

Inside Solaris™

Tips & Techniques for users of Sun Solaris

A DNS troubleshooting tool

by Atiq Hashmi

The nslookup program is very useful in finding the IP address of any host on the Internet. It's also very useful in debugging Domain Name System (DNS) problems. Sometimes, troubleshooting can be done by running nslookup as a name server for doing lookups. Doing this manually on nslookup can be tedious and time-consuming. This article offers a tool that makes this kind of DNS troubleshooting very convenient.

A quick DNS lookup overview

The DNS system is based on the client and server model shown in **Figure A**. The client that needs to look up a domain name (for example, *www.zdjournals.com*) is called the resolver, and the server that's resolving the

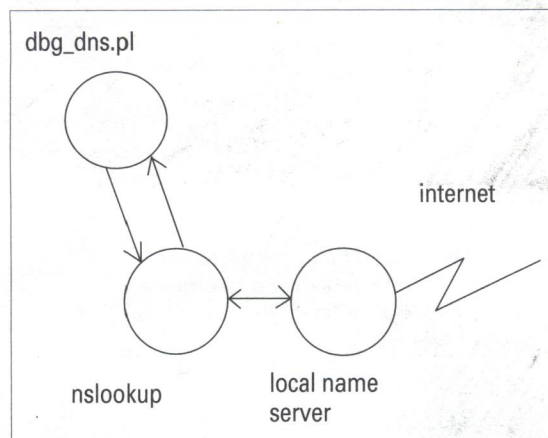


Figure A: The DNS communication flow is shown here.

domain, called the Name Server (NS), goes to the Internet to find the IP address of the domain. A resolver sends specific queries to the NS, passing the domain name as the argument.

There are two kinds of queries: recursive and non-recursive (or iterative). In both cases, the name server first checks to see if it has the information in its cache; then it returns that result. In a recursive query, the resolver sends the query with a domain name, and the NS goes out to the Internet and talks to other name servers on behalf of the resolver, eventually returning the IP address to the resolver. In a non-recursive query, the name server doesn't do all of the work; rather, it returns a list of name servers (known as referrals) that it thinks can provide the answer. The resolver then sends a similar non-recursive query to each name server in that list until it gets the answer.

Normally, when you run nslookup, it runs as a resolver and sends a recursive query to the local name server. The local name server, in turn, temporarily acts as a resolver and sends non-recursive queries to the other name servers in the DNS tree, so that it contacts each referral in a depth-first fashion until it finds a name server that returns the IP address.

An example of domain lookup using nslookup is shown here. Our keystrokes are in **color**. The `<host.company.com>` will be the actual local name server (the `>` is the nslookup prompt):

DOWNLOAD
ftp.zdjournals.com/sun

In this Issue:

1
A DNS
troubleshooting tool



5
MakeIndex: a tape utility

7
Managing licenses

8
Backup made easy

11
The ssh: what it is
and why you need it

14
Solaris Q & A:
Why does traceroute
work on my Windows
box, but fail on my
Sun box?

15
Quick Tip:
Easily convert
man pages to
text documents

16
Inside Solaris
reader survey


```
$ nslookup
Default Server: host.company.com
Address: 128.69.113.172
```

```
> www.zdjournal.com
Server: host.company.com
Address: 128.69.113.172
```

```
Name: www.zdjournal.com
Address: 152.175.1.108
```

```
>exit
```

Here, nslookup as a resolver sends a recursive query (which is default), and the server system host.company.com resolves the query with a crisp answer containing the IP address. However, this session doesn't tell the whole story as to how the name server went about resolving the query.

During troubleshooting, it's necessary to look under the hood and see what intermediate name servers are being used. In other words, we need to be able to see like a name server. This is where nslookup comes to the rescue. Nslookup can emulate either a resolver or a name server. We saw how it emulated a resolver above. To emulate a name server, it can be invoked with these options:

```
$ nslookup -norecurse -nosearch
```

The -norecurse tells nslookup to send non-recursive queries to the local name server. The -nosearch options tells it to not use the search-list because name servers don't. When you


enter a domain in this mode, the default name server returns a list of name servers. We then tell nslookup to use one of those name servers as the default server and look up the domain again. Repeating this process eventually returns the IP address.

Tool design

We need to do this kind of troubleshooting repeatedly, but for many domains it will be very inefficient and difficult. Our tool automates this activity in a Perl script. The idea is to invoke nslookup in a child process and set up a communication channel between our Perl script and nslookup. Thus, instead of us doing all the tedious work, the parent process takes the domain name and talks to the nslookup process and finally shows the path it traversed along the DNS tree, showing each name server it had to contact. **Listing A** shows the Perl script that implements this, and **Figure A** shows how the communication process is set up.

The tool has several options. You can perform inverse lookups (that is, enter an IP address and get the domain name). Also, you can provide it a list file of domains for a batch job. **Listing B** on page 5 shows sample runs of the tool. The dbg_dns.pl is the name of the tool. Use a higher verbose level to see more output.

Conclusion

This tool provides a means to perform a tedious task conveniently and allows you to debug problems by showing the DNS tree path being taken to resolve a domain name. 

Listing A: Require "getopts.pl";

```
# This script shows the path of DNS tree followed to
# obtain the ip address of a domain.
#
$dom=$ARGV[ $#ARGV ];
$cnt=0;      #starts at 0 upto $cnt, so total NSs = $cnt+1
$IP="";
$defserver="";

# Expected patterns for matching; Change if needed.
$NED="Non-existent domain";
$NRFS="No response from server";
$SB="Served by:";
$AAFF="^Authoritative answers can be found from:";
$NS="nameserver = ";
$N1="Name:";
$Addr="Address: ";
$Addrs="Addresses:";
$N2="name = ";
$NA="(non-authoritative)";
$DS="Default Server: ";
```

```
$usage="usage: $0 [options] [domain]
<options> :
-v<level> : verbose level 1-5
-i         : inverse query
-f         : force (use domain as given)
-l<file>   : path of the listfile, one per line
<domain>  : domain e.g. www.whitehouse.gov \n\n";
```

```
sub getOpts
{
    (! &Getopts('v:l:ifh')) && die $usage;
    ($opt_l) && ($in_file=$opt_l) && return;
    ($opt_h) && die "\n$usage";
    &chkValidDom;
}
```

```
sub chkValidDom
{
    ($opt_i) && (!$opt_f) &&
    (($dom =~ /[A-Z]/) || ($dom =~ /[a-z]/)) &&
    die "\ninvalid IP address: $dom\n";
}
```

Listing A: Continued

```

    (!$opt_i) && (!$opt_f) && ( $dom =~ /[0-9]/ ) &&
    die "\ninvalid domain name: $dom\n";
}

sub debug
{
    (@_[0] <= $opt_v) && print STDOUT @_[1];
}

sub cleanup
{
    $cnt=0; $tot=0; $dom="";
    $IP=""; $nonAuth="";
}

sub lookup
{
    local($server) = @_[0];
    local($type) = @_[1]; # 'fwd'=forward; 'inv'=inverse
    $stillNS=0;          # stillNS=1 means referrals returned
    $buf="";             # saves the answer from nslookup

    &debug(4, "parent: setting server to '$server'\n");
    &debug(4, "parent: writing '$dom'\n");

    # set the server
    print Pwld "server $server\n";
    # read back the output from nslookup
    $byte="0"; while ($byte ne ">"){ read(Prfd, $byte, 1);}

    # clear out buf; enter domain and read output
    print Pwld "$dom\n";
    $byte="0";
    while ($byte ne ">"){
        read(Prfd, $byte, 1);
        &debug(5, "parent: read char: $byte\n");
        $buf .= $byte;
    }

    &debug(4, "--- output read back: --- \n$buf\n");
    &debug(4, "--- output read back end: --- \n\n");

    # write the output to a file for file processing.
    open(th, "> /tmp/nslk.out");
    print th $buf;
    close(th);

    # if NS fails to find domain, return error
    open(th, "< /tmp/nslk.out");
    while(<th>){
        if(/$NED/i)!!(/$NRFS/i){
            close(th);
            print STDOUT $_;
            return 1;
        }
    }
    close(th);

    # if NSs returned, read them into an array.
    if ($type =~ /fwd/){
        $i=0; $retNSlist[0]="";
        open(th, "< /tmp/nslk.out");
        while(<th>){
            if (/NSB/){
                $stillNS=1;
            }
            if (/^ /) && ($stillNS == 1) {
                chop;
                @a=split(/ /, $_);
                $retNSlist[$i++]=$a[1];
            }
            close(th);
        }
        else{
            $i=0; $retNSlist[0]="";
            open(th, "< /tmp/nslk.out");
            while(<th>){
                if (/SAAFF/){
                    $stillNS=1;
                }
                if (/NS/ ) && ($stillNS == 1) {
                    chop;
                    @a=split(/ = /, $_);
                    $retNSlist[$i++]=$a[1];
                }
                close(th);
            }

            # Try each next NS if one fails
            if ($stillNS){
                for ($j=0; $j < $i; $j++){
                    {
                        $nslst[++$cnt] = $retNSlist[$j];
                        &debug(1, "trying $retNSlist[$j]\n");
                        if ($type =~ /fwd/){
                            if (! &lookup($retNSlist[$j], "fwd")){
                                last;
                            }
                        }
                        else{
                            if (! &lookup($retNSlist[$j], "inv")){
                                last;
                            }
                        }
                    }
                }
            }
            else{ # Get the IP addr. of the hostname
                open(th, "< /tmp/nslk.out");
                if ($type =~ /fwd/){
                    while(<th>){
                        if (/N1/){
                            $found=1;
                        }
                        if ( (/ $Addr/ ) !! ( / $Addr/ ) ) &&
                            ($found == 1){
                            chop;
                            @a=split(/:/, $_);
                            $IP=$a[1];
                            last;
                        }
                    }
                }
                else{
                    while(<th>){
                        if (/N2/){
                            chop;
                            @a=split(/ = /, $_);
                            $IP=$a[1];
                        }
                    }
                }
            }
        }
    }
}

```


Listing A: Continued

```
last;
    }
}
    }
    close(th);
}
return 0;
}

sub do_process
{
    local($dom) = @_;

    $dom = $tdom;
    &chkValidDom;
    # set first NS in the list to local NS
    $nslist[$cnt]=$defserver; #don't increment $cnt here

    if ($opt_i == 1){ #inverse lookup requested
        &lookup($defserver, "inv");
    } else{
        &lookup($defserver, "fwd");
    }
    # present results
    $tot = $cnt + 1; # $cnt had started at 0
    print STDOUT "\n\nThe NSs(tot=$tot) contacted for \
'$dom' were:\n";
    for ($j=0; $j <= $cnt; $j++) {
        $t=sprintf("%-40s\n", $nslist[$j]);
        print STDOUT $t;
    }
    ($cnt == 0) && ($nonAuth="$NA");
    if ($opt_i){
        print STDOUT "hostname of $dom $nonAuth = $IP\n";
    } else{
        print STDOUT "IP addr of $dom $nonAuth = $IP\n";
    }
    &cleanup;
}

# ----- main -----

($#ARGV < 0) && die "\n$usage";
&getOpts();

# spawn a child to take nslookup commands
pipe(Crfd, Pwfd); #Child reads & Parent writes it
pipe(Prfd, Cwfd); #Parent reads & Child writes it
if ($chpid = fork){ # parent here
    &debug(4, "I am parent\n");
    &debug(4, "child chpid = $chpid\n");
    close(Cwfd);
    close(Crfd);
    select(Pwfd); $! = 1;

    # Get the Default server from nslookup startup

    $byte="0 ";
    while ($byte ne ">"){
        read(Prfd, $byte, 1);
        $buf .= $byte;
    }
    open(hdl, "> /tmp/nslookup.start");
    print hdl $buf;
    close(hdl);
    open(hdl, "/tmp/nslookup.start");
    while(<hdl>) {
        if (/^DS/){
            @a=split(/\s/, $_);
            chop;
            $defserver=$a[3];
            $tstr= "default server=$a[3]\n";
            &debug(3, $tstr);
            last;
        }
    }
    close(hdl);

    if ($opt_i == 1){
        print Pwfd "set type=ptr\n";
        #read back the initial output from nslookup
        $byte="0";while ($byte ne ">") {read(Prfd, $byte, 1); }
    }
    if ($in_file){
        open(listh, $in_file) || die "can't open input file\n";
        while(<listh>){
            /^#/ && next;
            ($tstr="\n..... Looking up ",) .= $_;
            &debug(1, $tstr);
            chop;
            &do_process($_);
        }
        close(listh);
    }
    else{
        &do_process($dom);
    }
    close(Pwfd);
}

else { # child here
    &debug(4, "I am child\n");

    # Child stdin, out, err all now from/to the pipes
    open(STDIN, "<&Crfd") || die "Can't dup child stdin";
    open(STDOUT, ">&Cwfd") ||
        die "Can't dup child stdout";
    open(STDERR, ">&STDOUT") ||
        die "Can't dup child stderr";
    select(STDOUT); $! = 1;
    close(Prfd); close(Pwfd);
    close(Crfd); close(Cwfd);
    exec("nslookup -norecurse -nosearch");
}
```

Coming up...

- Solaris 7 security
- Using hosts.allow and hosts.deny

Listing B: Sample runs with our Pearl nslookup script

```
$ perl dbg_dns.pl www.zdjournal.com
```

The NSs(tot=3) contacted for 'www.zdjournal.com' were:

host.company.com

B.ROOT-SERVERS.NET

NS1.TOOL.NET

IP addr of www.zdjournal.com = 152.175.1.108

\$

```
$ perl dbg_dns.pl www.ang.af.mil
```

The NSs(tot=4) contacted for 'www.ang.af.mil' were:

host.company.com

C.ROOT-SERVERS.NET

MARS.AF.mil

NS.ANG.af.mil

IP addr of www.ang.af.mil = 132.80.207.6

MakeIndex: a tape utility

by Arthur Haigh

I'm fond of my backup tapes. They have saved the day more than just once. The problem I have, however, is not in making the backup tapes, but in keeping track of what's on them and when they were made. I'm sure I'm not the only systems administrator that has a pile of mystery tapes lying around. To help me keep that pile manageable, I wrote the MakeIndex script.

What's an index?

My backup scripts are similar to those presented in "Practical backups for the small Solaris system," in the June 1998 issue, and "No-brainer backup," in the November 1998 issue. The scripts create one ufsdump file per file system and sequentially put them onto tape.

In this article, I'll present a short Bourne shell script, called MakeIndex, that will take a tape that has been created with multiple ufsdump files and create an index of those files. The script doesn't create what the Solaris ufsrestore man page refers to as a table of contents. A *table of contents* is a list of every file and directory that's backed up within a ufsdump file. The MakeIndex script creates a list, including the name of each file system mount point, the date the dump was created, and the level of the dump.

The script is a hands-off program. One simply inserts a tape and executes the script. The index is written to standard output. The tape is automatically rewound and ejected when finished.

Creating the tape index

Execution of the MakeIndex script produces the output shown in Listing A. The script shows that the host europa had its root (/), /usr and /export/home filesystems dumped (with level 0 dump) on Thursday, February 18th at about 7:00 P.M. In addition, europa had

DOWNLOAD
ftp.zdjournal.com/sun



Listing A: Standard output from the execution of the MakeIndex shell script

```
# ./MakeIndex
```

```
File: 0
```

```
Dump date: Thu Feb 18 19:07:03 1999
```

```
Dumped from: the epoch
```

```
Level 0 dump of / on europa:/dev/dsk/c0t3d0s0
```

```
File: 1
```

```
Dump date: Thu Feb 18 19:18:00 1999
```

```
Dumped from: the epoch
```

```
Level 0 dump of /usr on europa:/dev/dsk/c0t3d0s5
```

```
File: 2
```

```
Dump date: Thu Feb 18 19:31:23 1999
```

```
Dumped from: the epoch
```

```
Level 0 dump of /export/home on europa:/dev/dsk/c0t3d0s6
```

```
File: 3
```

```
Dump date: Thu Feb 18 19:51:53 1999
```

```
Dumped from: Wed Feb 17 04:07:04 1999
```

```
Level 2 dump of /var on europa:/dev/dsk/c0t3d0s7
```

```
File: 4
```

```
EOM
```

the /var file system dumped at a level 2 dump on Thursday, February 18th. The line reading

```
Dumped from: Wed Feb 17 04:07:04 1999
```

indicates the date of the last lower level dump of /var. For a plethora of information on dump levels, refer to the man pages on `ufsdump`. You may have noticed that the file systems that had been dumped at level 0 produced the line:

```
Dumped from: the epoch
```

This indicates that there's no previous lower-level dump for this file system.

Picking it apart

The shell is shown in [Listing B](#). Line 1 defines the shell as a Bourne shell. The variable `TAPE` is defined on line 7, and line 8 initializes the variable `ERROR`.

Typically, the backup consists of a series of dump files. The `while` loop will loop over the number of files. For the sake of simplicity, performance, and flexibility, the script doesn't determine the number of files to loop over before looping. Rather, the `while` loop continues as long as the value of the variable `ERROR` is zero. You can see in line 12 of [Listing B](#) that the file number of the first record is determined by pruning the file number from the output of the

`mt status` command. The standard output from the `mt` command is piped to `grep file` and then further reduced by piping to the `awk` command. The result, a number indicating the current file position on the tape is assigned to the variable `FILENUM`. Line 13 echoes the file number to standard output.

The heart of the shell

Line 14 is the heart of the shell and, like line 12, jams many commands into this one line. At the heart of this line is the command:

```
ufsrestore -ivf $TAPE
```

You can see that the `-ivf` parameters are used. The `-iv` invokes the verbose and interactive modes. The `f` parameter indicates that we wish to read from the file represented by the variable `TAPE`. Remember, in UNIX, devices are files. We aren't really interested in using the interactive mode of `ufsrestore`, though. We use the interactive mode because, when it's invoked, the command generates a description of the dump file, which includes the file system that was dumped, the date of the dump, and the dump level. Unfortunately, the command gives us more information than we need, including a prompt for interactive use. Here's what it gives us:

```
# /usr/sbin/ufsrestore -ivf $TAPE
Verify volume and initialize maps
Media block size is 126
Dump   date: Thur Feb 18 19:18:00 1999
Dumped from: the epoch
Level 0 dump of /usr on europa:/dev/
↳ dsk/c0t3d0s5
Label: none
Extract directories from tape
Initialize symbol table.
ufsrestore>
```

The `ufsrestore` command directs the prompt to standard error, which is, by default, represented by file number 2. You can see, in line 14 of [Listing B](#), that the standard error is redirected to suppress the prompt:

```
ufsrestore -ivf $TAPE 2>/dev/null
```

To rid our index of unwanted information produced from the `ufsrestore` command, we can redirect the standard output to the `grep` command to pick out only the information of interest. We're interested only in lines that

Listing B: *The MakeIndex shell script*

```
1  #!/bin/sh
2  #-----
3  #
4  # MakeIndex: Creates an index for a tape in ufsdump format
5  #
6  #-----
7  TAPE=/dev/rmt/0n
8  ERROR=0
9
10 while [ $ERROR -eq 0 ]
11 do
12  FILENUM=`mt -f $TAPE status | grep file | awk '{print $3}'`
13  echo "\nFile: $FILENUM"
14  echo "quit\c"|ufsrestore -ivf $TAPE 2>/dev/null|grep "ump"
15  # echo "quit\c"|ufsrestore -ivf $TAPE 2>/dev/null|grep "ump "
16  ERROR=$?
17 done
18 echo "EOM"
19 mt $TAPE rewofl &
20 exit
```


have the string `dump` in them. These include `Dump`, `Dumped`, and `dump`. We pipe the output to `grep` and search for these strings. We need to restrict our search string to pick out the string `ump`:

```
ufsrestore -ivf $TAPE 2>/dev/  
↳null | grep "ump"
```

Since we invoked the interactive mode, we need to enter the `quit` command to `ufsrestore`'s standard input. We can again use a pipe to redirect the output of the `echo` to the input of `ufsrestore`:

```
echo "quit\\c"|ufsrestore -ivf $TAPE 2>/  
↳dev/null|grep "ump"
```

Finishing up the loop

Immediately following the `ufsrestore` command is the assignment of the variable `ERROR`. Line 16 assigns the return code from the previous command issued using the built-in shell variable `$?` . If the `ufsrestore` command was successful, then a zero is returned and the `while` loop continues. If zero isn't returned, indicating an error, the `while` loop finishes. The error we foresee is when there are no more dump files to restore. When the `ufsrestore` command fails, we're finished with the loop.

In line 18, I have the script `echo EOM` (for End of Media) to indicate that there are no more dump files on this tape. The `mt` command is then issued to run in detached mode (line 19). The tape is rewound and then ejected by the `rewofl` option, which means rewind and take

offline. Since the `mt` command is detached, the shell is free to exit without waiting for the rewind to finish.


Since I almost always perform level 0 dumps, I got tired of seeing the line

Dumped from: the epoch

If you're not interested in the dump level, or if you just don't like seeing the facetious reference to the epoch, you can comment out line 14 in **Listing B** and uncomment line 15. The difference between these lines is simply a space at the end of the string `ump`. The space eliminates the occurrence of the string containing `Dumped` in the `MakeIndex` output.

Conclusion

The `MakeIndex` script is simple, flexible, and versatile. Used as printed in **Listing B**, the script provides a hands-off utility for identifying and verifying the contents of your tapes. The commands can be incorporated at the end of a `ufsdump` backup script to verify the backup. You can be sure that the backup was successful because the `MakeIndex` program is reading the tape. Finally, you can redirect the standard output of the `MakeIndex` command to a file to create written records that can be filed or printed and attached to the tape.

The `ufsdump` command is an excellent resource for creating backups. Coupled with this handy script, you'll be able to simplify and automate the creation, verification, and organization of your valuable tapes. 

Managing licenses

by Vinay Gupta

I know it's bad to talk about license managers when the software paradigm is moving towards freeware. But, in reality, we use many software products that are licensed. `Flexlm` is one license manager that comes by default with the Solaris operating system and is the most widely used license manager among software developers.

Management of licenses becomes more difficult when application vendors assume that

their application is the only application (with licenses) running on your machine. This sometimes leads to a lot of confusion and may stop other license daemons from running on the machine.

License file semantics

A typical license file has three main categories: server identifier, application license daemon identifier, and application feature

(or increments) identifier. Server identifier has the license server host name, hostid, and tcp port. This is defined only once in a license file. The daemon identifier defines the controlling license daemon and its full path. A license file may have multiple numbers of daemon identifiers. The feature identifier has controlling daemon, version, license expiration date, number of licenses, and a license string. Each license daemon controls multiple application features.

License package

Flexlm either comes by default with the application software or one can install it from the SUN Workshop CD-ROM. The actual package name on the SUN CD-ROM is SUNWlicsw. One can also install SUNWlit, a license installation tool.

License installation


Installation and configuration of licenses can be implemented in multiple ways. This basically depends on how many licenses you're running and how many machines are using the license server. In a very simple case where only one machine is the license server and there are few licenses, you can follow this procedure:

1. Append all the licenses to one file. Make sure you have only one SERVER line and the path of all the daemons is correct.
2. Choose one area that's visible to all users from all the machines. You may choose any NFS-mounted partition like /usr/local. Create one directory called /usr/local/licenses.

3. Install the Flexlm tools (SUNWlicsw) in /usr/local/licenses.
4. Copy the licenses file as /usr/local/licenses/license_file.
5. On the license server, start the licenses with


```
/usr/local/licenses/bin/lmgrd.ste -c
/usr/local/licenses/license_file>>
/usr/local/licenses/license_log 2>&1 &
```
6. Edit the /etc/rc2.d/S85lmgrd (installed by SUNWlicsw package) file to start the licenses at every boot time.
7. Define LM_LICENSE_FILE variable to /usr/local/licenses/license_file for all the users.
8. If you get an update for some licenses, just edit/append them to the licenses file and ask the Flexlm to reread the license file with

```
/usr/local/licenses/bin/lmreread -c
/usr/local/licenses/license_file
```

If your site has a large number of licenses, then the maintenance of one single license file itself can become a problem. You can then make multiple license files for each application vendor, but make sure that all of them are talking to different tcp ports. Now you can start all of these licenses one by one, or better yet, edit /etc/rc2.d/S85lmgrd and define all the license files there. Again point the LM_LICENSE_FILE variable to all the license files. With some planning, using the license manager doesn't need to be a difficult experience. 

Backup made easy

by Vinay Gupta

Backup is a real problem if you're managing a small site with no dedicated system administrator. By default, the Solaris server set has a great backup tool called the Data Backup utility. Second, it will allow only one machine backup on the local tape device. If you want to back up two or more machines on one tape device, then you need to buy additional licenses.

A simple bourne shell

Here's a simple bourne shell script that can be used to make a backup of one or more machines. It's easy to understand, easy to change, and easy to customize per site requirements. This script basically uses two utilities provided by Solaris. These are `ufsdump` and `mt`, found in the /usr/sbin and /usr/bin directories, respectively. `ufsdump` backs up either all files or



only changed files (since the last full backup) in a file system to a tape device. `mt` manipulates the tape drives.

This script will take two arguments; the first is dump level and the second is host name. Normally we hate to make deeper than a level 1 backup, but you can always customize the scripts according to your site policy.

Note: In case of a disaster, you need to restore only two backup tapes, level zero and level one—but remember that any backup is always better than no backup.

Here are some assumptions: `host0` is the host name of the machine where you have physically connected the tape; the tape device is `/dev/rmt/0un`; and `host1`, `host2`, and `host3` are client machines. This also assumes that you're taking level zero on an increment level one on every other weekday.

This script will generate some log messages that will go to the `/var/backup/log` directory, and it will read the tape ID from the `/var/backup/tape_id` file. This is the only file that you have to change manually every time you change the tape in the drive. This file has a tape number that matches the tape physically present in the drive.

The number of tape changes will depend upon the capacity of your tape drive and the total size of your backup. For example, if you have four machines (`host0...host4`) each with 6 GB of data area, and you have a tape drive capable of handling 24 GB (6X4) of data, and you want only one full backup in a week, then you need to change the tape and the `tape_id` file once a week.

If your tape capacity isn't enough, then we suggest using level 0 for different machines on different days. After a week, you can think about appending the level 1 of some different machine to the same tape that has been used for level 0 of some other machine, provided your tape capacity allows that. For example, you can make a full backup of `host0` on `tape001` and append a level one backup of `host1` to the same tape. Yes, you guessed it, human intervention is dependent on the ratio of total backup data and tape capacity.

Step-by-step explanation

Here's a step-by-step explanation of the script shown in **Listing A** on page 10. The first line tells the script to use the bourne shell to exe-

cute; then we have some comments about the scripts. Line 7 through 13 will check for the proper command line arguments. If they aren't supplied, then we report the usage and exit.

We have defined some variables in lines 15 to 22. Variable `LEVEL` and `B_HOST` (host machine to be backed up) are derived from the command line arguments. Later on, we'll report an error if either host name isn't from the defined list or level is deeper than one. `T_DEV` is the tape device name—check your site for the correct device name. The `TAPE_ID` variable reads the tape ID from the user-defined file.

Lines 24 to 43 check for the defined host name and define the raw device name to be backed up depending on the machine. If some machines have more than one raw device to be backed up, then you can add that area here as `R_DEV1` and at the end of the script. This is where the actual `ufsdump` command starts the job.

Lines 45 through 63 make sure that you're taking either a level zero or a level one backup; otherwise, it prints a message and exits. It also positions the tape to the end of written media if you're using the same tape for multiple workstations. This is required, because if you've removed the tape from the drive or if you've rebooted the workstation, then the tape will rewind itself.

Here's an option if you want to erase everything before taking the backup. Just insert the following lines at this point:

```
if test `date +%a` = Tue
then
  rsh $T_HOST $MT -f $T_DEV erase
fi
```

This will effectively recycle the tape on every Tuesday. A better way to use this is to have it in a separate script and call from cron.

Lines 64 through 66 will write about the tape ID and the actual file number on tape to log file, and the last line will do the trick while logging everything to log file. Finally edit the crontab file and add the entry for the backup, one example entry on `host1` will look like:

```
0 19 * * 1 /root/backup 0 host1 > /
➡dev/null 2>&1 0 19 * * 2,3,4,5 /
➡root/backup 1 host1 > /dev/null 2>&1
```


This will take the zero level backup of `host1` on every Monday and level one backup on every other weekday.

You can also add more features to this script. For instance, you can create archive files of each dump file as we did in "Make-Index: a tape utility." This will be a table of contents that can be used by the `ufsrestore` command. If you have an auto changer, then you can use the `l` option of `ufsdump` to load the next tape from the magazine. If you have an autoloader, then you can also use the `mtx` utility (freely available on the Internet, and it comes with HP autoloaders). `mtx` (only for autoloaders) can reduce the human intervention needed to perform your backups a lot, as it provides a way to load and unload the tape from the host machine itself.

Restore

You can use `/usr/sbin/ufsrestore` to restore your data from the tape. You need to go to the right file on the tape if you're appending the multiple backups to one tape. The `mt` command can be used for jumping to a particular file on a tape. For example, if you want to go to the third file from the first file, then you should use:

```
mt -f /dev/rmt/0un fsf 3
```

Do this before starting the restore option. You can also use the `i` option of `ufsrestore` for interactive restore, where you can actually select the file or directories you want to restore. 

Listing A: Our level zero backup script

```
1  #!/sbin/sh
2  # Script to take level zero and one backup of UNIX file
   # systems.
3  # syntax: backup [dump level] [host to be backed up]
4  #
5  # ver 1.0    by Vinay Gupta
6
7  # Check for command line arguments.
8  if test $# != 2
9  then
10     echo "Syntax: "
11     echo "backup [dump level] [host to be backed up] "
12     exit
13 fi
14
15 # lets define some variables
16 LEVEL=$1
17 B_HOST=$2
18 T_HOST=host0
19 T_DEV=/dev/rmt/0un
20 DUMP=/usr/sbin/ufsdump
21 MT=/usr/bin/mt
22 TAPE_ID=`cat /var/backup/tape_id`
23
24 # Check if host to be backup up is one we have and
25 # define area to be backed up on that machine.
26 case $B_HOST in
27     host1 )
28         R_DEV=/dev/rdisk/c0t2d0s0
29         break
30         ;;
31     host2 )
32         R_DEV=/dev/rdisk/c0t2d0s6
33         break
34         ;;
35     host3 )
36         R_DEV=/dev/rdisk/c0t2d0s3
37         break
38         ;;
39     * )
40         echo "Please give one of host1, host2 or host3 \n"
41         exit
42         ;;
43 esac
44
45 case $LEVEL in
46     0 )
47         rsh $T_HOST /usr/bin/mt -f $T_DEV eom
48         break
49         ;;
50     1 )
51         rsh $T_HOST $MT -f $T_DEV eom
52         FILE=`rsh $T_HOST $MT -f $T_DEV stat! grep file |
           ➡ cut -c12-15`
53         break
54         ;;
55     * )
56         echo " Please take only level 0/1 dump only "
57         echo " No fun in takeing deeper level dumps "
58         exit
59         ;;
60 esac
61
62 DATE=`date +%m.%d.%Y`
63
64 # Generate some header for the log files.
65 echo "This backup is going to $TAPE_ID
   ➡"/var/backup/log/$B_HOST.$DATE
66 echo "This backup tape file is $FILE
   ➡"/var/backup/log/$B_HOST.$DATE
67 $DUMP "$LEVEL"uf $T_HOST:$T_DEV $R_DEV >>
   ➡/var/backup/log/$B_HOST.$DATE 2>&1
```


The ssh: what it is and why you need it

by Edgar Danielyan

Back in the good old days of the Internet, one could assume that the information sent over the network wouldn't be captured by sniffers and packet analyzers; we could just telnet or rsh to a host anywhere in the known Net and be 99 percent assured that no one was intercepting our passwords or data. Those days are long gone. If you're still using telnet or rlogin to access any host outside your LAN, read on—you've got a problem.

The problem, as it is

Standard Internet tools supplied with all UNIX systems, including the latest release of Solaris, use clear-text passwords for authentication over the network. These utilities include telnet, rlogin, rsh, ftp, and rcp. This means that any data (including user names and passwords) sent and received using these tools are transmitted in clear text over the network and are vulnerable to interception and misuse. For a long time, people didn't pay much attention to this problem, but nowadays with the everyday expansion of the Internet and influx of new, not necessarily good users, this problem deserved attention and got a solution: the Secure Shell (ssh).

The solution

The ssh, developed by Tatu Ylonen of the Computer Science Department of the Helsinki University of Technology, Finland, solves all the aforementioned, and many other, problems. The two most important features of the ssh are its ability to provide an encrypted and authenticated, thus secure, channel of communication between two hosts over insecure networks, such as the Internet; and its unique flexibility and portability, allowing porting to almost any UNIX and many non-UNIX operating systems. In addition to providing a secure version of telnet, rsh, and rlogin all in one tool, the ssh has many additional features not found in the previously mentioned standard tools, such as protection against:

- Interception of clear text user names and passwords
- IP spoofing (when one host pretends to be another host)
- DNS spoofing (when DNS entries are modified to provide false DNS information)
- Man-in-the-middle attack, when data transmitted over the network is modified on the fly
- Various X Windows attacks (spoofing of X11 connections, etc.)

How it works

The ssh uses public key cryptography to encrypt data and authenticate users across the network, almost seamlessly fitting into the standard UNIX environment. Using special protocol, also called the ssh, the client establishes a connection over TCP with the host, which is encrypted using one of the supported encryption algorithms (for example, IDEA, Triple DES, DES, or Blowfish), IDEA being the default. The encryption keys are exchanged using RSA, thus providing a very high degree of security. Keys aren't saved anywhere and are transparently replaced every hour.

Another great feature of the ssh is that it has built-in compression support, using the GZIP algorithm, which is really useful for sites with low bandwidth links. The actual compression ratio, of course, will depend on the nature of the data transmitted.

Some applications of the ssh

Besides being the plug-n-play replacement of telnet, rsh, rlogin, and rcp, the ssh has a number of not-so-widely used but, nevertheless very handy, features. For example, if you're using rsh for backups over the network, you can continue doing so—in most cases, transition from rsh to the ssh will be painless. You can even use the ssh across a firewall by using

a feature called TCP Forwarding, or you can connect two subnets using PPP over the ssh, thus creating a Virtual Private Network (VPN). There are, however, some protocol implications with running PPP over TCP that aren't discussed here, such as a TCP in PPP over TCP re-transmission problem. See www.inka.de/~bigred/sw/ssh-ppp-new.txt for more information.

Obtaining and installing the ssh

There are both free and commercial versions of the ssh, available from the ssh Communications Security (www.ssh.fi) and Data Fellows (www.datafellows.com) or directly from <ftp://ftp.cs.hut.fi/pub/ssh/>.

However, before downloading or using the ssh, keep in mind that there are laws in some countries restricting or banning the use of cryptography. In the United States, it's illegal to import and then export the ssh. In other countries, the laws may be similar.

Compiling the ssh is an easy task. Get the current version of the ssh, unpack it using

```
gzip -c -d ssh.tar.gz | tar xvf -
```

Then change to the newly created directory and

```
./configure
make
make install
```

will do the job. The ssh is written in ANSI C, and requires an ANSI C compiler to compile (GCC is fine). The default configuration options for configure probably will suit most users; however, if you have a non-standard installation or would like to explore all the compilation-time configuration options the ssh has to offer, check out file INSTALL in the ssh sources. Some options you may want to change for your installation are shown in [Table A](#). The ssh comes with a variety of programs that are listed in [Table B](#).

Table A: The available ssh configuration options

Option	Description
<code>--prefix</code>	Where to install the SSH files (default is <code>/usr/local</code>)
<code>--without-des</code>	Don't use single DES (you don't want to use it anyway, off by default)
<code>--with-securid</code>	Include support for Security Dynamics' SecurID card
<code>--with-tis</code>	Include support for TIS
<code>--with-kerberos5</code>	Include and enable support for Kerberos 5
<code>--disable-server-x11-forwarding</code>	Disable X11 connections forwarding in the ssh server
<code>--disable-suid-ssh</code>	Install the ssh server without the SUID bit

Table B: The ssh package contents

Program	Description
<code>sshd</code>	The ssh server
<code>ssh</code>	The ssh client
<code>scp</code>	Secure rcp
<code>ssh-keygen</code>	Utility to create RSA keys (both host keys and user keys)
<code>ssh-agent</code>	The ssh authentication agent
<code>ssh-add</code>	Utility to register new keys with the ssh-agent
<code>make-ssh-known-hosts</code>	Utility to create <code>/etc/ssh_known_hosts</code>

A compilation note to Solaris x86 users: Solaris x86 uses syntax for assembler that's not supported by gmp provided by the ssh. The configuration will automatically detect this and disable assembler code. After you've compiled and installed the ssh, you need to configure sshd, the sshserver, to fit your requirements.

Supported encryption algorithms

The ssh supports a number of encryption algorithms, namely Blowfish, IDEA, DES, Triple DES, and Arcfour. Here are their general characteristics:

- Invented by Bruce Schneier, *Blowfish* is a fast block cipher with 32- to 448-bit keys (the ssh uses 128-bit keys with Blowfish).
- *Arcfour* is a fast stream cipher with variable key lengths (the ssh uses it with 128-bit keys), compatible with the RC4 cipher developed by RSA Data Security, Inc. There are some problems with this cipher, so we recommend using IDEA or Triple DES.
- *IDEA* is a 128-bit block cipher, which is patented in many countries. It's faster than Triple DES but slower than Blowfish or Arcfour. This is the default cipher used by the ssh.
- *DES* is an old, 56-bit block cipher. It's three times faster than Triple DES but much slower than Blowfish or Arcfour. The key length is fixed and isn't serious. Don't use this cipher.
- *Triple DES*, a modification of DES, uses three keys in CBC mode. This cipher is required by the ssh, so you can't disable it.

For more information on these encryption algorithms and ciphers, consult Bruce Schneier's *Applied Cryptography* (John Wiley & Sons).

There are also two Usenet newsgroups on cryptography: sci.crypt and talk.politics.crypto. An FAQ for sci.crypt is at www.cis.ohio-state.edu/hypertext/faq/usenet/cryptography-faq/top.html. For really interested folks, there's an IEEE Symposium on Research in Security and Privacy at www.ieee.org, and the International Association for Cryptologic Research at www.swcp.com/~iacr. ACM also has a Special Interest Group on cryptography and security

at www.acm.org, as does the IEEE Computer Society at www.computer.org.


Configuring the ssh

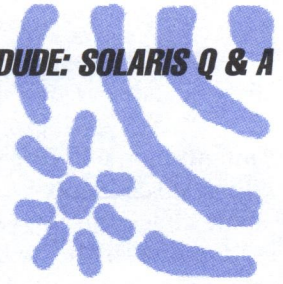
The default configuration, installed with make install in /etc/sshd_config, is a generic one and will make the ssh server run. However, should you wish to check or customize the configuration, take a look at man sshd, which has all the options and detailed explanations. Here are some options you might want to specify:

- IdleTimeout sets idle timeout limit for the ssh connections
- IgnoreRhosts specifies whether the ssh honors.rhosts
- IgnoreRootRhosts specifies whether the ssh has to use .rhosts for root logins
- PermitEmptyPasswords specifies whether users with empty passwords can log on
- StrictModes specifies whether the ssh has to check the file modes of the user's home directory and .rhosts file

We recommend that you disable telnet and rlogin/rsh/rcp in /etc/inetd.conf altogether and switch to the ssh. In this case, don't forget to start sshd on start-up in /etc/rc2.d. An optional argument to the sshd is "-b bits", specifying the RSA key size (default is 768, which is okay for most installations, with 512 bits being weak and 1024 bits very strong). Of course, a larger key size increases security but decreases performance, so it's probably best to leave it at 768. It's possible to start sshd from the inetd, but there's no real advantage in doing so—for each incoming ssh connection, sshd would need to generate an RSA key, which may take some time on slower machines. And don't forget to open port TCP/22 for the ssh on your firewall.

Conclusion

Any system is only as strong as its weakest link. Use of the ssh shouldn't preclude you from frequently changing passwords, using good, hard-to-guess passwords, taking care of physical security of your systems and networks, and, of course, keeping informed of new developments. 



Why does traceroute work on my Windows box, but fail on my Sun box?

by Lance Spitzner


Tracert is a popular utility used to troubleshoot networks. Implemented by Van Jacobson, traceroute identifies the hops that a packet takes from source to destination. This path lets us map the routers and systems that our packet goes through. Originally written for UNIX, Microsoft added its own version of traceroute (called tracert) to its Windows and NT operating systems. What many people don't realize is that while these tools look similar, they operate differently.

These differences are critical in today's world of firewalls and ACLs (Access Control Lists) on routers. Often you'll find that one version will work, while the other fails. This is a result of how they work.

UNIX versions of traceroute (www.metalab.unc.edu/pub/solaris) use UDP datagrams outbound, and ICMP inbound. Traceroute begins by sending three UDP datagrams (port > 30,000) to the destination with a TTL (Time to Live) of 1. The first hop decrements the TTL by 1. Since the TTL is 0, the first hop sends us an ICMP time exceeded in transit error (type 11, code 0). Traceroute receives this error, records the IP, and then sends three more UDP datagrams, with a TTL of 2. These packets will fail at the second hop. This mapping process of sending datagrams while incrementing the TTL continues until the destination host is reached. Hopefully, the remote system isn't listening on the UDP port we've attempted to reach. If there's no daemon listening, the remote system sends back an ICMP port unreachable (type 3, code 3). Traceroute then knows we've reached our target.

Windows tracert uses the same concept, but ICMP requests (type 8, code 0) instead of UDP datagrams. Tracert begins by sending 3 ICMP requests with a TTL of 1 to the destination. The first hop decrements the TTL by 1. Since the TTL is zero, it sends back an ICMP time exceeded in transit error (type 11, code 0). Tracert records this and sends three more ICMP packets to the destination, with a TTL of 2. These packets will fail at the second hop. This process continues until the destination is reached, which returns an ICMP reply (type 0, code 0).

As you see, traceroute and tracert implement the same concepts, but use a different method. It's this difference in methods that can cause your problems; one type of packet is allowed, while the other is denied. For example, let's say your routers allow all ICMP traffic, but deny all UDP traffic except for port 53, DNS. Tracert on Windows would work fine, as it uses only ICMP. However, traceroute on our Sun box would fail, as it uses UDP, which is denied by our routers.

Another problem could be firewalls. Many firewalls can be configured to allow stateful connections to go through. If an ICMP packet passes through your firewall, your firewall may expect an ICMP return packet. However, if a UDP datagram passes through your firewall, your firewall may expect a UDP datagram to return, when in fact it gets an ICMP packet, which it drops. By understanding the differences between traceroute on your Sun box and tracert on Windows, you can better understand why you're having problems. 

If you have any Solaris questions you would like answered, shoot your questions off to lance@spitzner.net, and Ping the Solaris Dude!

Easily convert man pages to text documents

by Al Alexander


Have you ever wanted to convert a man page into a plain text document? I do this occasionally when I want to share information via an email or other document format.

I used to think this was difficult, but then I discovered a simple way to do it. Here's the wrong way to write the man page for the `ls` command into a text file named `ls.bad`:

```
man ls > ls.bad
```

This keeps all of the formatting characters in your document, which is generally not what you want. Here's a better way that eliminates those formatting characters:

```
man ls | col -b > man.txt
```

The `col` command with the `-b` option removes the undesirable backspace characters from the text stream, so the only thing left in your document is the text you want, in the format you want. 

About our contributors

Alvin J. Alexander first began his career as an aerospace engineer. He's now the president of Mission Data Corporation, an employee-owned computer consulting firm in Louisville, Kentucky. You can reach him online at aja@missiondata.com.

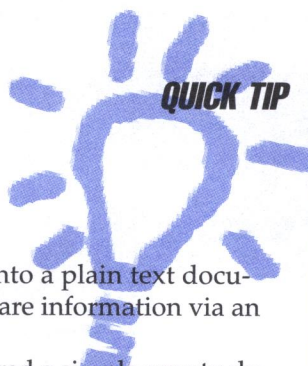
Edgar Danielyan is currently working as a network administrator and manager of a top level domain of Armenia. Previously, he spent some time studying US and UK law. He's also worked for the United Nations, the ministry of defense, a national telco, a bank, and has been a partner in a law firm. He speaks four languages, likes good tea, and is a member of ACM, IEEE CS, SENIX, CIPS, ISOC, IPG, and many other much less known organizations. He can be reached at edd@computer.org.

Vinay Gupta works as a UNIX/NT system administrator with a consulting company. He has been in this field for almost eight years. A resume can be found at www.members.tripod.com/vinay_k_gupta/resume.html.

Arthur Haigh is the Senior Systems Administrator in the Division of Nuclear Medicine, School of Medicine, University of Pennsylvania. He can be reached at art@rad.upenn.edu.

Atiq Hashmi is a software engineer at Telcordia Technologies (formerly Bellcore). His interests are in system and network administration, applications and tools, as well as Internet related technologies. He can be reached at hash100@mail.eclipse.net.

Lance Spitzner enjoys learning by blowing up his UNIX systems at home. Before this, he was an Officer in the Rapid Deployment Force, where he blew up things of a different nature. You can reach him at lsplitzner@enteract.com or www.enteract.com/~lsplitz.



QUICK TIP

Inside
Solaris
Tips & Techniques for users of Sun Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by
ZD Journals 500 Canal View Boulevard, Rochester, NY 14624.

Customer Relations

US toll free (800) 223-8720
Outside of the US (716) 240-7301
Customer Relations fax (716) 214-2386

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

ZD Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at zdjcr@zd.com.

Editorial

Editor Garrett Suhm
Assistant Editor Jill Suhm
Copy Editors Rachel Krayer
Christy Flanders
Taryn Chase
Contributing Editors Al Alexander
Edgar Danielyan
Vinay Gupta
Arthur Haigh
Atiq Hashmi
Lance Spitzner
Rachel J. King
Print Designer

VP & Publisher, Content Development Linda Edwards
Director, Advanced Technology Kent Michels
Managing Editor Joe Froehlich
Product Marketing Manager Jan Mater-Cavagnaro
Manager of Design & Production Services Charles V. Buechel
Executive VP of Operations Jerry Weissberg
Director of Customer Support Douglas Neff
Director of Operations & Fulfillment Kress Riley

You may address tips, special requests, and other correspondence to

The Editor, *Inside Solaris*
500 Canal View Boulevard
Rochester, NY 14623

Editorial Department fax (716) 214-2387

Or contact us via Internet email at sun@zdjournals.com.

Sorry, but due to the volume of mail we receive, we can't always promise a reply, although we do read every letter.

Postmaster

Periodicals postage paid in Rochester, NY and Additional Mailing Offices.

Postmaster: Send address changes to

Inside Solaris
P.O. Box 92880
Rochester, NY 14692

Copyright

Copyright © 1999, ZD Inc. ZD Journals and the ZD Journals logo are trademarks of ZD Inc. *Inside Solaris* is an independently produced publication of ZD Journals. All rights reserved. Reproduction in whole or in part in any form or medium without express written permission of ZD Inc. is prohibited. ZD Journals reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for personal and commercial use.

Inside Solaris is a trademark of ZD 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. Other brand and product names are trademarks or registered trademarks of their respective companies.

Printed in the USA.

Price

Domestic \$99/yr (\$9.00 each)
Outside US \$119/yr (\$11.00 each)

Our Canadian GST# is: R140496720. CPM# is: 1446703.

Back Issues

To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost \$9.00 each, \$11.00 outside the US. You can pay with MasterCard, VISA, Discover, or American Express. Collections of back issues are available on CD-ROM as well. Please call for more information.

Are you moving?

If you've moved recently or you're planning to move, you can guarantee uninterrupted service on your subscription by calling us at (800) 223-8720 and giving us your new address. Or you can fax us your label with the appropriate changes at (716) 214-2386. Our Customer Relations department is also available via email at zdjcr@zd.com.

PERIODICALS MAIL

Sun Technical Support
(800) 786-7638

C:7661905 00002096 04/00

CLINTON TOWNSHIP, MI 48035-4218



Please include account number from label with any correspondence.

Inside Solaris reader survey

With the explosive growth of the Internet and Solaris, we felt this would be a good time to get input from you, our readers. This will help us determine the topics that will best meet your needs. Your responses are extremely important—please take a few minutes to complete and return this form.

You can choose from several ways to complete this survey. If you prefer to respond electronically, go to our FTP site at <ftp.zdjournals.com/sun> and download the file `survey.txt`. Then, fill in your answers, copy the document into the body of an email message, and send it to inside_solaris@zdjournals.com.

To respond by fax or mail, first photocopy the survey form. Complete your answers, and then fax the form to (716) 214-2387 or mail it to us at:

Inside Solaris
ZD Journals
500 Canal View Boulevard
Rochester, NY 14623

From the responses we receive, we'll select three at random. The readers whose forms are drawn will receive a free one-year renewal to *Inside Solaris*. To be eligible for the drawing, we must receive your form no later than October 30, 1999. Thanks for your help!

1. What version of Solaris do you currently use?

- ☐ Solaris 7 ☐ Solaris 2.51
☐ Solaris 2.6 ☐ Other _____

2. If you are not using Solaris 7, when do you plan to upgrade?

- ☐ Immediately ☐ Within a year
☐ Within 6 months ☐ Not sure

3. How would you rate your level of experience with Solaris?

- ☐ Novice ☐ Experienced
☐ Intermediate ☐ Expert

4. How would you describe the technical level of the articles in *Inside Solaris*?

- ☐ Too simple ☐ Too Technical
☐ About Right

5. What kind(s) of articles would you like to see more of (1 being little interest)?

- Code-intensive programming techniques 1 2 3 4 5
Quick "how-to" tips 1 2 3 4 5
Performance and tuning articles 1 2 3 4 5
Case studies 1 2 3 4 5
Java articles 1 2 3 4 5

- | | |
|----------------|-----------|
| Network topics | 1 2 3 4 5 |
| Security | 1 2 3 4 5 |
| Perl | 1 2 3 4 5 |
| Tcl | 1 2 3 4 5 |
| Python | 1 2 3 4 5 |
| Web Servers | 1 2 3 4 5 |
| Firewalls | 1 2 3 4 5 |
| Upgrade Issues | 1 2 3 4 5 |
| Other _____ | 1 2 3 4 5 |

6. Do you use email and/or the Web?

- ☐ Every day ☐ Occasionally
☐ Never

7. Do you make use of our Web/FTP sites?

- ☐ Download sample files
☐ Read back-issue articles
☐ Receive ZDtips

8. Please share any suggestions or comments you have about *Inside Solaris*.
