Volume 6, Number 10 • October 2000
www.elementkjournals.com/sun
U.S. $11.00

Inside Solaris™
Tips & Techniques for users of Sun Solaris

Email for a small business

by Don Kuenz

Many small businesses with fewer than two dozen PCs currently use a haphazard, ad hoc email system. Typically, one PC at a time accesses email by laboriously connecting to the Internet with a modem. In this scenario, you should find it easy to convince management to install a Solaris host to consolidate email functionality.

Despite its flaws, email can boost productivity in most businesses. Management intuitively understands that email allows people to think things through instead of making rash statements during the course of a phone call. Effective communication increases the bottom line of every business. In this article, we'll show you how to connect a network of PCs to a Solaris host, which runs sendmail.

We strive to obtain three goals in our email architecture. First, we want our mail server to work with a wide variety of PC software. Second, we want an easy installation. Third, we want to move mail from our Solaris host to the employee's PC.

In our scenario, the company assigns one PC to each employee. We also allow employees to save important mail on a private samba share on the Solaris host. We back this share up each night. Given these constraints, we choose POP3 as our mail delivery protocol.

A small business network

Figure A shows a rather small network. This network contains a Solaris host, which provides mail services to two PCs. The Solaris host connects to an Internet service provider (ISP) using either a dialup modem or a dedicated link.

Notice that this network assigns private IP addresses to the PCs. This practice increases the security of those PCs, because public Internet routers drop packets that contain private addresses. This functionality allows those PCs to remain virtually invisible to the Internet. However, they can still receive and send mail to hosts using public IP addresses.

The Solaris host sends and receives mail on behalf of the PCs. We assign both

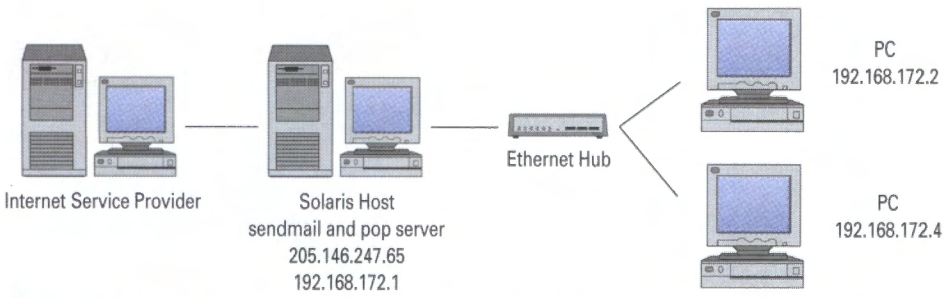

Figure A: *A small business network that uses a Solaris host to provide email services.*

In This Issue

Email for a small business

Hooking into news

Taking advantage of ToolTalk

Space—the final frontier

Replicate filesystems and directory hierarchies with rsync

Creating your own tunnel

elementk JOURNALS
A Division of Element K Press

a private address and a public address to the Solaris host. The public address makes the Solaris host visible to the Internet, while the private address allows the Solaris host to exchange mail with the PCs.

Listing A: *A typical POP3 session*

```
TELNET 192.168.172.1 110
+OK POP3 small.biz v7.64 server ready
USER don
+OK User name accepted, password please
PASS Passw0rd
+OK Mailbox open, 1 messages
RETR 1
+OK 575 octets
Received: from att.net (pc1 [192.168.172.2])
        by small.biz (8.9.1b+Sun/8.9.1) with ESMTP id WAA16491
        for <don@small.biz>; Thu, 25 May 2000 22:57:45 -0600 (MDT)
From: customer@att.net
Message-ID: <392E0449.55EC309F@attglobal.net>
Date: Thu, 25 May 2000 22:57:45 -0600
Reply-To: customer@att.net
X-Mailer: Mozilla 4.7 [en] (WinNT; U)
X-Accept-Language: en
MIME-Version: 1.0
To: don@small.biz
Subject: The is subject.
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Content-Length: 10
Status:

This is the body.

.
DELE 1
+OK Message deleted
QUIT
```

POP3 on Solaris

Let's take a look at the POP3 protocol that we use to enable PCs to retrieve mail from the Solaris host. **Listing A** shows a POP3 session where we retrieve and then delete a single message for user *don*. You can converse with a POP3 daemon by opening a telnet session and specifying port 110, which is a well-known port.

During the session shown in **Listing A**, we enter the following commands: `TELNET`, `USER don`, `PASS Passw0rd`, `RETR 1`, `DELE 1` and `QUIT`. The POP3 daemon responds with OK followed by other information. When you use a mail agent, such as Netscape Messenger or Outlook Express, the agent generates the commands for you.

You'll find it easy to install a POP3 daemon on a Solaris host. First, obtain the source from **ftp.cac.washington.edu/mail/imap.tar.Z**.

This source actually contains three mail daemons: imapd, ipop2d and ipop3d. For our mail server, you only need to install ipop3d. Use `make` to compile ipop3d and place the binary into a handy directory such as /usr/local/sbin.

Next, verify that the following line appears in /etc/services:

```
pop3  110/tcp
```

Add it if you find it missing. Finally, make sure the following line appears in /etc/inetd.conf:

```
pop3    stream  tcp     nowait  root
➥ /usr/local/sbin/ipop3d  ipop3d
```

Again, add it if you find it missing.

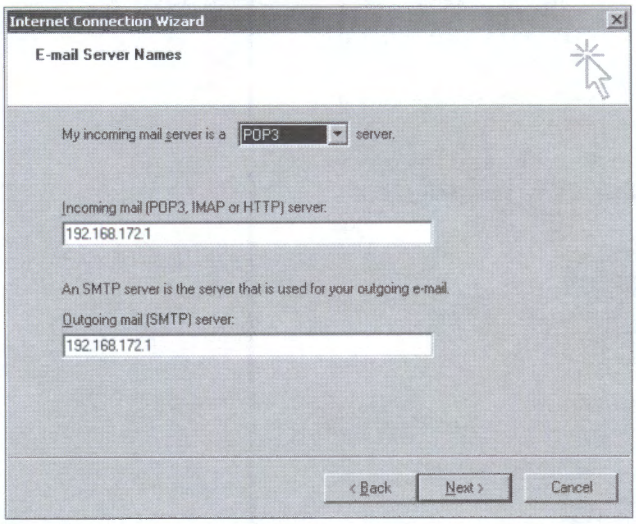

Figure B: *Use these settings to configure Outlook Express to use the Solaris mail server.*

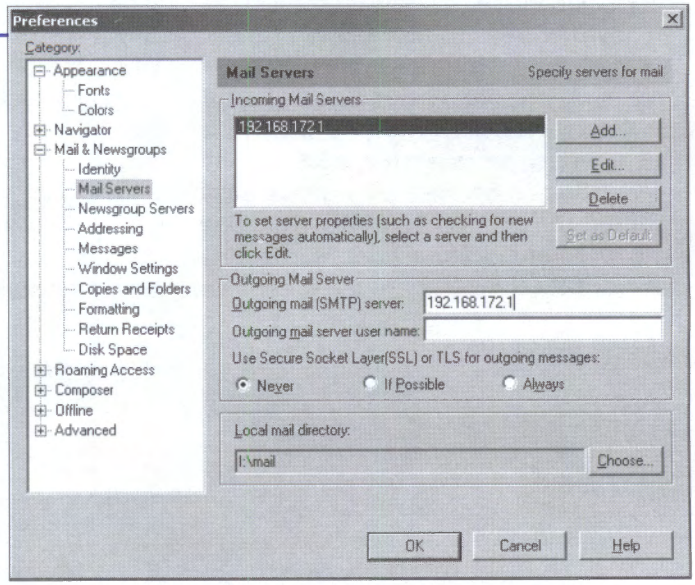

Figure C: *This shows the first Netscape Messenger dialog box, which configures mail. You can add new incoming mail servers and configure your outgoing mail servers here.*

Your Solaris host will now accept POP3 sessions from PCs. You should also find it fairly easy to configure PCs.

Configuring PC clients

Many PCs use either Outlook Express or Netscape Messenger as mail agents to send and receive mail. **Figure B** shows how to configure Outlook Express version 5 to use the mail services on our Solaris host. Outlook Express displays the dialog box shown in **Figure B** after you select Tools | Accounts | Add Mail from the main menu.

Netscape version 4.7 uses two dialog boxes to configure mail services. It displays the dialog box shown in **Figure C** after you select Edit | Preferences | Mail from the main menu. At this point, you can click the Add button to display the dialog box shown in **Figure D**.

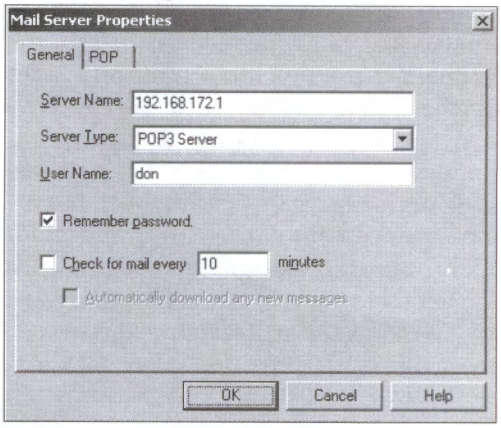

Figure D: *This shows the second Netscape Messenger dialog box that configures mail. Enter your mail server here.*

Conclusion

In this article, we've shown you how to use a Solaris host as a mail server in a small business network comprised of a few dozen PCs. We used the POP3 protocol because it's easy to install, most PC mail agents understand it, and it allows you to push mail onto PCs. ✳

Hooking into news

by Don Kuenz

News, or usenet, allows you to publish public messages, or articles. It works likes a message board. Anyone in the world can post an article to the message board, as well as read all of the other articles already posted. It serves as a tool that allows you to obtain technical help, float new ideas among your peers, buy and sell items, advertise or find a job, pursue a hobby, or visit with folks around the world.

In this article, we'll show you how to set up a simple news server using the INN (pronounced I-N-N) package. INN ranks as one of the more popular news servers in use today and works well for low- to medium-volume news servers.

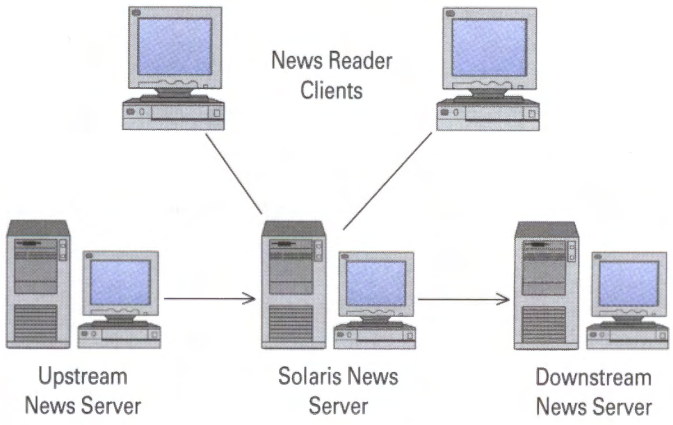

Figure A: *This schematic shows the architecture used by news.*

The architecture of news

News distributes and archives articles. It utilizes a client/server model to process articles from individual users. It also uses a peer-to-peer model to distribute articles between news servers. **Figure A** shows the architecture news uses. Upstream servers and clients feed articles into our news server. In turn, our news server feeds articles to downstream servers and clients. A given host may simultaneously function as both an upstream and a downstream news server.

Listing A, on the next page, shows a typical news article, which looks very similar to email. The topmost lines (before the first blank line) contain header information. Header information tells news servers the article's author, a brief title and the destination group. The actual message follows

the first blank line. News treats all blank lines after the first as part of the message.

The destination group contained in the header tells news where to deliver an article. News delivers the article shown in **Listing A** to a group named comp.unix.solaris. About 100,000 groups now exist, so news organizes articles into a hierarchy in order to keep similar topics grouped together. Computer-related groups begin with comp. Under this top level, you find groups like comp.ai, which concerns artificial intelligence.

Computer-related groups that carry information about UNIX begin with comp.unix. Within this level, you find groups like comp.unix.admin, which contains articles discussing administration. You'll also find comp.unix.solaris, which contains information about Solaris.

A news server assigns a unique group number to each article as it arrives. The `Xref:` header contains the assigned group number. Our server assigned the number 2869 to the article shown in **Listing A**. This number allows client software to browse header information and retrieve a specific article upon demand.

We use INN's traditional storage method to archive articles. The traditional storage method places the article shown in **Listing A** into a file named ~news/spool/articles/comp/unix/solaris/2869. We use ~news to designate the root directory where you install INN.

Listing A: *A typical news article*

```
From: "DonK" <kuenz@gtcs.com>
Subject: ProWorks 3.0 any good on Solaris 7 x86?
Newsgroups: comp.unix.solaris
X-No-Archive: yes
Lines: 9
Message-ID: <8EmS4.86$AT2.8249@news.uswest.net>
Date: Wed, 10 May 2000 23:53:40 GMT
NNTP-Posting-Host: 209.181.16.1
X-Trace: news.uswest.net 958002820 209.181.16.1
➥(Wed, 10 May 2000 18:53:40 CDT)
NNTP-Posting-Date: Wed, 10 May 2000 18:53:40 CDT
Path: apollo!news1.prserv.net!newsfeed.us.ibm.net
➥!ibm.net!news-spur1.maxwell.syr.edu
➥!news.maxwell.syr.edu!newsfeed.berkeley.edu
➥!news-out.uswest.net!news.uswest.net.
➥POSTED!gtcs.com!not-for-mail
Xref: apollo comp.unix.solaris:2869

Question about ProWorks 3.0 for Solaris x86 (originally
released during Solaris version 2.4 era).

Will this run with Solaris 7?

Does Sun offer a discounted upgrade to WorkShop (or whatever
they call their C++ product these days).

TIA
```

Installing INN

You need to compile INN from source code. We suggest that you use the GNU compiler and GNU make tool available at **http://sunfreeware.com**. We also recommend that you install Perl, because INN comes with a lot of Perl scripts to help you manage your news server. You can also obtain a Perl package from the previous Web site. To obtain the latest copy of the INN source distribution, go to **ftp.isc.org/isc/inn/snapshots**.

After you install GNU make, GNU cc, and Perl, you need to add a couple of users and a group before you compile INN. Log on as root and create a directory named */usr/local/news*. This serves as the root directory of your INN installation. Invoke the `groupadd` command to add a group named *news*. Next, invoke the useradd program to add a user named *news* and another user named *usenet*. Make sure that both of these users belong to the news group and use /usr/local/news as the home directory.

Next, log on as user news and invoke the `gzip` and `tar` commands to extract INN's source distribution. Invoke `./configure --with-perl`, followed by `gmake`, and then `gmake install`. User root must own a binary named /usr/local/news/bin/inndstart with *set user id* file permissions. This gives inndstart root access, even though user news invokes it. Use the `ls` command to double check that root owns inndstart with `-r-sr-x---` file permissions. Invoke `chown root inndstart` and `chmod 4550 inndstart` if necessary. Now you must configure INN.

Configuring INN

When you use the traditional storage method, at a minimum you need to configure the following files: inn.conf, newsfeeds, incoming.conf, expire.ctl and nnrp.access. You can find all these files under the /usr/local/news/etc directory. In this section, we'll take a look at each of these files and show you what they should contain.

The *inn.conf* file controls the overall configuration of your news server. Each line in inn.conf contains a parameter name along with a value. One or more spaces separate the name from its value. You need to change the following lines in this file:

```
organization: CRC
server:       localhost
pathhost:     apollo
complaints:   postmaster@crc.com
```

Substitute the values shown with your own organization's name, hostname and domain name.

The file named *newsfeeds* contains the names of downstream news servers. In order to distribute articles locally posted on your server, you must edit this file to include the host names of downstream news servers. **Listing B** shows the pertinent lines. You must uncomment the lines controlling the funnel master and append the lines that specify your downstream server's host name(s).

Of course, you need to replace news.ips.net with the host name of your upstream feed. Further, you need to append the bottom three lines with the appropriate values for each downstream server you feed.

You use the file named *incoming.conf* to specify the upstream news servers that send your articles. At the bottom of this file you must append three lines that contain the host name of your upstream feed. The three lines look similar to the following:

```
peer news.isp.net {
  hostname:  news.isp.net
}
```

Again, you need to replace `news.ips.net` with the host name of your upstream feed. Append these three lines with the appropriate values for each upstream feed you access.

The file named *expire.ctl* contains information that controls how long articles stay in your spool. As distributed, expire.ctl keeps articles for one to 10 days and allows expired headers to work. You may use it as is for the time being.

The *nnrp.access* file allows you to control client (newsreader) access to your news server. Lines in this file provide for read, as well as post access. You may also require password access to your news server. Append a line similar to the following one:

```
192.42.0.0/16:Read Post:::*
```

This line grants read and post access to all hosts whose IP address begins with 192.42. Again, you need to substitute your own IP address ranges.

Starting INN

Before you start INN for the first time, you must populate a few files in your ~news/db directory. Log on as user news and download a file named newsgroups from **ftp.isc.org/pub/usenet/CONFIG**. If you plan to carry most news groups, you can also download a file named active. If you prefer to start small, you can create the active file shown in **Listing C** and use `ctlinnd newgroup` to add groups after your news server starts. Practically speaking, the active file shown in **Listing C** creates one news group, comp.unix.solaris, on your server. You must include the remaining groups, which perform special tasks.

Listing B: *The required entries in a file named newsfeeds*

```
# Innfeed funnel master; individual peers feed into the funnel.
# Note that innfeed with "-y" and no peer in innfeed.conf
# would cause a problem that innfeed drops the first article.
innfeed!:\
    !*,\
    :Tc,Wnm*,S30000:/usr/local/news/bin/startinnfeed -y

# A real-time feed through innfeed.
news.isp.net\
    :!junk,!control/!foo\
    :Tm:innfeed!
```

Listing C: *The minimal entries required in the file named active*

```
comp.unix.solaris 0000000000 0000000001 y
control 0000000000 0000000001 n
control.cancel 0000000000 0000000001 n
control.checkgroups 0000000000 0000000001 n
control.newgroup 0000000000 0000000001 n
control.rmgroup 0000000000 0000000001 n
junk 0000000000 0000000001 y
```

Listing D: *Jobs that must be added to user news's crontab*

```
0 3 * * * /usr/local/news/bin/news.daily expireover lowmark
1,11,21,31,41,51 * * * * /usr/local/news/bin/nntpsend
```

Next, you must create an empty history database by using the `makehistory -i` command. Finally, set the permissions of all files under ~news/db to 0664 by using the `chmod 0664 *` command.

One more task remains before you can start your news server. You need to add the two jobs shown in **Listing D** to the crontab belonging to user news. The first job expires articles and rotates logs. The second job sends articles posted on your local server to downstream news servers.

Finally, you can start your news server with /usr/local/news/bin/rc.news. If everything goes well, you should see two new processes, one for innd and another for innwatch. You can also use innstat to monitor your server. If you run into problems, check /usr/local/news/log/news.crit and the other log files under the log directory for errors.

After innd starts, you can use a tool named ctlinnd to control it. Among other things, this versatile tool allows you to add new groups using `ctlinnd newgroup` and shutdown the news server by using `ctlinnd shutdown` *reason*. ✳

Taking advantage of ToolTalk

by Edgar Danielyan

If you're using the Common Desktop Environment (CDE), and are a frequent user of ps, then you have probably noticed a process called ttsession, quietly sitting in the background. For those from the Windows world, "tt" in ttsession doesn't stand for TrueType; instead it stands for ToolTalk and ttsession is the ToolTalk messaging service server process. In this article, we'll discuss what ToolTalk is, what it does, and how you can use it.

What's ToolTalk?

ToolTalk is a messaging service typically used by CDE applications to exchange messages on the same computer. In some respects, it's like the remote procedure call (RPC); however it's much easier to use and has a much better application-programming interface (API) than the RPC.

Applications utilizing ToolTalk use the ttsession message server to send and receive ToolTalk messages; the server itself is started either manually by the user or automatically by applications requiring it. It's worth noting that the user doesn't interact with the server directly. Instead, ToolTalk-enabled applications use the ToolTalk server to exchange messages. By using ToolTalk, application developers can integrate their independent software into the CDE and provide better connectivity between their applications and standard CDE applications. You may also use ToolTalk effectively in an object-oriented programming environment.

How to use ToolTalk

You're indirectly using ToolTalk when you're using most, if not all, of the CDE applications, such as the File Manager, Mail, etc. You can also use ToolTalk directly in your applications developed in C or C++. For your application to be able to use ToolTalk, it must include ToolTalk header files Tt/tt_c.h and Tt/tttk.h:

```
#include <Tt/tt_c.h>
#include <Tt/tttk.h>
```

You must also link your application with the ToolTalk library libtt. Accomplish this by using the `-l` switch for your C/C++ compiler or linker (`-ltt`). Before starting to write the code, you may want to consider the following:

- What functionality you want from your application
- How you'll implement what you require from your application

Most ToolTalk applications will use the *Desktop Services* protocol subset of the ToolTalk system. The process of developing a ToolTalk application includes defining a scenario by writing down the events and operations your application will be generating, receiving or doing. The next step is to follow that scenario uniformly in your application. All ToolTalk applications have to initialize the toolkit, join a ToolTalk session, and add ToolTalk to its session handler before calling any ToolTalk function or using any ToolTalk data structures. To initialize the ToolTalk session, your application has to obtain a process identifier using the `ttdt_open` function:

```
int ttfdesc;
char *procid = ttdt_open( &ttfdesc, ToolName,
➥"SunSoft", "%I", 1 );
```

Then, join the ToolTalk session using `ttdt_session_join`:

```
newsession = ttdt_session_join( 0, 0,
➥session_shell, xyz, 1);
```

The final step is to add the handler to Xt events loop:

```
XtAppAddInput(myContext, ttfdesc,
➥(XtPointer)XtInputReadMask,
➥tk_Xt_input_handler,procid);
```

After all these preparatory steps, your application is ready to participate in ToolTalk messaging. Your next step is to develop and debug your code.

Debugging and tracing ToolTalk applications

There are two useful utilities provided with the ToolTalk system: TTSnoop and Ttrace, which are used for debugging and tracing ToolTalk messages and applications. TTSnoop is particularly useful and easy to use, as it has an interactive graphic interface that greatly simplifies the task of developing and debugging applications using ToolTalk messaging.

For a more in-depth description of ToolTalk and other CDE features, see the *Solaris Common Desktop Environment Developer* collection (which is available online at **http://docs.sun.com**). The user guide for Tooltalk is available at **http://docs.sun.com/ab2/coll.45.13/TTUG/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1&DwebQuery=tooltalk**. ✳

Space—the final frontier

by Jerry L. M. Phillips

So, your new boss calls you into his office and tells you that he's having trouble saving files on his Sun Ultra 5 workstation running Solaris 7. He wants you to fix the problem—while he waits. Now is your chance to impress him with your abundant UNIX skills.

What's the problem?

With your boss looking over your shoulder, you quickly discover that the problem is twofold. First, he's attempting to save his StarOffice document files to the root partition. Second, the root partition is full, as shown in **Listing A**.

This is simple, you think. You advise your boss that he can save his documents in another directory that you'll set up (e.g., /usr/local). He won't notice the difference. Ah, but then he tells you that he likes to keep his document files on the root partition, not in a subdirectory. The files are easier to find there. Now you face another issue altogether. You have to find space on the root partition. (The root partition was probably sized too small during the original installation.) However, except for the boss's document files, the other files in the root partition appear necessary to the functioning of Solaris. Or, are they?

Big things come in small packages

You decide to examine the packages on the system. A quick way to list all packages on the system is to type the following command:

```
# pkginfo
```

This command prints out a list of the packages on the system with a brief description of each. A sample version of the list appears in **Listing B** on the next page.

All of the packages are located on the root partition. Your sharp eye notices that some of these packages may be unnecessary. Does your boss need the PCMCIA Card services on his Sun Ultra 5 workstation? Is ShowMe TV an application that he'll use? Are German, Spanish, French, Italian and Swedish versions of Netscape Communicator, Solaris 2.7 Sun Hardware Collection, Solaris 7

Listing A: *The root partition / is at 100 percent capacity with 0 KB available*

```
# df -k
Filesystem            kbytes    used   avail capacity  Mounted on
/proc                      0       0       0     0%    /proc
/dev/dsk/c0t0d0s0    1984230 1984230       0   100%    /
fd                         0       0       0     0%    /dev/fd
swap                  285320    3376  281944     2%    /tmp
/dev/dsk/c0t0d0s3    6484885 1875519 4544518    30%    /usr/local
```

Userbook Collection and Solaris 7 Installation Collection needed on his system? These packages, and others like them that can arrive preinstalled, are apt to consume a lot of disk space.

The next step is to remove the unnecessary packages. Before proceeding, you want some confirmation that you aren't removing the wrong packages. You generate a more detailed listing of packages on the system by typing the following command:

```
# pkginfo -l
```

The detailed listing of `pkginfo` allows you to see how much space each package occupies on the partition. For example, you discover that the ShowMe TV Receiver application SUNWsmtvr, occupies 21,321 disk blocks, as shown in **Listing C**. (According to the Solaris *Answerbook*, ShowMe TV is a television system for local- and wide-area networks. You can use it to view and broadcast live or prerecorded video programs on your network.)

Besides targeting packages that aren't useful to your boss, another key is to look for packages that have de, es, fr, it and sv after their names or embedded within their package names (e.g., SUNWitda). Other similar packages are more obvious because they have their language (e.g., Japanese, Chinese, Taiwanese, Korean, etc.) included in the `pkginfo` description.

Feeling a greater level of confidence, you begin removing the unnecessary packages by executing `pkgrm` using each package name as an argument. In **Listing D**, you choose to remove the ShowMe TV Receiver application.

In this case, don't let the following warning messages intimidate you:

- `<shared pathname not removed>`
- `The <SUNWsmtvt> package depends on the package currently being removed.`
- `Dependency checking failed.`

That just means that the SUNWsmtvt and SUNWsmtvr packages are interrelated. (The ShowME TV package actually consists of four packages.)

Better yet, you can type the following command and spare yourself the traumatic warnings. Don't forget, your boss is watching over your shoulder and may question the wisdom of your moves, especially when several ominous warning messages appear on the screen:

```
# pkgrm SUNWsmtvt SUNWsmtvu SUNWsmtvr SUNWsmtvh
```

All's well that ends well

Several packages later, you have recovered approximately 20 percent of the root partition, and

Listing B: *Sample pkginfo display of software package information, including category, name and description*

```
# pkginfo
Application GNUgzip     gzip
Application SUNWAhwde   Solaris 2.7 on Sun Hardware
➥ Collection - de
application SUNWAhwes   Solaris 2.7 on Sun Hardware
➥ Collection - es
application SUNWAhwfr   Solaris 2.7 on Sun Hardware
➥ Collection - fr
application SUNWAhwit   Solaris 2.7 on Sun Hardware
➥Collection - it
application SUNWAhwsv   Solaris 2.7 on Sun Hardware
➥ Collection - sv
application SUNWitdta   Solaris 7 Userbook Collection - it
system      SUNWpcmci   PCMCIA Card Services, (Root)
system      SUNWpcmcu   PCMCIA Card Services, (Usr)
system      SUNWpcmcx   PCMCIA Card Services, (64-bit)
system      SUNWpcmem   PCMCIA memory card driver
system      SUNWpcser   PCMCIA serial card driver
application SUNWsmtvh   ShowMe TV Online Help
application SUNWsmtvr   ShowMe TV Receiver
application SUNWsmtvt   ShowMe TV Transmitter
```

Listing C: *The long format of pkginfo displays all available information about the package*

```
# pkginfo -l
PKGINST:  SUNWsmtvr
NAME:     ShowMe TV Receiver
CATEGORY: application
ARCH:     sparc
VERSION:  1.2.1,REV=08.27.1998
BASEDIR:  /opt
VENDOR:   Sun Microsystems, Inc.
DESC:     ShowMe TV receiver application and support files
PSTAMP:   skew980827114647
INSTDATE: Feb 16 1999 08:36
HOTLINE:  Please contact your local service provider
EMAIL:    showmetv-comments@Eng.Sun.COM
STATUS:   completely installed
FILES:    60 installed pathnames
28 shared pathnames
15 directories
5 executables
21321 blocks used (approx)
```

Listing D: *pkgrm removes packages from the system (note that some removal lines aren't shown for the sake of brevity)*

```
# pkgrm SUNWsmtvr
The following package is currently installed:
 SUNWsmtvr   ShowMe TV Receiver
             (sparc)         1.2.1,REV=08.27.1998
Do you want to remove this package?  y
## Removing installed package instance <SUNWsmtvr>
## Verifying package dependencies.
WARNING:
The <SUNWsmtvt> package depends on the package
  currently being removed.
Dependency checking failed.

Do you want to continue with the removal of this
  package [y,n,?,q] y
## Processing package information.
## Removing pathnames in class <none>
/opt/SUNWsmtv/lib/locale/C/share/showmetv-defaults
/opt/SUNWsmtv/lib/locale/C/share/mc-titles.config
/opt/SUNWsmtv/lib/locale/C/share
/opt/SUNWsmtv/lib/locale/C/help/watchtimer.html
/opt/SUNWsmtv/lib/locale/C/help/comments.html <shared
  pathname not removed>
/opt/SUNWsmtv/lib/locale/C/help <shared pathname not removed>
/opt/SUNWsmtv/lib/locale/C/LC_MESSAGES/splitmov.cat
/opt/SUNWsmtv/lib/locale/C/LC_MESSAGES/showmetv.cat
/opt/SUNWsmtv/lib/locale/C/LC_MESSAGES <shared pathname
  not removed>
/opt/SUNWsmtv/lib/locale/C <shared pathname not removed>
/opt/SUNWsmtv/lib/locale <shared pathname not removed>
/opt/SUNWsmtv/lib/lib-SVR4/xil_rtvc_h261_ucode.a <shared
  pathname not removed>
/opt/SUNWsmtv/lib/lib-SVR4 <shared pathname not removed>
/opt/SUNWsmtv/lib <shared pathname not removed>
/opt/SUNWsmtv/bin/splitmov
/opt/SUNWsmtv/bin <shared pathname not removed>
/opt/SUNWsmtv/ShowMeTV/share/showmetv-defaults
/opt/SUNWsmtv/ShowMeTV/share/mc-titles.config
/opt/SUNWsmtv/ShowMeTV/share/ShowMeTV_Intro.mov
/opt/SUNWsmtv/ShowMeTV/share/Keep_It_Pure.mpg
/opt/SUNWsmtv/ShowMeTV/share
/opt/SUNWsmtv/ShowMeTV/man/man1/splitmov.1
/opt/SUNWsmtv/ShowMeTV/man/man1
/opt/SUNWsmtv/ShowMeTV/man
/opt/SUNWsmtv/ShowMeTV/bin/showmetv <shared pathname not removed>
/opt/SUNWsmtv/ShowMeTV/bin/bin-SVR4/splitmov
/opt/SUNWsmtv/ShowMeTV/bin/bin-SVR4/catmov
/opt/SUNWsmtv/ShowMeTV/bin/bin-SVR4 <shared pathname not removed>
/opt/SUNWsmtv/ShowMeTV/bin <shared pathname not removed>
/opt/SUNWsmtv/ShowMeTV <shared pathname not removed>
/opt/SUNWsmtv <shared pathname not removed>
## Updating system information.

Removal of <SUNWsmtvr> was successful.
```

the boss is happy, for the moment. It's best to think about scheduling a time when you can repartition the drive. That's another article entirely. Or, perhaps you can talk the boss into adding another drive and moving his application and document files to the new drive. Some requests can be tough, but sometimes they can encourage you to broaden your horizons and enhance your skill set. ✳

Replicate filesystems and directory hierarchies with rsync

by J.D. Baldwin

Copying files between systems is a common task for a system administrator, and there are many ways to accomplish it. Anyone who has been a system administrator for any time at all can use ftp. Most of us have used rcp at one time or another. A few intrepid (and trusting) souls have even used rdist. All of these methods suffer from the same basic deficiencies: they are generally insecure, or they're difficult to automate.

rsync, an open-source utility developed by Andrew Tridgell (the primary author of the Samba file-sharing utility) and Paul Mackerras, is a program for copying files between UNIX systems. It's flexible, powerful and incredibly fast; and it can be made highly secure with some additional effort. Before going into the details, let's look at a couple of examples.

Examples of rsync

The most basic use of rsync looks pretty much like rcp. (This isn't accidental.) For example, to copy the file /mydocs/thesis.doc from server *potter* to

the subdirectory *doctorate* of the current directory on the local machine, enter

```
$ rsync potter:/mydocs/thesis.doc doctorate
```

To copy all remote files with the .doc extension into the same directory, the command would be

```
$ rsync "potter:/mydocs/*.doc" doctorate
```

(The double quotes are, of course, to prevent the shell from acting on the wildcard.)

So far, rsync hasn't done anything particularly interesting for us that we couldn't have done with ftp or rcp. But what if we want to copy all .doc files with the exception of those that start with *chap7*? (Suppose, for example, chap7a.doc, chap7b.doc and chap7c.doc are newer versions on the local host, and we don't want them overridden.) rsync lets us do this:

```
$ rsync --exclude "chap7*" "potter:/mydocs/*.doc"
➥doctorate
```

Or suppose we want to copy only those .doc files that are newer than the ones on the local host:

```
$ rsync -u "potter:/mydocs/*.doc" doctorate
```

(The `-u` option stands for *update only*.)

Let's look at one final example before moving on to the meat of rsync. We want to create a mirror of our home directory on remote server malfoy, as a nightly backup to disk on a remote server. Note that the remote subdirectory to which we send the data may already have a copy. We want to ensure that any files deleted in the original get deleted in the copy; any symlinks get transferred as symlinks, not as the files to which they point; and that all access and modification times are preserved.

rdist, rcp, and obviously ftp all fall woefully short for what ought to be a fairly simple task. But rsync handles it easily:

```
$ rsync -a --delete /home/jbaldwin/ malfoy:
➥/backups/homedirs/jbaldwin
```

In this example, `--delete` means to delete any files existing on malfoy that don't match files existing in ~; the `-a` option is a way of saying, "Make everything an exact copy, including symlinks and times." The trailing slash on the source directory is important, for reasons we'll discuss later.

A few of you are already wondering if you can use rsync to copy files from remote server A to remote server B, without involving the host on which the rsync command is invoked. No, it can't. You'll have to log on to either A or B and invoke rsync from there.

How rsync works

rsync establishes a connection—either an rsync protocol connection (discussed below), or a shell transport (using rsh)—between machines and begins exchanging information about the files to be copied. It doesn't automatically copy every file requested. It first checks, using an MD5 checksum (this is a default; stronger checksum algorithms are available if needed), whether the file on the destination system is truly any different from the file on the source system. If it is, the file is copied; if not, it's left alone.

The effect of this feature is trivial for small transfers, but for systems that replicate thousands of files totaling tens of gigabytes every night, the difference in time could be a few orders of magnitude. One of our systems updates 16 GB of data every four hours, comprised of about 5.5 million files, in around three to 10 minutes, depending on how many files have changed since the previous replication. Other methods tried, including rcp, rdist and copying over NFS, took hours for each replication.

Obtaining and installing rsync

rsync is free software. You may copy, use and redistribute it as much as you like, and there are no license fees of any kind associated with it. Of course, the other side of this coin is that it comes with no warranty or support. You'll have to support it yourself (though aid from the rsync community is easily obtained—more on this later). rsync is distributed under the GNU General Public License. For further details, see the copy of the license that comes with every rsync distribution.

Note that rsync requires no special privileges (such as root access) to install or run, though of course installing and running it as a non-privileged user may limit its usefulness for general system tasks. If you wish to run rsync as a daemon, you'll need to do so as root. We'll assume that you have root access and are installing and configuring rsync in some system area (in this case, /usr/local/bin) for use by anyone on a system.

The primary site for obtaining rsync is http://rsync.samba.org. Click on the download link and you're presented with various options.

Probably the easiest and best in the long run is to obtain the source code, unpack it on your machine, and go through the traditional build and install cycle:

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

If that option isn't feasible because you don't have and can't obtain a C compiler, the rsync site also provides binary distributions for a number of platforms. A binary of rsync 2.4.3 (the current version at this writing) for Solaris 2.5.1 and later is available there. Note that these are available in gzip format—if you don't have gzip on your system, you can obtain a binary of it at **www.sunfreeware.com**.

Building rsync, and installing and using gzip, are well documented in the accompanying README files and won't be covered in detail here. If you should download the rsync binary distribution, the actual installation is extremely simple (assuming the unzipped tar file is in /tmp):

```
# cd /usr/local
# tar xf /tmp/rsync-2.4.3.tar
```

(The actual filename is longer.) This will create bin and man directories under /usr/local, if needed, and install the rsync binary and man pages for rsync and rsyncd.conf in the appropriate places.

Configuring rsync and the rsync daemon

Now that rsync is installed, and assuming you have /usr/local/bin (or wherever you installed it) in your PATH, you can begin running rsync. You can use rsync in this form, without any further configuration, to copy files to and from any system on which you have r-service (rlogin, rsh or rcp) privileges (as long as rsync is installed on both hosts). This is because rsync uses rsh itself to do its work, in the previously mentioned form (unless it's being used to copy files from one location to another on the same machine, which can be another way to use it).

Unfortunately, rsh requires the establishment of trust relationships, either system-wide (through /etc/hosts.equiv) or personal (through ~/.rhosts). Trust relationships are often, and with good reason, frowned upon in security-conscious environments. And if your environment isn't security-conscious to this degree, it probably ought to be. So we'll now examine a way to give machines running rsync a little tighter control over just who does and doesn't have access to their files.

You accomplish this by setting up an rsync server, which runs rsync in daemon mode. It may be invoked by simply executing the regular rsync binary with the `--daemon` parameter. You may do this either at system startup, or from within inetd by modifying inetd.conf. For further details, see the documentation accompanying the rsync distribution.

When running in daemon mode, rsync uses a special protocol for client-server communications. It listens on TCP port 873 (this is selectable with the `--port` option), so if you need to use rsync across a firewall, you'll want to enable that traffic. The rsync protocol is extremely flexible and powerful, and controllable through a configuration file.

That configuration file is /etc/rsyncd.conf, which is documented in the rsyncd.conf man page distributed with rsync. In it, one may designate multiple-named services, called, as in Samba, *modules*, each with completely different configuration parameters.

Consider the following example /etc/rsyncd.conf file on server malfoy, which first defines a few global parameters and then goes on to define two modules, `submissions` and `assignments`, each with its own parameters:

```
# Global parameters for server malfoy:
uid = nobody
max connections = 23
dont compress = *.gz *.tgz *.z *.zip
use chroot = yes

# Parameters for the submissions module
[submissions]
comment = class work submission area
list = no
uid = root
auth users = jdoakes, jwayne, jbaldwin
secrets file = /etc/rsyncd.secrets
path = /classes/submissions
read only = no

# Parameters for the assignments module
[assignments]
comment = assignment download area
path = /classes/assignments
hosts deny = *.support.hogwarts.edu
read only = yes
```

There are 29 named parameters that may be specified in rsyncd.conf, and all are well documented in the man page. The ones used in this example are the only ones we'll examine here, and should be enough to get you running a basic installation of rsync.

In the global parameters, we see that the UID with which file transfers will take place is `nobody`, and that rsync on this machine will permit a maximum of 23 simultaneous users. Furthermore,

files with extensions indicating they have already been compressed won't be recompressed by rsync, even if the user asks for this (by specifying `-z` on the command line).

The `use chroot = yes` line is a little trickier. It causes a specific directory to become a sort of local root to the rsync process, making it (theoretically) impossible for the process to access any files outside that path. It's intended as an extra security measure, but, like many extra security measures, it has hidden pitfalls. If you're transferring symlinks, for example, you probably want to specify the use of `chroot = no` for that module, or else your symlinks will be rewritten in ways you may not expect.

Next come the parameters that apply only to the `submissions` module. The first item specified is an optional comment field. This may be an explanatory note to others who may be listing rsync modules from a remote system. In this case, it's for internal documentation only, because the `list = no` parameter means that public listing of this module isn't permitted. A user on another system who types the command

```
$ rsync malfoy::
```

(which is a notation meaning "list the rsync modules provided on malfoy") would see only

```
assignments    assignment download area
```

In this situation, submissions is known as a hidden module.

The `uid` parameter then overrides the global `uid` parameter. It specifies that all transfers taking place under this module are to take place as the superuser. This is necessary if we wish to use the command-line option that preserves the original ownership of the files transferred.

The `auth users` parameter lists those user IDs permitted access to this module. This is for authentication only. Once authenticated, read and write access to files is determined on the basis of the uid specified in the rsyncd.conf file (in this case, nobody). An `auth users` parameter requires a secrets file, specifying a file in which `username:password` (with password in plain text) combinations are held for authentication. Of course, the secrets file must not be world-readable. (rsync will fail with an error message if anyone other than the file owner has read permission on this file.) These passwords are for rsync use only, and have nothing to do with system login passwords. It should go without saying that they should *never* be set to be the same as the system login password for any account.

The `path` parameter specifies the path to which all specified paths will be judged as relative. So, when a user puts a file in submissions/cs435, it will end up in absolute path /classes/submissions/cs435.

Modules are read-only by default, so a `read only = no` line is required if users are to be permitted to write to these file areas. For the `assignments` module, the fields mean the same things. The only new item here is the hosts `deny` parameter, which specifies that all access will be refused to connection attempts coming from hosts in the support subdomain. If you use `hosts deny` and `hosts allow`, keep in mind that matches (wildcard matching, as in the example, is permitted) will be done against the hostname as the system resolves it. This may or may not be the same as what you think it will resolve as; that is, you may expect the host name only and get the fully qualified domain name, or vice-versa. Be sure of your hostname resolution scheme and test a lot before relying on these directives.

Note also that this module is read-only. This directive really makes no difference, as read-only access is the default, but it's a good idea to specify it in every module for purposes of clarity.

Using rsync at the command line

Keeping in mind the rsyncd.conf file listed, we'll now see some example uses of rsync commands in conjunction with these two modules. All of the options discussed here are applicable as well to file transfers using shell transport (as discussed previously) and those taking place entirely on the local system.

Note first that any file transfers performed using the rsync daemon *must* use either the `submissions` or `assignments` module. There's no generic, or global module in place, despite the presence of global parameters. This is sometimes a source of confusion to those new to rsync.

Now let's move to a concrete example. Suppose user jdoakes wishes to upload his assignment, which consists of multiple files and directories in a hierarchy starting with the directory halting_problem in his home directory on his own workstation. In order to transfer it to server malfoy, he might type

```
$ rsync -rlptgoD ./halting_problem
➥ malfoy::submissions/cs999/
```

after which the system will require a password. He must, at this point, type the password associated with his logon in /etc/rsyncd.secrets on malfoy in order to authenticate the operation.

This command, once authenticated, will cause the contents of halting_problem and all files and subdirectories to be copied recursively into /classes/submissions/cs999/halting_problem on server malfoy. A trailing / on `halting_problem` in this command would cause the files and directories under halting_problem to be copied directly into /classes/submissions/cs999. The trailing slash on the destination (after `cs999`) is meaningless and may be omitted without effect. The options provided mean the following:

- `-r` copies all subdirectories recursively; without this option, only files in halting_problem would be copied.
- `-l` copies all symbolic links and preserves their contents; note that copied symlinks may be disrupted due to the `use chroot` option in rsyncd.conf.
- `-p` updates the remote permissions to be the same as on the local files.
- `-t` updates file modification times on the remote host.
- `-g` updates the group ownership of every remote file to match those on the local host.
- `-o` updates the owner of every remote file to match those on the local host (because of these two options, the remote files will be owned by jdoakes instead of root; that's why rsyncd.conf had to specify a uid of root for this module—more about this later).
- `-D` re-creates all character and block devices on the remote host (this option also requires a uid of root specified in rsyncd.conf).

This seems like an unlikely combination of options for a simple class assignment upload, but we've combined them in this way to make the point that the `-a` (for archive) option subsumes all of these. Thus, the command

```
$ rsync -a ./halting_problem
➥malfoy::submissions/cs999/
```

is exactly equivalent to the above. `-a` is a good general option that means, "Make a copy of this hierarchy as exact as possible at the destination."

As with the `copy` (and `rcp`) command, multiple files (or wildcard expressions representing multiple files) may be specified for the source, with a single destination. Thus, the command

```
$ rsync -a ./halting_problem
➥./linear_np malfoy::submissions/cs999/
```

would copy both directories into the same destination location.

Now let's look at transfers in the other direction—downloads. The most simple command one can give specifies a remote source file (or wildcard) with no local destination. This is a way of requesting a listing of available files from the remote server. The following code

```
$ rsync "malfoy::assignments/*"
```

might result in the listing

```
skipping directory /square_circle
skipping directory /elegant_flt_proof
skipping directory /perpetual_motion
-r--------  4177 2000/06/10 11:00:15 all.txt
```

which shows the names of subdirectories and the attributes of one regular file in the root directory (actually /classes/assignments on malfoy). If we're interested in the contents of directory square_circle we type

```
$ rsync "malfoy::assignments/square_circle/*"
```

which might yield

```
-r--------  11409 2000/06/10 12:12:05 diagram1.gif
-r--------  8223 2000/06/10 12:21:48 diagram2.gif
-r--------  16339 2000/06/10 13:11:16 diagram3.gif
-r--------  5772 2000/06/10 11:01:14 proof.txt
-r--------  1283446 2000/06/10 14:51:31 handwaving.mpg
```

The `-r` (recursive) option works here, as well, and simply typing

```
$ rsync -r malfoy::assignments
```

would list every file available under this module, along with their attributes.

Receiving a file or files from a module like this is simple and straightforward. Simply invoke rsync with a remote source specifier (wildcards are acceptable, of course, but don't forget to escape or quote them) and a local destination. Thus,

```
$ rsync -rvz malfoy::assignments/perpetual_motion
➥ ./myassignments
```

creates the directory perpetual_motion under ./myassignments, containing the same files and directories as the perpetual_motion directory on

malfoy. We've also introduced two new options here: -v (verbose) will cause every file transfer to be listed back to the terminal, and -z will cause rsync to compress files before transfer (and uncompress them afterwards), except for the types specified in the don't compress line of /etc/rsyncd.conf.

We've already mentioned the --delete option in the introductory section. Note that any transfer not specifying this option will transfer all of the requested data, but leave any extra files on the destination intact. In order to specify their removal, you must include --delete. We use a command similar to the following to accomplish mirroring of filesystems between systems:

```
# rsync -az --delete hagrid::xfer/remote_fs/ /local_fs
```

When this is finished, the two filesystems will be exact mirrors of each other, right down to special device files and symlinks. (Recall, however, the special problems using chroot causes with respect to the latter.) Note the trailing slash on the remote (source) specifier. This ensures that the contents of remote_fs get written directly into local_fs, instead of having a directory called remote_fs created under local_fs. The distinction is critical, and you should review this article's examples and the rsync documentation until you understand it thoroughly. If we'd left the trailing slash off, the only thing in /local_fs would be a directory called remote_fs, containing all of the files and directories in remote_fs on hagrid.

Automation of replications

The requirement for authentication for a non-anonymous rsync server presents a problem for the admin who wishes to automate the periodic replication of a filesystem. The requirement for password input for checking against the rsyncd.secrets file doesn't mesh well with shell scripts and cron jobs.

You can provide authentication in an automated environment by setting the RSYNC_PASSWORD environment variable to the desired password before invoking rsync. A better approach would be to put the password in a mode 600 file and using the option --password-file=<file> to have rsync read the file and use the password therein for its authentication.

An even better approach would be to set up SSH as rsync's shell transport, and configure it to permit non-interactive authentication based on machine identity and very restricted (and strongly authenticated) trust relationships. We'll discuss SSH briefly in the next section.

Security concerns

Recall that the assignments module previously discussed is an anonymous rsync module—no login and password information is required to retrieve information from it. All that's required is that the connecting host not resolve to one that matches *.support.hogwarts.edu. Any connection from any other machine will be permissible. Recall also that this module is read-only; no uploads are possible.

The submissions module is more problematic from a security standpoint. It must operate as root, since this is the only way users can use the -o option to preserve ownership of their files. However, the end result is that any authorized user may write root-owned files—including setuid-root files—to the remote system. Users might also, either inadvertently or intentionally, overwrite the files of others.

There are a few ways to address these security concerns. For some environments with a small number of reasonably sophisticated users, it may be appropriate simply to trust them not to misuse the power of rsync in this scenario. (Environments where system administrators with root access are sharing files is an example of this.)

Another option is to create a separate module for every individual user, with a specific directory and a uid = <username> directive. This approach, combined with authentication, would address most of the security deficiencies. For large numbers of users, the addition and deletion of modules could be automated.

If you need maximum flexibility for users to copy their own files among systems, as well as maximum security for authentication of both individuals and machines, there's another solution. To achieve this level of security, you must abandon the rsync server model we've been discussing, and go back to using a shell transport. This is *not*, however, the rsh service we first discussed; you'll want to use SSH.

SSH is an open-source utility for obtaining a strongly encrypted and authenticated shell session on a remote host. It comes with its own file-copying utility, called scp, that can securely transfer files among servers (though it's not nearly as efficient or flexible as rsync).

If you have SSH installed, it can be configured to allow access, with or without an interactive password, between systems based on the invoking user, the connecting system, or a combination of the two. To use rsync with SSH, specify the shell transport with the -e option, like so:

```
$ rsync -a -e ssh predictions
➥malfoy::divination/final_exams/hpotter
```

For a copy of SSH (a commercial product, freely available to some users) or OpenSSH (a truly free version of the same utility), visit **www.ssh.fi** or **www.openssh.com**, respectively. Further discussion of SSH is beyond the scope of this article.

So, go use it!

We've only touched on perhaps one third of rsync's options, features and capabilities. Still, this article should give you the basics you need to go out and get started. The rsync and rsyncd.conf man pages should tell you just about anything else you need to know.

You'll find that **http://rsync.samba.org** has a wealth of documentation, information and resources, including an archive of the rsync mailing list. If you need support, you can usually get a quick answer from this source. If you're supporting an implementation of rsync that's performing a mission-critical service (we support a system that updates data on a critical disaster-recovery machine using rsync), subscribing to the mailing list is imperative.

You can subscribe to the rsync mailing list by sending an email to **listproc@samba.org** with the words *subscribe rsync* in the body of the email. Alternatively, you can check out the Samba list Web interface at **http://lists.samba.org/cgi-bin/weblist**. ✳

About our contributors

J.D. Baldwin is a UNIX and security consultant living in west Michigan. He's a graduate and former faculty member of the U.S. Naval Academy in Annapolis, Md. and has a master's degree in computer science from the University of Maryland. He can be reached at **baldwin@netcom.com**.

Edgar Danielyan is currently self-employed. He's a Cisco Certified Network Associate, has a diploma in company law from the British Institute of Legal Executives, and is a certified paralegal from the University of Southern Colorado. Edgar has been working as a network administrator and manager of a top-level domain of Armenia. 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, USENIX, CIPS, ISOC and IPG, to name a few. He can be reached at **edd@danielyan.com**.

Don Kuenz works at Computing Resources Company (**http://gtcs.com/crc**). They provide programming, administration and hardware for Sun and PC platforms. You can reach Don at **kuenz@gtcs.com**.

Jerry L.M. Phillips, M.S. is director of the Database Center at Eastern Virginia Medical School. In addition to his administrative duties, he manages Sun/Solaris-based platforms for the medical school, including DNS, sendmail, WWW, anonymous FTP, proxy and library servers. Jerry is also an Oracle DBA and manages Oracle production and development database servers for business and student information system services running on Sun/Solaris servers as well.

Inside Solaris™
Tips & Techniques for users of Sun Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by Element K Journals, a division of Element K Press, 500 Canal View Boulevard, Rochester, N.Y., 14624.

Customer Relations

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

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

Element K Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at journals@element-k.com.

Editorial

Editor .....Garrett Suhm

Assistant Editor .....Jill Suhm
Managing Editor .....Michelle Rogers
Assistant Managing Editor .....Dianne Galloway
Copy Editors .....Rachel Krayer
Glenna Lechner
Contributing Editors .....J.D. Baldwin
Edgar Danielyan
Don Kuenz
Jerry L.M. Phillips, M.S.
Print Designer .....Rachel J. King
Cover and Content Design .....Melissa Ribaudo

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) 272-0064

Or contact us via Internet email at inside_solaris@elementkjournals.com.

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

Element K Journals

General Manager .....Kelly Baptiste
Manager of Customer Relations .....Nicole Pate
Manager of Operations .....Cristal Haygood
Manager of Print Design .....Ian Caspersson
Manager of Product Marketing .....Mike Mayfield
Senior Product Marketing Manager .....Brian Cardona

Postmaster

Periodicals postage paid in Rochester, N.Y., and additional mailing offices.

Postmaster: Send address changes to

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

Printed in the U.S.A.

Price

Domestic .....$129/yr ($11.00 each)
Outside U.S. .....$149/yr ($13.00 each)

Our Canadian GST# is: R140496720. CPM# is: 1446703.
QST# is: 1018491237.

Back Issues

To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost $11.00 each, $13.00 outside the U.S. You can pay with MasterCard, VISA, Discover, or American Express.

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 journals@element-k.com.

Coming up...

- Visualizing CPU activity
- Partition switching

Creating your own tunnel

Security is a major concern for almost everyone these days. Every day we're bombarded with stories of hackers who have compromised another critical business system. Many tools (both proprietary and open-source) have become essential to hardening our systems from attack. One open-source tool that's useful in the battle against hackers is an application called VTun. VTun stands for Virtual Tunnel and is available from **http://VTun.sourceforge.net**. VTun was written by Maxim Krasnyansky and is supported by numerous contributors.

According to the VTun FAQ, "VTun is the easiest way to create Virtual Tunnels over TCP/IP networks with traffic shaping, compression, and encryption. It supports IP, PPP, SLIP, Ethernet and other tunnel types. VTun is easily and highly configurable, it can be used for various network tasks." So how can this help you? We've found one use that makes VTun almost indispensable. Many virtual private network (VPN) products have problems when you try to connect between two LANs on the Internet and both sides (or even one) are behind firewalls using translated addresses (NAT'd). This is because most VPN software will try to establish a connect back to the client, which will be blocked by the client's firewall.

For example, your network at the office may be running Checkpoint's firewall product, requiring access via a VPN using their SecureRemote software. Unfortunately, you have a cable modem with a Linux firewall protecting your Solaris box at home. SecureRemote isn't available in this environment, and wouldn't work through your Linux firewall anyway. VTun gets around this problem by needing only one port for communication. Once communication is established, a network is set up between the client and the server. You can then route packets onto this network from each side of the encrypted tunnel. You now have secure access to all resources on your corporate network. All you need is a client machine at work that can connect to your VTun server at home. Of course, you'll also need to open the port on your firewall through which VTun communicates (which is user-definable). **Figure A** shows how a setup like this would work.

VTun works its magic by talking through a driver called TUN. This driver simulates a network device to allow low-level kernel IP tunnelling. Unfortunately, there is no TUN driver available for Sparc at the time of this writing. This means that VTun won't currently work with Sparc-based Solaris releases. However, precompiled packages are available for Linux as well as Solaris x86.

VTun is quite easy to set up. A simple script on both the client and the server called VTund.conf provides the ability to alter encryption, compression, port, routing, and other parameters. If you have a configuration that requires a VPN, VTun is worth checking out. ✳

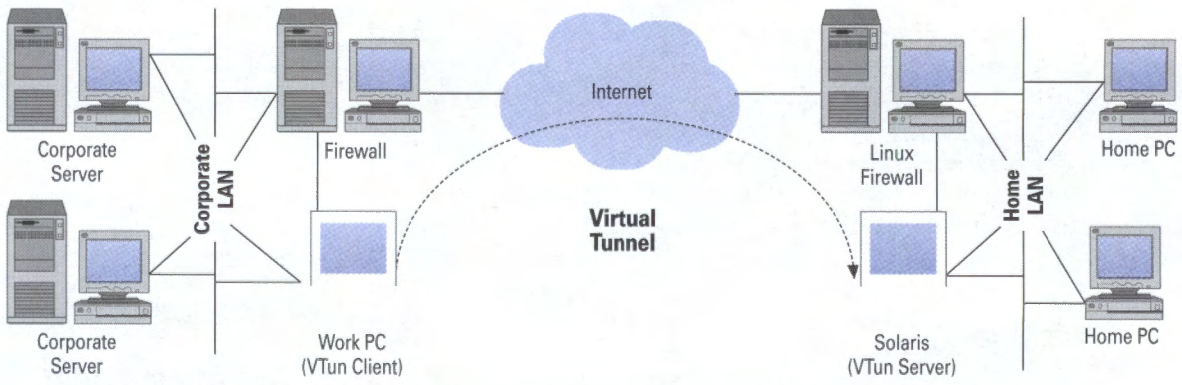

Figure A: *Using VTun to create a VPN between two private networks across the Internet.*