ZD JOURNALS

February 1999 • $9.00
Vol. 5 No. 2
www.zdjournals.com/sun

Inside Solaris™

Tips & Techniques for users of SunSoft Solaris

PPP setup tips

by Alan Orndorff

In the early 1970s the Internet was born, and over the years it's evolved. Archie has given way to search engines and gopher is being replaced by Web browsers, but some things haven't seen much change. Usenet and email are still essentially the same. Yet all of this is meaningless if you can't access the Internet. Microsoft Windows 95 and NT have made accessing the Web a breeze. Whether you used the vendor-supplied Dial-Up Networking, RAS, or any of the third-party utilities, anyone can be on the Web in minutes. But hey, what about UNIX, the platform that invented and keeps evolving these things?

Unfortunately, it's much more difficult to get online with Solaris than it is with any of the Windows operating systems. But it can be done! The following is a recipe for getting online.

Check your package

Figure A shows the basic parts needed before we can begin. Once we have a modem and a connection to a remote network, we then need to make sure we have the right packages installed. The packages necessary for PPP are:

```
SUNWapppr
SUNWapppu
SUNWpppk
SUNWbnur
SUNWbnuu
```

To see if the packages are installed, type:

```
pkginfo | more
```

This will provide you with a list of your currently installed packages.

Configuration files

After verifying that we have the correct packages installed, we'll need to edit certain configuration files. These files will tell your machine how to dial out and connect to the Internet. The files that you will need to change, and their permissions, are listed below:

```
/etc/hosts lrwxrwxrwx 1 root root

/etc/asppp.cf -rwxr--r-- 1 root sys
/etc/resolv.conf -r-xr-xr-x 1 root other
/etc/nsswitch.conf -rw-r--r-- 1 root root
/etc/uucp/Dialers -r-xr-xr-x 1 root other
/etc/uucp/Devices -r-xr-xr-x 1 root other
/etc/uucp/Systems -r-xr-xr-x 1 root other
```

A complete listing of all of the files that we modified are shown in Listing A on page 3. Remember to be careful when modifying configuration files, since changes can have unintended consequences.

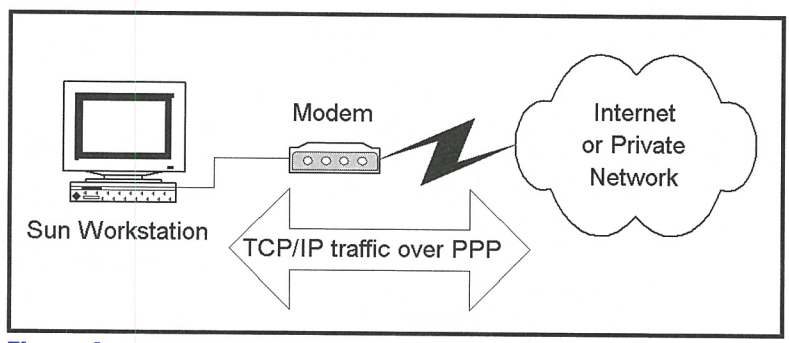


Figure A: *This is the basic hardware setup for using PPP.*

In this issue:

The first file you need to modify is /etc/hosts. Add the following entries:

```
10.1.1.1 dwarf1 <-- my_machine_name

10.1.1.2 netcom <-- remote_machine_name
```

Replace `dwarf1` and `netcom` with the name you wish to use for both sides of the PPP link. These are arbitrary and up to you. Also, the IP addresses are arbitrary, as well. The IP address information will be replaced with Dynamic IP addresses once you're connected to the Internet.

Second, you'll need to edit `/etc/asppp.cf`. This is pretty straightforward and is documented in the file itself. Our asppp.cf file looks like this:

```
ifconfig ipdptp0 plumb dwarf1 netcom up netmask
   255.255.0.0

path
 inactivity_timeout 1800
 interface ipdptp0
 peer_system_name netcom
 default_route
 negotiate_address on
 debug_level 8
```

On the `ifconfig` line, notice that `dwarf1` and `netcom` are referenced in `/etc/hosts`. The word *up* tells `asppp.cf` to dial when the machine is first booted up. If you don't want this to happen, replace it with *down*. `Inactivity timeout` sets the amount of time, in seconds, before an idle connection will hang up the phone line. `Peer_system_name` is referenced in `/etc/uucp/Systems`. `Default_route` tells Solaris to use this as the default router to the Internet. `Negotiate_address` tells Solaris to request an IP address from your ISP. `Debug_level 8` tells PPP to create a verbose log file.

Next, you'll need to modify the files located in `/etc/uucp`. You may or may not need to edit `/etc/uucp/Dialers`. This file contains the AT commands that are sent to your modem. If you have an Internal Modem, you may need to modify your `/etc/uucp/Dialers` file in the following manner: add `P_ZERO` to your modem definition string; for example,

```
hayes   =,-, ""   P_ZERO  ""
  \\dA\\pTE1V1X4QQS2=255S12=255\\r\\c
```

Next, you'll need to modify the `/etc/uucp/Devices` file. At the end of the file, add a line similar to the following:

```
ACUNETCOM cua/b - Any hayes
```

Note that `ACUNETCOM` is referenced in `/etc/uucp/Systems`. Also, `cua/a` is the com1 serial port and `cua/b` is com2. Hayes is the dialer to use in `etc/uucp/Dialers`. This must match an entry in `/etc/uucp/Dialers`; otherwise the `asppp` deamon won't know the correct commands to send to your modem.

The next file you'll need to edit is `/etc/uucp/Systems`. Ours has the following entries:

```
netcom Any ACUNETCOM 38400
  *705551212 "" \\ login:
     username word: password
```

Recall that `netcom` is referenced in `/etc/asppp.cf` as peer system name. The `Any` flag means that the number can be dialed at any time. The `ACUNETCOM` is referenced in `/etc/uucp/Devices`. `38400` gives us the port speed and `*705551212` is the phone number to dial. `Login: username` tells asppp.cf to wait for the login prompt and send the username, then wait for the password prompt and send the password. This is where having verbose logging really helps. You can look at `/var/adm/log/asppp.log` to see the entire connect sequences and use this data help to debug all sections.

The next step is to set up the `/etc/resolv.conf` file. Your ISP provider will provide this information for you. Our resolve.conf file has the following entries for our ISP:

```
domain ix.netcom.com
nameserver 199.182.120.203
nameserver 199.182.120.202
```

And lastly, you'll need to edit the `/etc/nsswitch.conf` file. Simply find the line that has *hosts* and add *dns* to the end of the line. Now you can reboot the machine to test the connection.

Starting and ending your PPP connection

After you reboot your machine, try to start your PPP connection with the following command:

```
/etc/init.d/asppp start
```

Listing A: *Listings of the files we modified to configure PPP*

```
/etc/hosts
#
# Internet host table
#
127.0.0.1 localhost
10.1.1.1 dwarf loghost
10.1.1.2 dwarf1
10.1.1.3 netcom

/etc/asppp.cf
ifconfig ipdptp0 plumb dwarf1 netcom up netmask 255.255.0.0
path
 inactivity_timeout 1800
 interface ipdptp0
 peer_system_name netcom
 default_route
 negotiate_address on
 debug_level 8

/etc/uucp/Dialers  - Pertinent section for my setup
#   Hayes Smartmodem -- modem should be
#   set with the configuration
#   switches as follows:
#
#       S1 - UP  S2 - UP  S3 - DOWN S4 - UP
#       S5 - UP  S6 - DOWN S7 - ?  S8 - DOWN
#
hayes =,-, ""  P_ZERO "" \\dA\\pTE1V1X4Q0
➥S2=255S12=255\\r\\c OK\\r \\EATDT\\T\\r\\c
   CONNECT

/etc/uucp/Devices
ACU cua/b - Any hayes
Direct cua/b - Any direct
ACUNETCOM cua/b - Any hayes

/etc/uucp/Systems
netcom Any ACUNETCOM 38400 *705551212 ""
➥\\ login: username word: password

/etc/resolv.conf
domain ix.netcom.com
```

If all your configuration files were set correctly, you should now have made a PPP dial-up connection. When you want to end your PPP connections, simply type:

```
/etc/init.d/asppp stop
```

If you hang up the line and you wish to reconnect to your ISP without rebooting, you'll need to flush your routing table. You'll also need to type the following command before restarting PPP:

```
route -f defaultroute
```

More information

This article gives you a basic overview of connecting with PPP. If you need to use pap or chap authentication, then please go to **http://docs.sun.com:80/ab2/coll.47.4/NETCOM/@Ab2PageView/idmatch(TAILORINGPPP11-26606)?#TAILORINGPPP11-26606**.

If you find Sun's PPP too confusing and incomplete, please stay tuned. Next month, we'll explore a freeware PPP alternative to Sun's own bundled PPP package.

About our contributors

Alvin J. Alexander is the president and chief scientist of Mission Data Corporation (**www.missiondata.com**), an employee-owned software engineering and systems integration firm. You can reach him online at **alvina@missiondata.com**.

Alan Orndorff has been working with computers since 1990. He's using Solaris as a platform for Lotus Notes and at home in his spare time. He currently lives in San Francisco and can be reached at **dwarf333@hotmail.com**.

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 **lspitzner@enteract.com** or **www.enteract.com/~lspitz**.

Robert Owen Thomas is an aspiring blues guitarist earning his living as a UNIX and networking consultant. He can be contacted through email at **robt@cymru.com**, or visit his Web site at **www.cymru.com/~robt**.

Asim Zuberi received his masters degree in mechanical engineering in May 1993, at the New Jersey Institute of Technology, NJ and works now at Collective Technologies (a Pencom Company). Currently, he's onsite at Lucent Technologies, one of CT's clients. He started with UNIX in 1992, worked with Sun, SGI, and Linux, and has been heavily involved in Solaris administration since 1996. You may reach him at **asim@colltech.com**.

Routing with Solaris

by Lance Spitzner

Last month, we discussed how to configure, modify, and troubleshoot network interface cards. Here, we'll discuss routing issues for systems with two or more network interface cards. We won't be discussing gated, nor any routing protocols, such as RIP or OSPF. This article will focus only on implementing static routing tables. Throughout the article, we'll be using the octet method for denoting subnet masks, as opposed to the more modern method of using a /. For example, we'll be designating a class C network as 255.255.255.0, as opposed to /24. We decided to use the older notation, as this is what Solaris uses. If you have any questions about IP addressing or subnetting, I highly recommend you first review **www.3com.com/nsc/501302.html**.

Routing

Routing is the process of forwarding a packet from point A to point B. Solaris does this by building a routing table. When it forwards a packet, it first refers to the routing table to decide where to send the packet. The key to successful routing with Solaris is building a proper routing table.

You start building your routing table with your first network interface device. When you configure a network interface device, the kernel automatically builds a static routing table. For example, let's say you're on a system that has a single interface, elx0. You configure the device to have an IP address of 207.229.165.133, with a netmask of 255.255.255.0. To see the routing table the kernel has built, use the command `netstat nr` to show results similar to ours in Table A.

Here we see the system's routing table. The first column is *Destination*, which is the network that the packet wants to go to. The second column is *Gateway*, the IP address to which the packet must go to get to the destination—the next hop. The third column is *Flags*, which denotes interface information, such as *U* for up, and *G* for Gateway. The fourth column is *Ref*, which denotes how many times that specific MAC address is referenced in the routing table. The fifth column is *Use*, or how many packets have gone through the interface. The last column is *Interface*, which shows the device the packet must go through if the destination is the local network.

In the example, we have two routes. The bottom one, 127.0.0.1, is the standard loopback route. All systems have this route; the kernel uses it to talk to itself. The second entry is a result of the elx0 interface. This entry says if you need to get to a node on the 207.229.165.0 network, go to 207.229.165.133, which is the system interface. This entry is called the *local network entry* and is added by default. The kernel assumed that because elx0 has an IP address of 207.229.165.133 and a netmask of 255.255.255.0, it must be connected to the local network 207.229.165.0. Thus, if you want to talk to any node on the 207.229.165.0 network, the kernel knows exactly where to send the packet.

Any time you add a new interface, the kernel adds a routing entry similar to the one above. It assumes that the new interface can talk to the local network.

Your system can now talk to the local network, 207.229.165.0. But what about other networks, such as the Internet? If you were to attempt to talk to any node on any other network, such as 206.54.252.8, you'd get the following error:

```
lisa #ping 206.54.252.8
ICMP Net Unreachable from gateway lisa
   (207.229.165.133)
 for icmp from lisa (207.229.165.133)
```

Table A: *We used netstat nr to check our routing tables.*

Destination	Gateway	Flags	Ref	Use	Interface
207.229.165.0	207.229.165.133	U	1	20	elx0
127.0.0.1	127.0.0.1	UH	0	94	lo0

The system has no idea how to reach this node. To fix this, we need to give the system a default route. When the kernel is given a destination it doesn't know, it sends the packet to the default route. The default route is usually the IP address of another router. This router takes the packet and does one of two things with it. If the destination is local to the router, it sends the packet to the local destination. If not, the router sends the packet upstream to another router. This process repeats itself until the packet has reached its destination.

By default, Solaris uses a routing protocol to dynamically determine the default route, RIP or Route Discovery. During the init process, the `/etc/rc2.d/S69inet` will attempt to find a router running route discovery (`/usr/sbin/in.rdisc`). If this fails after three attempts, the script then launches `/usr/bin/in.routed`, otherwise known as RIP. However, we'll use neither method. Instead, we're going to manually set the default route. When the default route is manually set, neither routing protocol is initiated. The advantages to this are a simpler and more secure system to administer.

You manually define the default route with the file `/etc/defaultrouter`. This file consists of a single entry, the IP address of the default router. This file is read during the init process (specifically `/etc/rc2.d/S69inet`) and added to the routing table.

For this system, we've identified the default router as 207.229.165.1. This is the IP address of the router that connects us to the Internet. If our system doesn't know a packet's destination, it sends the packet to the default router. With our default route, the new routing table would look as shown in **Table B**.

Based on this table, the system now has two choices for forwarding a packet. If the destination is on the 207.229.165.0 network, the packet is sent to the local network. However, if the destination is any other network, then the packet is sent to the default router. Notice that the default router, 207.229.165.1, is on the local network. If the default route isn't on the local network, the system can't reach the default router.

IP forwarding

Up to this point, we've been discussing single-homed systems. Single-homed systems have one of two choices: talk to the local network or to the default router. Things get more complicated when you add a second interface. Your system now becomes multi-homed, and potentially a gateway. A *multi-homed host* is any system with two or more interfaces, usually on different networks. A *gateway* is any multi-homed system that routes packets between different networks.

Let's take a look at what happens when we add a second interface. We'll add the interface elx1 to our system, with an IP address of 10.1.6.1, netmask 255.255.255.0. The result is shown in **Table C**.

Looking at the routing table, you notice only one change, the addition of interface elx1. If a packet is destined for any node on the 10.1.6.0 network, the packet is forwarded out the elx1 interface.

However, there's another, far more important, change not seen here. IP forwarding has just been enabled on this machine. Basically, IP forwarding means the system will route packets between networks. Based on the table above, the gateway will forward a packet one of two ways. If it doesn't know the destination, it will forward the packet to the default router. If the destination is on one of the two local networks, then the packet will be forwarded to its destination.

IP forwarding is enabled during the init process, in `/etc/rc2.d/S69inet`. If the system detects more than two interfaces (including the loopback), IP forwarding will be enabled by default. Your system is now a gateway.

You can have a system with two or more interfaces and not forward packets if you want.

Table B: *Our routing table with our new default route*

Destination	Gateway	Flags	Ref	Use	Interface
207.229.165.0	207.229.165.133	U	2	20	elx0
default	207.229.165.1	UG	0	20	
127.0.0.1	127.0.0.1	UH	0	30	lo0

Table C: *Adding a second interface to our system*

Destination	Gateway	Flags	Ref	Use	Interface
207.229.165.0	207.229.165.133	U	2	20	elx0
10.1.6.0	10.1.6.1	U	1	123	elx1
default	207.229.165.1	UG	0	20	
127.0.0.1	127.0.0.1	UH	0	30	lo0

This is done in one of two ways. The first is by touching the file `/etc/notrouter`. During the init process, `/etc/rc2.d/S69inet` will look for this file. If it finds it, it turns off IP forwarding by executing the following command:

```
ndd -set /dev/ip ip_forwarding 0
```

You can manually turn off IP forwarding any time by executing the same command.

Route command

The `route` command allows you to manually change the route table. You can add, delete, or change routes in real time. For example, let's say the IP address of the default router has changed, but you can't afford to reboot the system. You have to change the IP address of the default route without rebooting. You do this with the `route` command. The syntax is simple:

```
lisa #route change default 207.229.165.5 1
```

This command changes the default router from 207.229.165.1 to a .5. The syntax is simple: type the network information as you want it to appear in the routing table. The last number at the end of the command is the metric, or how many hops to the next gateway. Any node, including a router, on the local network is a hop of 1. The `route add` command allows you to add additional routes to the routing table, just as `route delete` removes them. If you want to make a route command permanent, add the command to the bottom of /etc/rc.2/S69inet. The init script will execute the `route` command and update the routing table.

VLSM

Starting with version 2.6, Solaris supports VLSM (Variable Length Subnet Mask). VLSM means a network can be variably subnetted into smaller networks, with each smaller network having a different subnet mask. What that means to you, is that life just got a lot easier.

Under Solaris 2.5.1 or earlier, you could only define a single subnet for a network. For example, if you defined the network 10.1.6.0 with a 255.255.255.0 subnet mask as we've done, older versions of Solaris would assume that any network starting with 10 was a 255.255.255.0 subnet mask. You have now locked yourself in. You had to manually add an individual route for any 10.0.0.0 network that didn't have this subnet. This could easily reach into the hundreds!

VLSM doesn't make this assumption—it gives you flexibility in setting up your routing tables. You can have as many different subnets as you want for a network. Let's take a look at an example. Our current routing table (see the previous example) is configured for two local networks and a single default route for everything else (the Internet). However, this system is to be the gateway for a large corporation, the company's firewall. That means all inbound and outbound traffic must go through it.

The corporation is made up of an internal 10.0.0.0 network, which is subnetted into over 100 smaller networks. Each smaller, subnetted network has a different subnetmask. For example:

10.1.8.1.1	255.255.254.0	(510 hosts)
10.128.112.0	255.255.248.0	(2046 hosts)
10.220.160.0	255.255.240.0	(4094 hosts)

Here you see the company's various networks. We have to create a routing table that routes all default traffic to the Internet, but at the same time routes anything on the 10.0.0.0 internally. Remember, our internal network is really over a hundred smaller 10.0.0.0 networks, all variably subnetted.

Table D: *Our routing table using VLSM*

Destination	Gateway	Flags	Ref	Use	Interface
207.229.165.0	207.229.165.133	U	2	20	elx0
10.1.6.0	10.1.6.1	U	2	123	elx1
10.0.0.0	10.1.6.5	U	0	0	
default	207.229.165.1	UG	0	20	
127.0.0.1	127.0.0.1	UH	0	30	lo0

First, we have to identify the internal router. In our case, we'll use 10.1.6.5. This is the IP address of the router on the internal network. Notice how this router is on the local network of interface elx1. Now, since we're using Solaris 2.6, which supports VLSM, we need only one command to route all the variably subnetted 10.0.0.0 networks:

```
lisa #route add net 10.0.0.0  10.1.6.5  1
```

With this single command, we've taken care of all routing issues, something possible only with VLSM. Let's take a look at the routing table in **Table D** and explain what we mean.

The first line states that if your destination is on the 207.229.165.0 network, the network is local to the interface elx0. If your destination is on the 10.1.6.0 network, the network is local to the interface elx1. If your destination is on any 10.0.0.0 network *except* to 10.1.6.0, the packet is forwarded to the gateway 10.1.6.5. Finally, if the destination meets none of the above criteria (the Internet), the packet is forwarded to the default router, 207.229.165.1.

You may be confused as to how the system knows where to forward anything on 10.1.6.0. By looking at the routing table, you see two entries that would work, one for 10.1.6.0 and one for 10.0.0.0. Both work for 10.1.6.0. The system always selects the most specific path first.

Now, if this were on a system that didn't support VLSM, such as 2.5.1, things would be *much* uglier. As stated earlier, the 10.0.0.0 is variably subnetted into smaller networks. Without VLSM, you'd have to manually add a static route for each separate network with the `route add` command. If you don't, the kernel will assume that all the 10.0.0.0 networks are subnetted the same, causing all sorts of interesting routes. As you can see, VLSM is extremely powerful.

CIDR

I decided to discuss CIDR, as well, as it's easy to confuse with VLSM and they're both closely related. Defined in 1993 by rfc 1519, Classless Inter-Domain Routing is used for routing aggregation, also known as *supernetting*. Simply stated, this means lumping several networks into one. The purpose is to reduce routing tables, which are beginning to overload backbone routers. An example would be taking 256 class C networks and defining them as a single network, aggregating them together. You can define the networks 207.229.0.0 – 207.229.255.0 with the single routing entry of 207.229.0.0 subnet mask of 255.255.0.0.

CIDR aggregates several networks together for simpler routing, compared to VLSM, which variably subnets a network into smaller networks. Confused? Don't feel bad, so is half of the Internet. To learn more, I highly recommend you read 3Com's Whitepaper on IP addressing, VLSM, and CIDR at **www.3com.com/nsc/501302.html**.

Conclusion

The key to successful routing is your routing tables. By defining a proper routing table, your packets will get from point A to point B. VLSM is a standard that allows greater flexibility in developing a proper routing table. By following the guidelines discussed in this article, your systems will effectively route packets the way you intended.

Jumpstart in a nutshell

by Asim Zuberi

I've been receiving the *Inside Solaris* journal for the past two and a half years and have never seen an article on Jumpstart. Recently, I jumpstarted quite a few workstations at my client site with Solaris 2.6 and noticed that Sun has slightly changed the setup, configuration, and jumpstart procedures from previous releases of Solaris. I had a hard time figuring out how to truly configure Jumpstart on Solaris 2.6, as there was limited documentation available for reference.

After learning the hard way, I decided to compose an article on Jumpstart. A lot has been said and written about the concepts in the book *Automating Solaris Installations* by Paul Kasper and Alan McClellan. If you really

want to understand the concepts of Jumpstart and its limitations, I strongly recommend that you read this book.

Following are just the how-to steps to configure and use Jumpstart on Solaris 2.6. (Even though I've written this article in the scope of Solaris 2.6, these techniques will still be useful if applied to any other previous releases of Solaris.)

What's Jumpstart?

You must have heard of or experienced jumpstarting, as it pertains to the automobile. When the battery of your car dies for some reason, you basically jumpstart the car with one that's already running, thus bringing your car back to life. The same principle applies to installing the OS on a system. You either load or upgrade the operating system on the client systems from a local CD-ROM drive or over the network.

Installing from the local CD-ROM drive is relatively slow and it's limited, and thus takes quite a bit of your time, whereas network installation is much faster and not limited. Since network installation is more flexible, it gained popularity and was named Jumpstart by Sun Microsystems. (A system that is being upgraded or loaded with the new operating system is called the Client, whereas the system that basically makes this all happen is called the Jumpstart server.) So, in other words, you can automate the Solaris installation—make it a hands-off process—and install the operating system for as many clients as you want, all at the same time, saving yourself valuable time.

Jumpstart configuration

A Jumpstart configuration is best approached as a series of steps, which we'll outline here:

1. Gather all system and network information

Before we begin, we need to gather a few facts about our server and target client. Table A shows the information we need, as well as the values for our example.

2. Create the boot/install server

In this example, the boot/install server is the same (as is generally the case), to keep it simple. You'll first need to load the OS image from the Solaris 2.6 CD-ROM onto the server's local disk. You'll need around 350 MB of free space in this directory. So mount the Solaris 2.6 CD-ROM on the server system and do the following:

```
server# cd /cdrom/cdrom0/s0/Solaris_2.6/Tools

server# ls
                Boot                    dial
setup_install_server
                add_install_client
rm_install_client

server# mkdir -p /local/CDimage

server# ./setup_install_server /local/CDimage
                Verifying target directory...
                Calculating the required disk
space for the Solaris_2.6 product
                Copying the CD image to disk...
                Install Server setup complete

server# cd /local/CDimage/Solaris_2.6

server# ls
                Docs      Misc      Patches   Product
Tools
```

Table A: *Our server and client information for our Jumpstart setup*

Server Information	Notes
Name	Server (We're assuming boot server and Jumpstart server is the same system.)
OS image dir	/local/CDimage (You can change the location of Cdimage.)
Config dir	/jumpstart
Client Information	**Notes**
Name	client
Ethernet address	8:0:20:ab:cd:ef (Always available—at the "ok" prompt, type "banner".)
IP address	129.151.29.10 (Clients proposed IP—by the Admin.)
Kernel Architecture	sun4m

3. Create the configuration directory on the server

Now you can create the directory and copy the necessary files in order to perform a custom Jumpstart installation. You set this up by copying the sample directory from the OS image directory (/local/CDimage/...) to the /jumpstart directory:

```
server# mkdir /jumpstart
server# cp -r
/local/CDimage/Solaris_2.6/Misc/jumpstart_sample/*
/jumpstart
```

4. Create a profile for the client machine

The profile file is used as a template for the custom Jumpstart installation. For this example, the profile file is called *client_profile*. The file is created in the /jumpstart directory. This can be created, in many different ways, to suit your individual needs. The following is just the very basic profile, to keep the ball rolling.

```
server# cat /jumpstart/client_profile
  install_type          initial_install
  system_type           standalone
  partitioning          explicit
  cluster               SUNWCuser
  filesys               c0t3d0s0  free /
  filesys               c0t3d0s1  200 swap
  filesys               c0t3d0s3 40 /var
```

5. Create the sysidcfg file

The sysidcfg file is used to automate the system identification portion of the Solaris install. We then need to edit the `/jumpstart/sysidcfg` file:

```
system_locale=en_US
timezone=US/Eastern
timeserver=135.111.130.27  (server's IP address)
network_interface=le0
terminal=dtterm
name_service=NONE   (not running NIS or NISplus )
```

6. Update the rules file

The *rules file* is a text file used to create the rules.ok, and it's probably the most important file for custom Jumpstart installations. You can view this file as a look-up table, consisting of one or more rules that define how install clients are installed, based on their system attributes. In this example, we used the `any` keyword for the first rule (machine attributes) and the file `client_profile` for the fourth rule (profile name), and all others are left blank. ("-" = match always succeeds)

```
server# cat /jumpstart/rules
 any    -   client_profile  -
```

7. Check the rules file

This command is run to validate the rules file. It creates the rules.ok file, which is required by the installation software to match install clients to the predetermined rules. For this example, you should have one line of information in the rules that is *un*commented (`any - client_profile -`). Delete any other uncommented lines in this file that don't pertain to this particular install client *before running the check script*.

```
server# cd /jumpstart

server#  ./check
Validating rules...
Validating profile any_machine...
The custom JumpStart configuration is ok.

server# cat rules.ok     (make sure that
 there aren't any unwanted lines!!)
    any -   -   client_profile  -
```

8. Check to make sure the directories are shared

We need to share the jumpstart and CD image directories for our installation. We then need to edit the `/etc/dfs/dfstab` file:

```
share -F nfs -o ro,anon=0 /jumpstart
share -F nfs -o ro,anon=0 /local/CDimage
```

We then need to start the NFS server (if necessary) and share the directories.

```
server# /etc/init.d/nfs.server (stop | start)
server# shareall

server# dfshares

RESOURCE
   SERVER ACCESS    TRANSPORT
server:/local/CDimage
   server  -      -
```

```
server:/jumpstart
    server  -       -
```

9. Set up the client to install over the network

After setting up the /jumpstart directory and appropriate files, you use the `add_install_client` command on the server to set up the client to install Solaris from the server. You'll also have to add the entry for the client into the local `/etc/hosts` file manually.

```
# Internet host table
#
127.0.0.1           localhost
135.111.130.27  server  timehost loghost
135.111.130.40  client
~
```

We then run the `add_install_client` command.

```
server# cd /local/CDimage/Solaris2.6/Tools
server# ./add_install_client -e
   <CLIENT_ETHERNET_ADDRESS>
-s  <INSTALL_SERVER>:<OS_IMAGE_DIRECTORY
-c  <CONFIG_SERVER>:<CONFIGURATION_DIRECTORY>
-p <CONFIG_SERVER>:<PATH_TO_SYSIDCFG_FILE>
-n [SERVER]:name_service[netmask]
                CLIENT_NAME  ARCHITECTURE
```

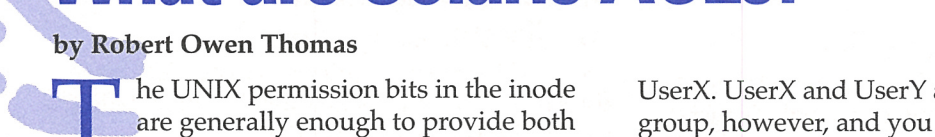
Just as an example of the real system:

```
server# cd /local/CDimage/Solaris_2.6/Tools
server# ls
Boot        dial     setup_install_server
        add_install_client    rm_install_client*
server# ./add_install_client -e 8:0:20:ab:cd:ef
   -s server:/local/CDimage -c server:/jumpstart
   \
   -p server:/jumpstart -n [server]:none  client
   sun4m
```

10. Boot the client and install the Solaris software

Now, on the client machine, just type the following at the okay prompt:

```
ok boot net - install
```



PING THE SOLARIS DUDE: SOLARIS Q & A

What are Solaris ACLs?

by Robert Owen Thomas

The UNIX permission bits in the inode are generally enough to provide both security and filtered access for most installations. However, ever more frequently, there arises an access requirement that the default UNIX permissions can't accommodate. Fortunately, Solaris provides just the enhancement for such situations: Access Control Lists (ACLs).

Those of you who provide care for and feeding of screening routers are already familiar with the concept of ACLs. ACLs provide a means of highly granular access filtering. With Solaris ACLs, additional access permissions and restrictions can be applied to regular files, special files, and named pipes.

Why would you use Solaris ACLs? Consider a situation where a user, UserY, requires read access to a data file owned by another user, UserX. UserX and UserY aren't in the same group, however, and you don't wish to add UserY to UserX's group because it would allow UserY access to other files that she shouldn't be allowed to peruse. You could create another group, of course, but that could lead to an administrative nightmare if this situation arises frequently. Solaris ACLs to the rescue!

With Solaris ACLs, you can allow multiple users to access a file, each with his or her own set of permissions. Using our example above, you would allow UserX to read and write to the data file. UserY, however, would only be allowed to read the file. Members of UserX's group would continue to access the file based on group permissions. With Solaris ACLs, you can selectively grant access on a per-file basis, without confusing additions to groups or, worse, granting worldwide access.

I need this! How do I do it?

The Solaris implementation of ACLs is simple to use. From the command line, all you need are two commands: `getfacl(1)` and `setfacl(1)`.

`Getfacl(1)` allows you to query the file for the ACL settings. Let's check the ACLs on our Really Important Data File, datafile.txt:

```
: yoda; ls -l datafile.txt
-rw-rw----  1 userx    finance     2381
Sep 26 13:37 datafile.txt

: yoda; getfacl datafile.txt

# file: datafile.txt
# owner: userx
# group: finance
user::rw-
group::rw-              #effective:rw-
mask:rw-
other:---
```

The entries indicate:

- **user::rw**—The owner of the file has read and write permissions to the file.
- **group::rw**—The group of the file has read and write permissions to the file.
- **#effective:rw**—This is the result of the interaction of the ACLs with the inode permission bits. This is the real access level, and indicates that the group of the file does have read and write access. More on this later.
- **mask:rw**—The maximum permissions granted to any user, regardless of ACL, except for the file owner.
- **other:**—The permissions for the world.

We see that UserX, a member of the finance department, owns datafile.txt. Other members of the finance group have read and write access to the data file. All other users are denied access to the file. However, UserY, a member of the sales department, requires read access to the file. We don't want UserY to be able to read all the finance files, thus we can't simply add UserY to the finance group. So we need to add an ACL to datafile.txt that allows UserY to read the file. Using `setfacl(1)`, we can grant such access to UserY:

```
: yoda; setfacl -m user:usery:r-- datafile.txt
```

The command portions translate to:

- **-m**—Modify the ACLs for the file.
- **user:usery**—Add an ACL for the user "usery."
- **:r**—Give usery only read perms.
- **datafile.txt**—The file to be modified.

With this simple command line, UserY is now able to read the file datafile.txt. There have been some subtle changes to the file, of course, so let's take a closer look:

```
: yoda; ls -l
total 6
-rw-rw----+  1 userx    finance     2381
   Sep 26 13:37 datafile.txt
```

Note the plus sign (+) after the other permission bits. This is the way Solaris tells you that one or more ACLs exist on this file. What are those ACLs? `Getfacl(1)` will tell us:

```
: yoda; getfacl datafile.txt

# file: datafile.txt
# owner: userx
# group: finance
user::rw-
user:usery:r--          #effective:r--
group::rw-              #effective:rw-
mask:rw-
other:---
```

Here we see that UserY has been added to the ACL list for datafile.txt, with read permission being granted.

How can we use ACLs to restrict access? Simple! Using `setfacl(1)`, we'll keep UserZ, a member of the finance group, from accessing the file datafile.txt. The command line:

```
: yoda; setfacl -m user:userz:--- datafile.txt
: yoda; getfacl datafile.txt

# file: datafile.txt
# owner: userx
# group: finance
user::rw-
user:usery:r--          #effective:r--
user:userz:---          #effective:---
group::rw-              #effective:rw-
mask:rw-
other:---
```

adds an ACL to datafile.txt that restricts UserZ from accessing the file at all. No command is complete without a means of reversing it. Using the `-d` option to `setfacl(1)`, we can remove the ACLs we've put in place:

```
: yoda; setfacl -d user:userz,user:usery
  datafile.txt
```

Note the comma on the command line. We can combine multiple ACLs on a single command line, thus removing the ACLs for UserY and UserX in one command. Now, let's ensure that our changes were successful:

```
: yoda; getfacl datafile.txt

# file: datafile.txt
# owner: userx
# group: finance
user::rw-
group::rw-              #effective:rw-
mask:rw-
other:---
: yoda; ls -l datafile.txt
-rw-rw----   1 userx    finance     2381
Sep 26 13:37 datafile.txt
```

Note that the plus sign (+) is gone from the long listing of datafile.txt. Our file is back to the default of using standard UNIX permission bits for access control.

Hits and misses

There are many other features of `setfacl(1)`, `getfacl(1)`, and the ACL subsystem, including several programmatic APIs. Two of the best features of `setfacl(1)` are the ability to add, modify, or delete multiple ACLs from a single command line entry, and the ability to read in ACLs from an ASCII file.

Everything in Solaris is finite, and ACLs are no exceptions. There can be no more than 1024 ACLs of each type (user, group, other) per file. If you suspect that you might exceed 1024 ACLs for a given type on a given file, you'll need to find an alternate solution. Managing 1024 ACLs on a file would likely be an administrative nightmare.

Only the owner of the file (as reported by `getfacl(1)`) may add or delete file ACLs. Even root cannot add ACLs to a file not owned by root! Keep this in mind when using ACLs.

Note that the interaction of ACLs and permission bits can be confusing. Using our example, we might assume that ACLs take precedence over permission bits. This isn't always the case. For example, if we add an ACL that allows UserY to read datafile.txt, then chmod 600 datafile.txt, we might think that UserY should still be able to read the file. A quick check with `getfacl(1)` tells us otherwise:

```
: yoda; ls -l datafile.txt
-rw-------+  1 userx    finance     2381
Sep 26 13:37 datafile.txt
: yoda; getfacl datafile.txt

# file: datafile.txt
# owner: userx
# group: finance
user::rw-
user:usery:r--          #effective:---
group::rw-              #effective:---
mask:---
other:---
```

Note the `#effective` entry for UserY. The effective permissions allow no access at all. When in doubt, consult `getfacl(1)`!

Unfortunately, the error reporting of `setfacl(1)` isn't the most robust. In fact, unsuccessful attempts to apply an ACL to a file may not result in an error message at all. As with anything in UNIX, check your work! Use `getfacl(1)` to ensure the proper application, modification, or deletion of the ACL in question.

Beware of the filesystem type. ACLs over NFS may or may not be honored. This can be particularly tricky in mixed UNIX environments, as file ACLs aren't a standard. ACLs also don't work on swap filesystems.

If an attempt to access a file results in seemingly impossible *access denied* errors, or a seemingly closed file is accessible, it may be the result of ACLs. Check for the plus sign (+) next to the permission bits in the output of ls –l. You can also use `getfacl(1)` when in doubt.

Take some time to experiment with Solaris ACLs. I recommend using a dummy file until you are fully proficient with the use of both `setfacl(1)` and `getfacl(1)`.

Conclusion

Solaris ACLs give you a wide range of file access options to solve the trickiest of access requirements. With UNIX in general, and Solaris in particular, there's always a way! Ping the Solaris Dude today! Contact the Solaris Dude at **rthomas@dimension.net**

The go command—the cd command gains aliases and a memory

by Alvin J. Alexander

Despite all the hoopla and great things about GUI environments, I spend most of my day pounding away at Solaris command lines on Internet and intranet servers. I don't do this because it's more attractive—I do it because I work faster and more effectively this way. From what I've seen, I believe many other administrators work the same way.

One day, after almost eight years of working at UNIX command lines, I came to realize that there had to be something better than the `cd` command to move between directories. As an administrator, I'm constantly moving from one directory to the next and then back again. To me, there's a lot of wasteful typing in this process.

Sure, I use things like shell aliases as `cd` shortcuts, but they fall apart as soon as I rename a directory. They also require double input—once at the command line and again in a startup file, and I'm too lazy for that.

In a moment of inspiration motivated by laziness, I realized that the `cd` command could easily be improved by giving it built-in aliases and a memory for where it had been before. In that moment the `go` command was conceived, and forever replaced `cd` in my mind.

A simple example

To understand my problems with the `cd` command, let's look at a simple comparison. First, let's examine six `cd` commands that I might normally use to move between four directories. For our purposes, let's ignore everything else I might normally do at the command line, and just focus on the `cd` commands:

```
cd /home/fred/docs/new/laser
cd /usr/local/lib/apache/htdocs
cd products/printers/newlaser
cd ../../../../logs
cd /home/fred/docs/new/laser
cd/usr/local/lib/apache/htdocs/products/
   printers/newlaser
```

For me, that's a lot of typing. Yes, I can shorten the process with a few wildcards here and there. But without throwing wildcards into the equation, let's look at how this same process can be improved:

```
go /home/fred/docs/new/laser
go ht
go products/printers/newlaser
go logs
go laser
go newlaser
```

I think you'll agree this is a significant improvement. At the very least, my typing has been reduced from 187 characters down to 82 characters, which my friends *carpal* and *tunnel* really appreciate.

In addition, I think it's also a more natural way of working, because it works the way we think. Because, like you, the `go` command has a memory of the places you've already been to, it's more intuitive and your typing is greatly reduced.

Using the go command

The `go` command I've created works exactly like the `cd` command, including the use of wildcards (which is why I ignored them in my example above). This is because the `go` command is actually just several shell functions wrapped around the `cd` command.

The power of the `go` command comes primarily from two important capabilities that are not in the `cd` command.

1. Creating aliases

First, you can define *aliases* with the `go` command. Like shell aliases, `go` aliases are shortcut names you can assign that let you move quickly to directories. I use this feature for directories that I travel to frequently, such as my Web server subdirectories.

As an example, let's assume that I want to assign the alias `ht` to the directory `/usr/local/lib/apache/htdocs`. Any time I type `go ht`, I should be moved to Apache's `htdocs` directory. To create this new alias, I just type:

```
go -a /usr/local/lib/apache/htdocs
```

The go command then prompts me for the name of the alias I want to assign to this directory. After assigning a name like ht, I can just type

```
go ht
```

to move to this directory any time in the future.

For a different way to assign the alias, if I'm already in the directory that I want to assign an alias to, I can just type

```
go -a.
```

The "." stands for the current directory. Go uses the pwd command to determine the name of the current directory, and then prompts me as usual to assign an alias to the directory.

To give you an idea of how the alias data is stored, Listing A shows a few sample records from the .go.aliases file.

2. Using the history function

The go command also maintains a history of directories you've visited in the past, so you can easily move back to those directories in the future.

As you saw in one of the first examples in this article, you can go to a directory named laser like this:

Listing A: *The .go.aliases file contains a list of aliases and directory names, with the alias and directory name separated by the "!" character.*

```
..!../..
...!../../..
ht!/usr/local/lib/apache/htdocs
cgi!/usr/local/lib/apache/htdocs/cgi-bin
bin!/usr/local/bin
```

```
go /home/fred/docs/new/laser
```

In the course of your work, you'll visit many other subdirectories on the system. When you need to go back to the directory named laser, just type

```
go laser
```

The go command searches the current directory for a subdirectory named laser. If it doesn't find a subdirectory with this name, it then searches your aliases. If it doesn't find laser in your aliases, it searches the history list (its *memory*). It searches the list in reverse order, and finds the most recent occurrence of a directory named laser. If a match is found, you're moved to that directory; otherwise you get an error message.

Other things you can do with go

Table A provides a list of other things that you can do with the go command, including some of the command-line arguments that go supports.

Installing the go command

If you're interested in using the go command, I think you'll find it pretty easy to install. It's simply made up of a series of Korn shell functions that are all contained in one file.

Note: The go command uses several features of the Korn shell, and loads itself into memory as a series of Korn shell functions. Therefore, you must use the Korn shell at the command line to be able to take advantage of these capabilities. If this proves to be too much of a problem, I may also write the code using another shell, such as the C shell.

Table A: *Possible uses of the go command, including command-line arguments that are supported.*

Command	Uses
Go	Takes you to your home directory
go -	Takes you to your last working directory
go ..	Moves up one level
go ...	Moves up two levels (assuming the alias ... is defined)
go	Moves up three levels (assuming the alias .... is defined)
go ht	An alias I use that takes me to Apache's htdocs directory
go -h20	Prints the last 20 lines in your go history
go -help	Prints a simple help/usage statement
go -a.	Prompts you to add the current directory to your defined aliases
go -t	Lists your defined aliases
go -s cgi	Searches your aliases for anything containing cgi

The first step in the process is downloading the `.go` file from our Web site. Simply ftp anonymously to **ftp.zdjournals.com/sun** and download `.go` as an ascii file.

Once you've downloaded the file, put it in your home directory and name it something like `.go`. Then, put the following statement in your `.profile`, so the functions are loaded into memory each time you log in:

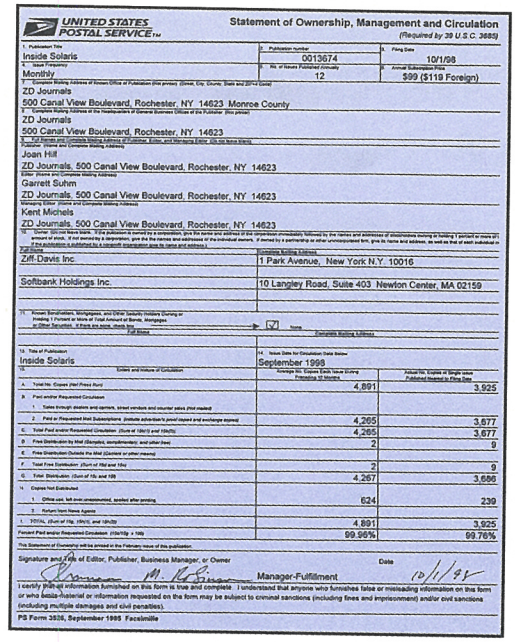
```
. ~/.go
```

When you begin using the `go` command, it creates two other files in your home directory. The first file, `.go_history`, contains the memory for the `go` command. This is a list of directories you've gone to in the past.

The second file is named `.go.aliases`, and it contains the aliases you've defined. A sample of this file was shown earlier in **Listing A**.

Depending on the security requirements at your site, you may want to change the permissions of these files. If you work in an open, trusting environment, mode 644 (`rw-r--r--`) is fine, but if you need more security, mode 600 (`rw-------`) should be used. The important part is that you have the permission to read and write to this file.

The `.go` file also defines a variable named `GO_HISTSIZE` that defines the number of records that `go` keeps in your history file. Once that record-length limit is reached, `go` creates a temporary file named `.go_history.tmp` in your home directory. This file is used when `go` trims the `.go_history` file to the



UNITED STATES POSTAL SERVICE™ — Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685)

1. Publication Title	2. Publication Number	3. Filing Date
Inside Solaris	0013674	10/1/98
4. Issue Frequency	5. No. of Issues Published Annually	6. Annual Subscription Price
Monthly	12	$99 ($119 Foreign)

7. Complete Mailing Address of Known Office of Publication (Not printer): ZD Journals, 500 Canal View Boulevard, Rochester, NY 14623 Monroe County

8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher (Not printer): ZD Journals, 500 Canal View Boulevard, Rochester, NY 14623

9. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor (Do not leave blank)
Publisher (Name and Complete Mailing Address): Joan Hill, ZD Journals, 500 Canal View Boulevard, Rochester, NY 14623
Editor (Name and Complete Mailing Address): Garrett Suhm, ZD Journals, 500 Canal View Boulevard, Rochester, NY 14623
Managing Editor (Name and Complete Mailing Address): Kent Michels, ZD Journals, 500 Canal View Boulevard, Rochester, NY 14623

10. Owner (Do not leave blank. If the publication is owned by a corporation, give the name and address of the corporation immediately followed by the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, give the names and addresses of the individual owners. If owned by a partnership or other unincorporated firm, give its name and address, as well as that of each individual owner. If the publication is published by a nonprofit organization, give its name and address.)

Full Name	Complete Mailing Address
Ziff-Davis Inc.	1 Park Avenue, New York N.Y. 10016
Softbank Holdings Inc.	10 Langley Road, Suite 403 Newton Center, MA 02159

11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities. If none, check box → ☑ None

Full Name	Complete Mailing Address

12. Tax Status (For completion by nonprofit organizations authorized to mail at nonprofit rates) (Check one)
The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes:
☐ Has Not Changed During Preceding 12 Months
☐ Has Changed During Preceding 12 Months (Publisher must submit explanation of change with this statement)

13. Publication Name	14. Issue Date for Circulation Data Below
Inside Solaris	September 1998

15. Extent and Nature of Circulation	Average No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
A. Total No. Copies (Net Press Run)	4,891	3,925
B. Paid and/or Requested Circulation		
1. Sales through dealers and carriers, street vendors and counter sales (Not mailed)		
2. Paid or Requested Mail Subscriptions (Include advertiser's proof copies and exchange copies)	4,265	3,677
C. Total Paid and/or Requested Circulation (Sum of 15b(1) and 15b(2))	4,265	3,677
D. Free Distribution by Mail (Samples, complimentary, and other free)	2	9
E. Free Distribution Outside the Mail (Carriers or other means)		
F. Total Free Distribution (Sum of 15d and 15e)	2	9
G. Total Distribution (Sum of 15c and 15f)	4,267	3,686
H. Copies Not Distributed		
1. Office use, left over, unaccounted, spoiled after printing	624	239
2. Return from News Agents		
I. TOTAL (Sum of 15g, 15h(1), and 15h(2))	4,891	3,925
Percent Paid and/or Requested Circulation (15c/15g x 100)	99.96%	99.76%

16. This Statement of Ownership will be printed in the February issue of this publication.

17. Signature and Title of Editor, Publisher, Business Manager, or Owner: Manager-Fulfillment — Date: 10/1/98

I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including multiple damages and civil penalties).

PS Form 3526, September 1998 Facsimile

One reason to use a function instead of a script

Quick tip

by Alvin J. Alexander

The `go` command, described in the accompanying article, is a good example of a time when you need to create a program as a shell *function* instead of as a shell script. The `go` command just won't work as a shell program.

The problem with a shell program in this case is that it runs in a sub-shell. When the sub-shell exits, you're returned to your regular shell, and the sub-shell can't affect the working directory of your normal shell. Even if the shell program performs the `cd` to the correct directory, you're returned to your previous directory as soon as the sub-shell exits.

To test this, use `cd` to move to the `/tmp` directory. Then put the following lines in a file named `up`:

```
#!/bin/sh
cd /
echo "PWD = `pwd`"
```

Make the file executable, and run the program. Where do you end up? Back in the `/tmp` directory. Your sub-shell moved to the root directory, but as soon as that subprocess ended, you came right back to where you were before.

Inside Solaris™
Tips and techniques for users of SunSoft 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 ..... Alvin J. Alexander, Alan Orndorff, Lance Spitzner, Robert Owen Thomas, Asim Zuberi
Print Designer, Cover and Content Design ..... Lance Teitsworth

General Manager ..... Jerry Weissberg
Editor-in-Chief ..... Joan Hill
Editorial Director ..... Michael Stephens
Managing Editor ..... Kent Michels
Circulation Manager ..... Jeff Marcus
Print Design Manager ..... Charles V. Buechel
VP of Operations and Fulfillment ..... Michael Springer

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 Louisville, KY.

Postmaster: Send address changes to

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

Price

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

Our Canadian GST# is: R140496720.

Back Issues

To order back issues, 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.

ZD Journals publishes a full range of journals designed to help you work more efficiently with your software. To subscribe to one or more of these journals, call Customer Relations at (800) 223-8720.

proper length limit. By default, this limit is set to 128 records.

As a final note, you can change the names of all of these files by editing the `.go` file and changing the variable names that are located near the top of the file. That's all you need to do to install the `go` command.

How go searches for directories

When you're using the `go` command, it's important to understand the search process that `go` uses to find directories. By default, it searches for directories in the following order:

1. If the directory path begins with a slash (`/`), it assumes that it's an absolute path and goes right to that location.
2. If the directory you enter doesn't begin with a slash, it looks in the current directory for a subdirectory with this name.
3. If it's not found in the current directory, `go` searches your list of aliases for this name.
4. If it's not found in your list of aliases, `go` searches the history of places you've been recently.
5. If it still can't find the directory, it gives up the search.

Conclusion and improvements

The `go` command has made my command-line life a little easier and more enjoyable, and I hope it helps you as well. It's amazing what a few aliases and a little memory do to improve the `cd` command.

At this point, I think the `go` command is in pretty good shape, but I still hope to make it better. A few other things that may be added to the `go` command are listed here.

1. A function needs to be added to remove aliases. I'll implement this with the `-r` or `-d` command-line args. Until recently, I've been removing them manually from the `.go.aliases` file, but this obviously isn't a good solution for the masses.
2. The speed of the history search will be improved. The current method isn't too bad, but it can be made faster, which will help those with slower or really busy machines.
3. Check for misspellings. If a directory isn't found, `go` should look a little harder for the name, in case I misspelled it.
4. Support for other shells. The `go` command is currently written for ksh users, and won't work with other shells, unless they support the `ksh` functionality.

Please let me know if you think of other ways to improve this command.

Coming up...

- Using freeware PPP to simplify connecting to remote networks
- Setting up network printers
- Creating automatic filters for your log files

New design and great content bring you a winning combination

To our subscribers:

At ZD Journals, we're always striving to provide you with the liveliest, most informative journals possible. This is why we decided it was time for a face-lift! The changes start with our exciting new cover design. Inside, you'll find great content plus some other changes that we hope will improve the usability of the information. New visuals will accentuate our regular features, making them easier to find. Also, we'll give you a preview of the exciting topics we're working on in our new "Coming Up" section. If you like what you see—or if you don't—please let us know. Our goal is to serve you the best we can and we're eager to hear your comments!

Sincerely,

Joan Hill
Editor-in-Chief
ZD Journals
joan_hill@zd.com