The Bourne shell has two prompts: PS1, the primary prompt that we've been playing with, and PS2, the secondary prompt. You'll see the PS2 prompt, which defaults to >, when you enter a statement and the shell determines that you haven't completed the command. For example, when you enter the following for loop, the Bourne shell prompts you for the second and third lines using the PS2 prompt:

```
$ for J in 1 2 3; do
> echo $J
> done
1
2
3
$
```

When you press the [Return] key after do, the shell prompts you with a > prompt. It knows that the for loop isn't completed yet, because it hasn't seen the done statement. Only after you enter the done statement will the shell execute the for statement and give you another primary (PS1) prompt.

The Korn shell adds two more prompts. The PS3 prompt, which defaults to #?, is used to prompt you for the select statement. The PS4 prompt is used as a prefix for each line used in an instruction trace, when you're debugging shell scripts. Since other shells (bash, tcsh, etc.) may have additional prompting conventions, you must carefully check the man page for the shell you're using. ❖

Are you a good tipper?

Do you have any great Solaris tips that you've discovered? If so, send them our way! If we use your tip, it will appear on our weekly online ZDTips service. (Visit **www.zdtips.com** to check out all our available tip services.) We may also publish it here in *Inside Solaris*. Your byline will appear with the tip, along with your E-mail and/or Web addresses.

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

Inside Solaris
The Cobb Group, Suite 300
9420 Bunsen Parkway
Louisville, KY 40220

Running Wabi on 24-bit video displays

I don't know about you, but I prefer to run my video card with as many colors and as many pixels on the screen as possible. This way, I can see a great deal of my project, and my graphics applications don't show annoying color banding.

Until recently, however, "as many colors as possible" amounted to 256. You see, I had a relatively inexpensive video card in my machine. When I upgraded my video card, I could display more pixels on the screen, as well as get 24-bit color depth. My definition of paradise consists of—1280x1024 resolution with 24 bits on a 19-inch monitor!

A few days after settling in with my new screen and monitor, I started Wabi so that I could run a Windows program. However, instead of seeing the Windows desktop on my screen, I saw this message from Wabi in the xterm from which I tried to start Wabi:

```
/export/home/marco> Wabi
Starting WABI...
    •   •   •
Unrecoverable Error in Wabi

Unsupported display depth 18 in init_gdi.

Wabi will exit now.
```

Paradise lost. I've got a great display, but I can't run Wabi. From the error message, it appears that Wabi doesn't like my 24-bit video display. At this point I closed X windows, started a command-line session as root, and ran `kdmconfig -u` to clear the display configuration. I then ran `kdmconfig -cf` to select a new display configuration with only 8-bit video. When I restarted Wabi, it came up successfully.

While this procedure let me run Wabi successfully, I bought this hardware to make my life simpler, not more difficult. Reconfiguring my video display whenever I want to start and stop Wabi won't simplify my life. Using the same old video modes doesn't cut it either. When I brought up the subject with Sun's technical support representative, he said "Oh, it looks like you need patch 103587-03. This patch brings Wabi up to version 2.2D and includes support for 24-bit video displays."

Patching Wabi

Before Sun stopped development on Wabi, it cleaned up several issues, bringing the latest (and last) version to revision 2.2D. To update your version of Wabi to 2.2D, you'll need one of these patches:

- 103586-03 for Solaris SPARC
- 103587-03 for Solaris x86
- 103588-03 for Solaris PPC

Just go get the appropriate patch file from Sun's patch page or your SunPatches™ CD-ROM. From there, it's a simple install. First, you must uncompress the patch file. If you obtained the patch from the Internet, it's compressed with *compress*. The file has a .Z extension, so you can `uncompress` and extract the patch files with the commands

```
root# uncompress 103587-03_tar.Z
root# tar xf 103587-03_tar
```

On the SunPatches™ CD-ROM, Sun uses GNU's `gzip` to compress the patch files. Since `gzip` gives better compression ratios, Sun can place more patches on the CD-ROM. (See the article "Compressing Your Files to Save Disk Space" in the February issue.) In this case, you can copy the `gzcat` program from the CD-ROM's *gzip/bin/platform* directory to your site's local bin directory, then expand and extract the patch file with the command

```
root# gzcat 103587-03_tar.gz | tar xf -
```

Now, move to the subdirectory that `tar` just created, and ensure that no one's using Wabi by checking the running processes, as follows:

```
root# cd 103587-03
root# ps -ef | grep wabi
root 6860 6846  0 07:00:12 pts/4  0:00 grep wabi
```

If you see any processes other than `grep`, then you must tell all users to exit Wabi. Once they do and you verify that it's not running, you can install the patch with this command:

```
root# ./installpatch .
```

When the patch is installed, start Wabi. You should see that the startup banner in the xterm window shows the version 2.2D and that the new splash screen is much more colorful. If this is true, then you've successfully installed the new Wabi patch, and you can let people use the system again.

Notes

The Wabi patch files described here work for Solaris 2.4, 2.5, and 2.5.1. When Sun released Wabi 2.6, it changed the *pkginfo* file slightly. While you can still upgrade Wabi, you first must edit the file */var/sadm/pkg/SUNWwabi/pkginfo* and remove the ",REV=6" from the line starting with `VERSION=`. Once you do so, you should be able to install the patch upgrade with no problems.

An alternative (and slightly more difficult) solution is to remove the SUNWwabi package and install the version from Solaris 2.5 or 2.5.1, then install the patch. (We recommend the first solution, but if Sun comes out with a newer patch that's incompatible, this method may still work.)

Another potential problem for Solaris x86 users is that there are reported interactions between this patch and the one used to correct the Pentium F00F bug described next in "Attention Solaris x86 Users!" Some users report that version -01 of the Pentium patch causes Wabi to reboot the system when you start it. Hopefully, by the time you read this, Sun will have released a new version of the Pentium patch to correct this bug.

Conclusion

Well, I can now switch between applications without random—and annoying—changes to other applications' palettes. I can also view complex graphics without color banding at the same time that I run a Windows program. Paradise regained. ❖

RELIABLILTY REPORT

Attention Solaris x86 users!

The fact that the Pentium had a huge, well-publicized bug (the old division bug) surprised few, if any, engineers. The Pentium chip is a very complex system. It's probably impossible for anyone to create a system that complex without it displaying *some* faults. There are several bugs logged against the Pentium chip—most of which are very minor. You probably won't run into any unless you're creating a new chipset or motherboard to interface with the Pentium.

What's the new bug?

Recently, we've become aware of a newly discovered Pentium bug. This one is severe—it can hang *any* Pentium-based system. Although the explanation of how the bug causes a system to crash is very complex, the bug itself is far too simple to re-create. You can do so by simply executing this sequence of bytes:

```
0xF0  0x0F  0xC7  0xC8
```

Intel confirmed this bug soon after it was discovered, identifying it on November 7, 1997. Since that time, Intel has worked around the clock with OS vendors to find a workaround. As a result of this collaboration, Sun released a patch to prevent the problem in late November.

Give us the technical details!

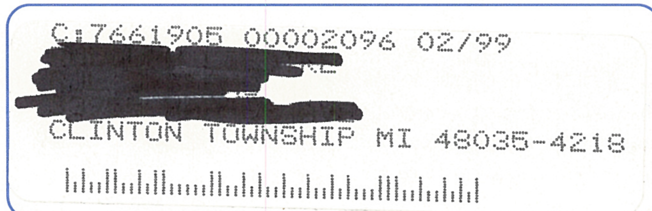
Basically, an instruction named `CMPXCHG8B` compares a 64-bit value in one of the CPU registers (the source) against a 64-bit value in RAM (the destination) and exchanges the two values. Using a register value for the destination operand is illegal.

The bug occurs if the offending code first uses the `LOCK` prefix (the 0xF0) to lock the CPU's bus, then executes the `CMPxCHG8B` instruction with a register destination. At this point, the CPU issues an Illegal Instruction exception. However, with the CPU bus locked, it can't service the interrupt, and the CPU hangs.

Intel devised a pair of workarounds. Basically, both workarounds put the first seven

SunSoft Technical Support
(800) 786-7638

PERIODICALS MAIL



Please include account number from label with any correspondence.

entries of the IDT (Interrupt Descriptor Table) into a non-writeable page. This way, when the Illegal Instruction exception arrives, the CPU issues *another* exception (Page Fault) because the IDT entries are in non-writeable RAM. The CPU may then execute code in the Page Fault handler to detect the problem and terminate the offending application.

How will this bug affect you?

Fortunately, this bug affects only Pentium-based computers, both with and without MMX. Anything older (386, 486) or newer (Pentium Pro, Pentium II) is unaffected. Because of the nature of the code sequence that's used to force the bug to occur, it's highly unlikely that any applications actually contain the necessary code to cause this problem. If you encounter this bug in an application, it's quite probable that someone placed it there intentionally.

The patches you need

This bug has the potential to cause so much trouble on systems. Not because any known applications have the bug, rather, it's so simple to evoke that any curious user (malicious or otherwise) will have no trouble trying it out. If they try it on a mission-critical system, it has the potential to be very serious. In order to prevent this disaster, Sun put the following patch files on its publicly-accessible patches page at *http://sunsolve.sun.com/pub-cgi/us/pubpatchpage.pl*. You'll need one of the following patches, depending on which version of Solaris x86 you're running. As always, be sure to get the latest version of any available patch, as Sun may improve the patches without notice:

- 105640-01 for 2.5
- 105638-01 for 2.5.1
- 105639-01 for 2.6

Notes

Please note that the initial release of the Pentium bug patch has a fatal interaction with Wabi 2.2D. When you start Wabi 2.2D after installing this upgrade, your system will reboot. So, if you run Wabi, you must choose between installing this patch and running the latest version of Wabi. (Also, this patch may interact with other versions of Wabi as well. We haven't tested this, nor have we seen any reports concerning other versions.)

Conclusion

If you're running a mission-critical Pentium system, be sure to keep everyone off the system but your most trusted personnel. It's just too tempting for someone armed with this information to resist trying it out. (Don't you ever get the urge to crash your workstation?)

Be sure you install the appropriate patch on a backup system, and test all your applications to verify that they still operate as needed. Once you've verified that the patch works with your system, then install it on your machine. Alternately, you may decide that this is the right time to upgrade your server(s) with meatier machines and invest in a few Pentium IIs. ❖

Using du to determine the size of a subdirectory tree

In some operating systems, it's hard to obtain directory information. For instance, you may want to know the size of a subdirectory, including all of the files contained in each of its subdirectories. To determine the size of an entire subdirectory tree, use the Solaris "disk usage" command, `du`. Include the `-k` option for the result in kilobytes and the `-s` option for only a summary listing. For instance, on our Solaris 2.5 system, this command

```
du -ks /usr/local/bin
```

yields this output:

```
6654     /usr/local/bin
```

This indicates that more than 6MB (6,654KB) is stored in this directory tree.

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