



# 91

# AIX

*May 2003*

---

## In this issue

- 3 Keep the wraps on those TCP ports
  - 10 AIX5L System V Release 4 print subsystem
  - 26 Illustrated usage of various shell commands and shell features
  - 39 Terminal capabilities
  - 51 AIX news
- 

© Xephon plc 2003

update

# AIX Update

---

## Published by

Xephon  
27-35 London Road  
Newbury  
Berkshire RG14 1JL  
England  
Telephone: 01635 38342  
From USA: 01144 1635 38342  
E-mail: [trevore@xephon.com](mailto:trevore@xephon.com)

## North American office

Xephon  
PO Box 350100  
Westminster, CO 80035-0100  
USA  
Telephone: 303 410 9344

## Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; \$275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1999 issue, are available separately to subscribers for £16.00 (\$24.00) each including postage.

## AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at <http://www.xephon.com/aix>; you will need to supply a word from the printed issue.

## Editors

Trevor Eddolls

## Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

## Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 (\$260) per 1000 words and £100 (\$160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 (\$80) per 100 lines. In addition, there is a flat fee of £30 (\$50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from [www.xephon.com/nfc](http://www.xephon.com/nfc).

---

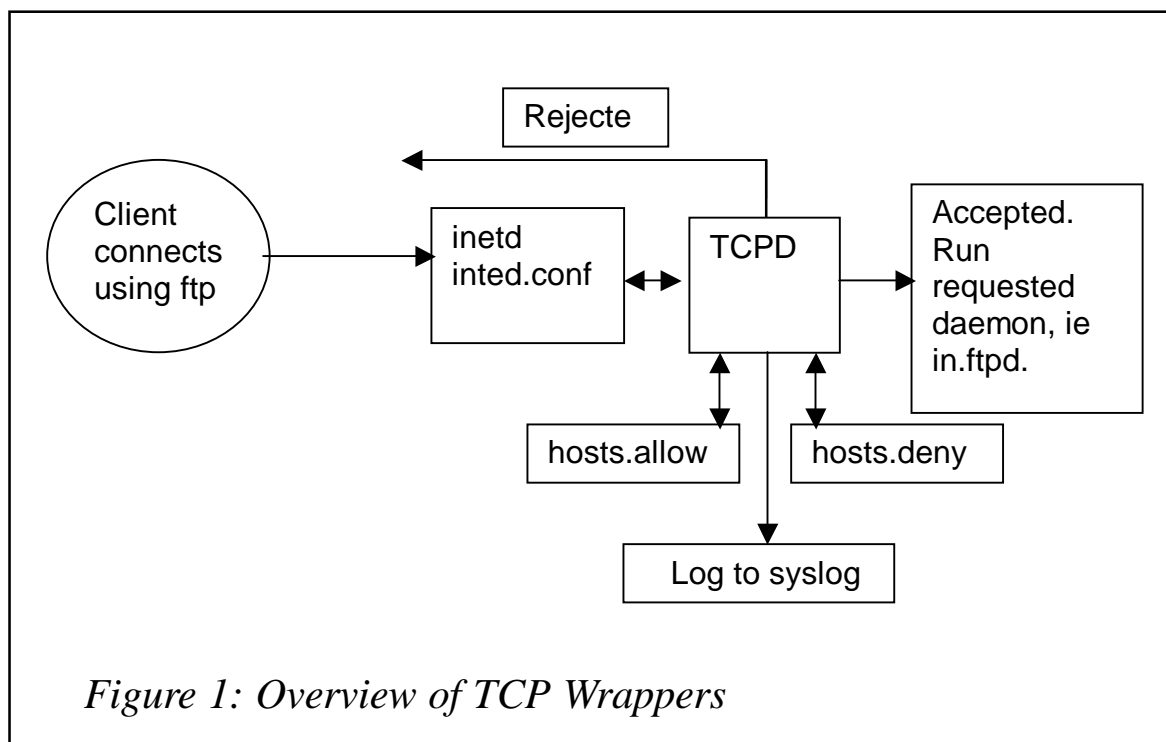
© Xephon plc 2003. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher. Copying permits are available from Xephon in the form of pressure-sensitive labels, for application to individual copies. A pack of 240 labels costs \$36 (£24), giving a cost per copy of 15 cents (10 pence). To order, contact Xephon at any of the addresses above.

*Printed in England.*

## Keep the wraps on those TCP ports

TCP Wrappers wrap themselves around all incoming TCP connections to your system – rather than TCP daemons, which are controlled via `inetd`. Notice I said TCP – this utility does not help you if you want to protect your Web pages that run under HTTP.

When a TCP connection is made on your system, TCP Wrappers (TCPD) is run instead of the required daemon. For instance if a user connects with ftp, the daemon `in.ftpd` is not invoked; rather, TCPD is instead. TCPD will then look at two files – `/etc/hosts.allow` and `/etc/hosts.deny`. These two files, as their names suggest, either allow or deny connections based on rules or patterns. Once TCPD has read these files and found a match, the connection will either be granted or denied. If the connection is allowed, TCPD writes to `syslog` (the system messages file), then hands over control to the real daemon that was called – in our example, `in.ftpd`. TCPD's work is now done, and it will sleep until the next connect is invoked through `xinetd`. If the connection is denied, eg it fails due to the access rules or a pattern match in



the *hosts.allow* or *hosts.deny* file, a message is written to syslog, logging this failure attempt. The connection is broken and TCPD goes back to sleep awaiting the next connection. See Figure 1 for an overview of this process. Some of the most popular TCP daemons are telnet, ftp, shell, rdate, tftp, and talk. The rule here is, if it is TCP and is invoked from inetd, then you can control access to that service from outside connections.

## GETTING INETD TO RECOGNIZE TCP WRAPPERS

You have got to tell the daemon inetd that TCPD is there if you wish to use its services. Generally speaking, you need to change every entry line that has TCP as its protocol so that tcpd is used instead, or its own server program. For ftp, for instance, you may have an entry like this:

```
ftp stream tcp6    nowait root  /usr/sbin/ftpd          ftpd -u 022
```

Change it to:

```
ftp stream tcp6    nowait root  /usr/local/bin//sbin/tcpd ftpd -u 022
```

I have seen problems with some AIX set-ups when trying to do DNS resolving. To fix this, disable the ipv6 services, and use just plain IP, thus:

```
ftp stream tcp     nowait root  /usr/local/bin//sbin/tcpd ftpd -u 022
```

After making changes, use the service command to restart inetd:

```
# refresh -s inetd
```

## THOSE ACCESS FILES

When a connection is initially established TCP Wrappers will first look in */etc/hosts.allow*, then it will check */etc/hosts.deny*, and if there is a pattern match access will be denied or allowed. Confused? Don't be; the general rule of thumb here is to allow access unless otherwise specified. In other words, keep it simple – if the *hosts.allow* and *hosts.deny* file do not exist, TCP Wrappers will deny access to everybody, except connections from the localhost (the actual system where TCP Wrappers is

running). All connections are logged via syslog to */var/adm/messages*.

The general format of the rules or patterns for both files:

```
daemon_list : client_list : [Shell Commands][Banners]
```

where *Shell Commands* are optional, as are *Banners*. We shall look at banners later in the article.

The daemon list is the names of the daemons you wish to allow or deny. The client list is host names, IP addresses, or domain names you wish to allow or deny. To specify multiple daemons or clients, use a comma to separate the entries.

You can also use wildcards to specify daemons or clients. For instance:

- ALL – will match every daemon or every client list.
- LOCAL – will match the local host only, ie any host that does not have a ‘.’ in the name.
- . (that’s a dot) – will match anything. It’s bit like the \* in the bash shell. For example, .boo.com will match any domain that ends in boo.com.

When making changes to the *hosts.deny* or *hosts.allow* file, the changes are dynamic, by which I mean you do not have to restart any daemon or process.

## TYPES OF ACCESS

As usual, most things become clearer with examples, so let’s do that now.

To allow access to all daemons that belong to the domain mycompany.com and to deny access to everybody else:

- */etc/hosts.allow:*  
ALL: . mycompany. com
- */etc/hosts.deny:*

ALL: ALL

Notice in the above example, with *.mycompany.com*, the dot is a wildcard and means 'match all domains that have *mycompany.com* as the end part of their domain name'. In the *hosts.deny* file all (other) daemons and hosts are denied.

When initially learning the rules and patterns, it is best to keep the *hosts.deny* file to ALL:ALL and allow only hosts/daemons specified in the *hosts.allow* file to be allowed access. Remember: keep it simple – it works!

To allow (only) telnet and ftp from everybody:

- */etc/hosts.allow:*  
i n. tel netd , i n. ftpd: ALL
- */etc/hosts.deny:*  
ALL: ALL

Note the use of the comma to separate the two daemons in the client list.

To allow access to telnet only from hosts that have the network address part 192.168.1.:

- */etc/hosts.allow:*  
i n. tel entd: 192. 168. 1.
- */etc/hosts.deny:*  
ALL: ALL

Notice the use of the dot at the end of 192.168.1. This will match all IP (network) addresses that start with that IP number (192.168.1.).

To allow access to all hosts that belong to the domain *mycompany.com* but not those that belong to the *bighacker.com* domain:

- */etc/hosts.allow:*  
ALL: .mycompany. com EXCEPT bi ghacker. com

- */etc/hosts.deny:*

ALL: ALL

In the above example using the EXCEPT does what it says, it allows the client lists on the left of the word EXCEPT, but disallows access to the right of the word EXCEPT.

Similarly, when using TCP Wrappers internally, please do not use EXCEPT with IP numbers on an exposed side of your network because you are open to potential spoofing. You can use EXCEPT to allow all of the 192.168.2 network in, but not the hosts with, say, the following IP addresses: 192.168.2.12, 192.168.2.12, and 192.168.2.22:

- */etc/hosts.allow:*

ALL: 192.168.2. EXCEPT 192.168.2.12, 192.168.2.12, 192.168.2.22

- */etc/hosts.deny:*

ALL: ALL

When a host tries to connect to your AIX machine through a denied daemon, all the connecting host will get on its screen is a, well, a blank screen. It is considered good form to display a refusal message, that way the connecting user will immediately know that they are not allowed to access this particular host. These types of message are called banner messages. You have a banner message for each daemon that you wish to protect or guard. In most cases you will want to display the same message, so it makes sense to copy the same message across to the different banner daemon files that you are creating. Let's do that right now. We will create a denial message for telnet and ftp connections, which are denied access. From the */etc* directory create a new directory structure to hold the banner file(s):

```
$ pwd
/etc
$ mkdir banners
$ cd banners
$ mkdir deny
$ cd deny
```

First, create the banner file for the telnet daemon; insert the following text into the file called `in.telnetd`, in the `/etc/banners/deny` directory:

```
You are not authorized to enter this
machine! Your attempt has been logged.
Access denied to %c
```

Note the `%c` at the end of the text. This will display the calling host's IP address.

Next, we handle the ftp connection; no need to re-type the text, simply copy the file. Staying in the same directory:

```
$ cp in.telnetd in.ftpd
```

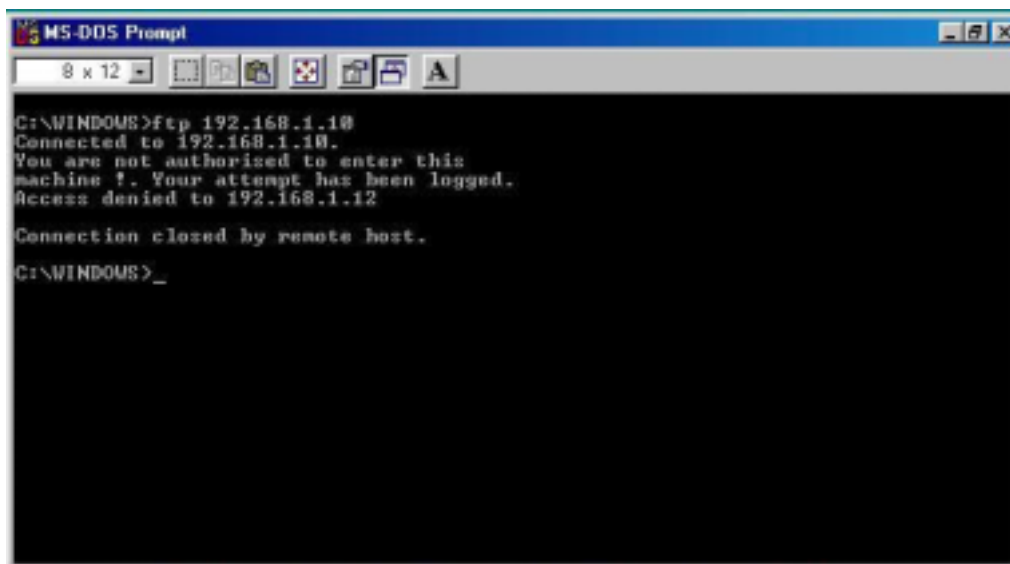
The next task is to tell TCP Wrappers about the banners. Edit the `/etc/hosts.deny` and add the following to the end of the line entry:

```
:banners /etc/banners/deny/
```

So the `hosts.deny` file should look like this:

```
ALL:ALL :banners /etc/banners/deny/
```

Now when a host tries to connect via telnet or ftp and is not allowed access based on your rules in `hosts.deny` or `hosts.allow`,



```
MS-DOS Prompt
8 x 12
C:\WINDOWS>ftp 192.168.1.10
Connected to 192.168.1.10.
You are not authorized to enter this
machine !. Your attempt has been logged.
Access denied to 192.168.1.12

Connection closed by remote host.
C:\WINDOWS>_
```

*Figure 2: Host refusing an access attempt from an ftp client*



it will get a denial message before the connection is closed. Figure 2 shows a Windows 98 client trying to ftp to a host with TCP Wrappers active, and being denied access. The connecting host has an IP address of 192.168.12. My *hosts.allow* file contains the following:

```
ALL: 192.168.1. EXCEPT 192.168.1.12
```

Note that the above example accepts all IP addresses that start with 192.168.1, except a host that has an IP address of 192.168.1.12.

Using the rules in the last example the following message is printed to the */var/log/messages* file courtesy of syslog. You know the IP address of the host trying to connect; though this will probably be the NAT address or the gateway address the user connected to via the Web. If you're running TCP Wrappers on an internal network, then you've got your culprit.

```
Jan 03 20:43:54 bumper inetd[1057]: refused connect from 192.168.1.12
```

Similarly, the following messages are printed to the */var/log/secure* file from the previous example, informing you that access was denied and what service the calling host tried to connect with:

```
Jan 03 20:43:53 bumper inetd[658]: START: ftp pid=1057 from=192.168.1.12
Jan 03 20:43:54 bumper inetd[1057]: FAIL: ftp libwrap from=192.168.1.12
Jan 03 20:43:54 bumper inetd[658]: EXIT: ftp pid=1057 duration=1(sec)
```

## LISTEN IN PLEASE

After putting your rules together, and when you are ready for testing, it is always a good idea to start off allowing everyone access, including all daemons. Then gradually start cutting down on the hosts you want in. Once that is accomplished, start on the daemons. Believe me, you will save yourself a steep learning curve. Hopefully, the basic examples I have demonstrated in this article are enough to get you going, and some will probably do the job for you.

## CONCLUSION

The TCP Wrappers utility is written and maintained by W Venema. This utility has made a huge impact in the general Linux/Unix community on the internal network security front. It allows you quickly and easily to close the doors of your computer to potential trouble. Be sure to check out the man pages of TCPD and hosts\_options for a full description of this utility. As I mentioned at the beginning of the article, Mr TCPD is your doorman, standing guard over your TCP ports.

---

*David Tansley*

*Global Production Support ( IBM-p series) (UK)*

© Xephon 2003

---

## **AIX5L System V Release 4 print subsystem**

Among many areas affected by the introduction of AIX5L is the introduction of System V print tools. Packaged as a part of System V affinity environment, these tools are destined to become a default print solution for AIX. Currently both AIX and System V Release 4 print subsystems are available, with AIX subsystem being a default.

### SYSTEM V PRINT SERVICE OVERVIEW

The System V print subsystem was ported from SCO's UnixWare 7 to AIX 5L.

The print subsystem, as such, supports local printing (parallel and serial), remote printing using BSD's lpd protocol (RFC1179), and network printing using Hewlett-Packard's (HP) JetDirect. The code was internationalized to conform to and comply with AIX international standards and requirements.

The System V print service is a collection of utilities that assists you, as system administrator (or printer administrator), to

configure, monitor, and control the printers on your system.

The print service:

- Receives files that users want to print.
- Filters the files (if needed), so they can print correctly.
- Schedules the work of one or more printers.
- Starts programs that interface with the printers.
- Keeps track of the status of jobs.
- Alerts you to printer problems.
- Keeps track of mounting forms and filters.
- Issues error messages when problems arise.

When a user sends a file to a printer, the print service assigns a unique name, the request ID, to the request (print job). The request ID consists of the name of the printer on which the file is to be printed and a unique number identifying the file. Use this request ID to find out the status of the print job or to cancel the print job. The print service keeps track of all the print requests in an associated request log. The print job is spooled, or queued up, with other print jobs to be sent to a printer. Each print job is processed and waits its turn in the queue to be printed. This queue of pending print jobs is called a print queue.

Each printer has its own queue; you can hold jobs in the queue, move jobs up in a queue, or transfer jobs to another queue. Each print request is sent to a spooling daemon, lpsched, which keeps track of all the jobs. The daemon is created when you start the print service. The spooling daemon is also responsible for keeping track of the status of the printers and slow filters. When a printer finishes printing a job, the daemon starts printing another job if one is queued.

You can customize the print service by adjusting or replacing some of the items, including:

- Printer configuration definitions stored on disk.

- Interface scripts and terminfo database descriptions containing capabilities of printer devices.
- Slow and fast filter definitions enable conversion of data spooled for printing between different print formats.

## PACKAGING AND INSTALLATION

When the tools were first introduced in AIX Version 5.1, they were packaged as members of following filesets:

- bos.msg.en\_US.svprint – System V Print Subsystem
- bos.svprint.fonts – System V Print Fonts
- bos.svprint.hpnp – System V Hewlett-Packard
- bos.svprint.ps – System V Print Postscript
- bos.svprint.rte – System V Print Subsystem
- bos.terminfo.svprint.data – System V Printer Terminal.

In AIX Version 5.2L SMIT, integration of System V print services into SMIT has been introduced and the list of filesets has become:

- bos.msg.en\_US.svprint – System V Print Subsystem
- bos.svprint.dir\_enabled – System V Directory-enabled
- bos.svprint.fonts – System V Print Fonts
- bos.svprint.hpnp – System V Hewlett-Packard
- bos.svprint.ps – System V Print Postscript
- bos.svprint.rte – System V Print Subsystem
- bos.svprint.trans – System V Print Translation
- bos.terminfo.svprint.data – System V Printer Terminal.

The AIX and System V print subsystems are both packaged with the base operating system, but which filesets are installed during

the initial base installation depends on the hardware configuration of your system. The option chosen for the *Installation Configuration (default/minimal)* under the *Advanced Options* menu during the base system installation process does not have any impact on the selection and installation of the print subsystem filesets.

The filesets given below provide the core function of the AIX print subsystem:

- `bos.rte.printers` – front-end printer support
- `printers.rte` – printer back-end
- `printers.msg.xx_XX.rte` – printer back-end messages for the system-specific locale indicated by `xx_XX` in the fileset name.

The front-end printer support, `bos.rte.printers`, is part of the `bos.rte` file package, and therefore is always installed on the system. This fileset provides front-end print commands, such as **qpri**, **lpr**, **enq**, **mkque**, and **rmque**, which allow a user or the system administrator to interact with the `qdaemon`'s spooler queues. For compatibility and usability reasons, the traditional AIX print subsystem maps several System V and BSD print commands to the AIX-specific print commands.

For example, the **lp** command used to be nothing more than a program that translated the System V **lp** flags to their counterparts of the **enq** AIX command, and after all the command line arguments were processed, the translated list, `offlags`, was finally used to call the **enq** command. As far as the front-end is concerned, the System V commands affected are **cancel**, **lp**, and **lpstat**. For BSD, the relevant front-end commands are **lpq**, **lpr**, and **lprm**.

In AIX 5L, the System V and BSD front-end print commands are still in the `/usr/bin` directory, but, by default, they are now linked to the traditional AIX print command wrappers in the `/usr/aix/bin` directory:

```
# ls -l /usr/bin | grep '/usr/aix'
```

```

lrwxrwxrwx 1 root system 19 Feb 15 13:22 cancel ->/usr/ai x/bi n/cancel
lrwxrwxrwx 1 root system 20 Feb 15 13:22 di sabl e->/usr/ai x/bi n/di sabl e
lrwxrwxrwx 1 root system 19 Feb 15 13:22 enabl e->/usr/ai x/bi n/enabl e
lrwxrwxrwx 1 root system 15 Feb 15 13:22 lp->/usr/ai x/bi n/lp
lrwxrwxrwx 1 root system 16 Feb 15 13:22 lpq->/usr/ai x/bi n/lpq
lrwxrwxrwx 1 root system 16 Feb 15 13:22 lpr->/usr/ai x/bi n/lpr
lrwxrwxrwx 1 root system 17 Feb 15 13:22 lprm->/usr/ai x/bi n/lprm
lrwxrwxrwx 1 root system 19 Feb 15 13:22 lpstat->/usr/ai x/bi n/lpstat

```

The AIX printer back-end is a collection of programs called by the spooler's **qdaemon** command to manage a print job that is queued for printing. The printer back-end performs the following functions:

- Receives a list of one or more files to be printed from the **qdaemon** command.
- Uses printer and formatting attribute values from the database; overridden by flags entered on the command line.
- Initializes the printer before printing a file.
- Runs filters as necessary to convert the print data stream to a format supported by the printer.
- Provides filters for simple formatting of ASCII documents.
- Provides support for printing national language characters.
- Passes the filtered print data stream to the printer device driver.
- Generates header and trailer pages.
- Generates multiple copies.
- Reports paper out, intervention required, and printer error conditions.
- Reports problems detected by the filters.
- Cleans up after a print job is cancelled.
- Provides a print environment that a system administrator can customize to address specific printing needs.

As mentioned before, the traditional AIX print subsystem maps several System V and BSD print commands to the AIX-specific print commands. As far as the back-end print support is concerned, the only two System V commands affected are **disable** and **enable**. In AIX 5L, these specific System V back-end print commands are still in the */usr/bin* directory, but by default they are now linked to the traditional AIX print command wrappers in the */usr/aix/bin* directory:

```
# ls -l /usr/bin/enable
lrwxrwxrwx 1 root system 19 Feb 15 13:22 /usr/bin/enable ->/usr/aix/
bin/enable
# ls -l /usr/bin/disable
lrwxrwxrwx 1 root system 20 Feb 15 13:22 /usr/bin/disable ->/usr/aix/
bin/disable
```

In addition to the AIX print command wrappers for System V and BSD print commands in the */usr/aix/bin* directory, a new lock file, *\_AIX\_print\_subsystem*, is installed under the */usr/aix* directory. The existence of the lock file indicates that the AIX print subsystem is active. For reference, a full listing of the */usr/aix* directory is provided in the following:

```
#: ls -lR /usr/aix
total 8

-r-r-r-      1 root      system          0 Feb 15 13:22
_AIX_print_subsystem
drwxr-xr-x   2 bin      bin             512 Oct 21 11:34 bin
/usr/aix/bin:
total 576
-r-xr-xr-x   1 bin      bin            33648 Apr 08 2001  cancel
-r-xr-x-    1 root      printq        33488 Apr 08 2001  disable
-r-xr-x-    1 root      printq        33376 Apr 08 2001  enable
-r-xr-xr-x   1 bin      bin            34228 Apr 08 2001  lp
-r-xr-xr-x   1 bin      bin            33916 Apr 08 2001  lpq
-r-xr-xr-x   1 bin      bin            35236 Apr 08 2001  lpr
-r-xr-xr-x   1 bin      bin            34312 Apr 08 2001  lprm
-r-xr-xr-x   1 bin      bin            35424 Feb 10 2002  lpstat
```

All System V and BSD commands that are mapped by the executables in the */usr/aix/bin* directory to the AIX print subsystem-specific commands have their native System V or BSD counterpart in the */usr/sysv/bin* directory. During a switch from the AIX to the System V print subsystem, the respective duplicate commands

will be handled by removing the inactive print subsystem's command symbolic links and adding new symbolic links for the active commands. The following directory listing reflects this configuration on a system where the initially active AIX print subsystem was deactivated and switched to the System V print subsystem by the use of the newly-introduced **switch.prt** command:

```
# ls -RI /usr/sysv
total 8
drwxr-xr-x  2 bin      bin          512 Dec 02 15:08 bin
/usr/sysv/bin:
total 2504
-x-x-x      1 lp        lp          115186 Jul 28 2002  cancel
-s-x-      1 root      lp          115338 Jul 28 2002  di sable
-s-x-      1 root      lp          115338 Jul 28 2002  enabl e
-x-x-x      1 lp        lp          141650 Jul 28 2002  lp
-r-sr-xr-x  1 lp        lp          172714 Jul 28 2002  lpq
-r-xr-xr-x  1 bin      bin         130374 Jul 28 2002  lpr
-r-xr-xr-x  1 bin      bin         124538 Jul 28 2002  lprm
-x-x-x      1 lp        lp          197898 Jul 28 2002  lpstat
```

Once the System V print subsystem is active, the new lock file, `_SYS5_print_subsystem`, will be present in the `/usr/sysv` directory and the AIX print subsystem lock file `/usr/aix/_AIX_print_subsystem` will no longer exist. You will find the recursive listing for the `/usr/sysv` directory in the following example (note the differences in user and group ownership in comparison with the executables in the `/usr/aix/bin` directory):

```
root@rsc204:/home/root: ls -RI /usr/sysv
total 8
-r-r-r-    1 root      system          0 Feb 15 13:56
_SYS5_print_subsystem
drwxr-xr-x  2 bin      bin          512 Dec 02 15:08 bin
/usr/sysv/bin:
total 2504
-x-x-x      1 lp        lp          115186 Jul 28 2002  cancel
-r-xr-xr-x  1 bin      bin         14400  Feb 04 2002  df
-s-x-      1 root      lp          115338 Jul 28 2002  di sable
-s-x-      1 root      lp          115338 Jul 28 2002  enabl e
-x-x-x      1 lp        lp          141650 Jul 28 2002  lp
-r-sr-xr-x  1 lp        lp          172714 Jul 28 2002  lpq
-r-xr-xr-x  1 bin      bin         130374 Jul 28 2002  lpr
-r-xr-xr-x  1 bin      bin         124538 Jul 28 2002  lprm
-x-x-x      1 lp        lp          197898 Jul 28 2002  lpstat
```



AIX 5L introduces a new user named lp and a related group named the same.

The user lp is added to the */etc/passwd* file for ownership of a majority of the files, which belong to the bos.svprint package. The entry in the */etc/passwd* file is similar to the following example:

```
lp: *: 11: 11: : /var/spool/lp: /bin/false
```

The group lp is added to the */etc/group* file for group ownership of a majority of the files, which belong to the bos.svprint package. The entry in the */etc/group* file is similar to the following example:

```
lp: !: 11: root, lp, printq
```

Furthermore, the lp user is added to the formerly existing printq group. The entry in the */etc/group* file is similar to the following example:

```
printq: !: 9: lp
```

The lp user and a user who belongs to the lp group can administer the System V print subsystem, while root user and a user who belongs to the printq group (the newly added lp user is also a member of the printq group) can administer the AIX print subsystem. The root user can administer both print subsystems, since the root user belongs to both printq and lp groups.

## SYSTEM V PRINT SUBSYSTEM MANAGEMENT

In general, the recommended way to manage the System V subsystem is by using the WSM (Web-based System Manager). This section will describe the available command line utilities, which can be divided into two groups:

- The print service commands available to all users.
- The print service commands available only to the system or print administrator.

The following summarizes the print management commands available to all users:

- **cancel** – allows users to cancel print requests previously sent with the **lp** command. This command permits cancellation of requests based on their request-ID or based on the loginID of their owner.
- **lp** – arranges for the named files and associated information (collectively called a request) to be printed. If file names are not specified on the command line, the standard input is assumed. Alternatively, the **lp** command is used to change the options for a request submitted previously. The print request identified by the request ID is changed according to the print options specified with this command.
- **lpstat** – displays information about the current status of the print service. If no options are given, **lpstat** displays the status of all print requests made by the user.

The administrator can give users the ability to disable and enable a printer so that, when a printer is malfunctioning, the user can turn off the printer without having to call the administrator. (However, in your printing environment, it might not be reasonable to allow regular users to disable a printer.)

To use the administrative commands, you must have root user authority or be a member of either the `printq` or the `lp` group. All of the administrative print service commands Table are located in the `/usr/sbin` directory with two exceptions – the `lpsched` program resides in the `/usr/lib/lp` directory, and the **enable** and **disable** commands are found in *the* `/usr/bin` directory.

The following summarizes the administrative print service commands:

- **accept, reject** – **accept** allows the queueing of print requests for the named destinations. A destination can be either a printer or a class of printers. **Reject** prevents queueing of print requests for the named destinations.
- **enable, disable** – the **enable** command activates the named printers, enabling them to print requests submitted by the **lp** command. If the printer is remote, the command will

only enable the transfer of requests to the remote system. The **disable** command deactivates the named printers, disabling them from printing requests submitted by **lp**.

- **lpadmin** – configures the **lp** print service by defining printers and devices. It is used to add and change printers, to remove printers from service, to set or change the system default destination, to define alerts for printer faults, to mount print wheels, and to define printers for remote printing services.
- **lpfilter** – used to add, change, delete, and list a filter used with the **lp** print service. These filters are used to convert the content type of a file to a content type acceptable to a printer.
- **lpforms** – used to administer the use of preprinted forms, such as company letterhead paper, with the System V print service.
- **lpmove** – moves requests that were queued by **lp** between destinations (printers or classes of printers).
- **lpsched** – allows you to start the System V print service.
- **lpshut** – shuts down the print service. All printers that are printing at the time **lpshut** is invoked will stop printing.
- **lpssystem** – used to define parameters for the LP print service, with respect to communication (using a high-speed network like TCP/IP) with remote systems.
- **lpusers** – used to set limits to the queue priority level that can be assigned to jobs submitted by users of the System V print service.

The administrative print service commands listed above are supplemented by three default printer filters used by interface programs, **lp.cat**, **lp.set**, and **lp.tell**, which are located in the `/usr/lib/lp/bin` directory. The **lp.cat** program reads the file to be printed on its standard input and writes it to the device to be

printed on. Interface programs may call **lp.set** to set the character pitch, line pitch, page width, page length, and character set on the printer. Also, interface programs can use **lp.tell** to forward descriptions of printer faults to the print service. **lp.tell** sends everything that it reads on its standard input to the print service. The print service forwards the message as an alert to the print administrator

Finally, the four BSD compatibility commands (**lpc**, **lpr**, **lpq**, and **lprm**) are available in the */usr/bin* directory for users and administrators.

## USER INTERFACE FOR SYSTEM V PRINT SERVICES

In the current releases of AIX 5L, the Web-based System Manager (WSM) provides the graphical user interface that will be used for the most common functions of the System V print subsystem. For more advanced functions, or to use less common features, users and administrators have to rely on the command line interfaces.

The System V print subsystem management tasks to be performed by the Web-based System Manager application include:

- Adding new printers or classes (parallel, serial, remote, and network).
- Setting the default printer.
- Removing printers or classes of printers.
- Switching to AIX print subsystem.

The status information to be displayed by the Web-based System Manager application includes:

- Showing the default printer.
- Displaying the requests on the default printer.
- Displaying the printers defined on the system.

- Displaying the stopped printers on the system.
- Showing the printers that currently have problems.

Before you can use the Web-based System Manager environment that supports System V printing, you have to switch from the AIX to the System V print subsystem. You can either utilize the **switch.prt -s** SystemV command or use the following sequence of menu selections and operations with the Web-based System Manager tool:

- Select Printers/Overview and Tasks.
- Select the Switch to System V print subsystem task.

After the task has been completed, the Printer container icon is replaced by the Printers (System V) container icon. The Web-based System Manager environment for System V printing is now accessible through the following sequence of menu selections on the Web-based System Manager console:

- Select Printers (System V)/Directory Disabled Overview and Tasks.

If, for example, you would like to define a local print queue named prop24p for your predefined IBM Proprinter 24 P print device / *dev/lp0*, select the new printer task, follow the instructions of the Add New Printer wizard, and complete the task by clicking Finish. Note that the device support for the printer must be installed on the system and that the configuration for lp0 must be completed before you engage in the System V print queue configuration. The printer type can be selected from the pull-down menu next to the field *What is the printer type?* in the Step 3 of 4: Specify Printer Options wizard menu.

If the user-defined printer class ASCII does not already exist, it will be created during the final command execution of the Web-based System Manager wizard.

Also, the final commands executed by the Web-based System Manager Add New Printer wizard allow the newly configured prop24p printer to accept (**accept** command) queueing requests

and enable (**enable** command) the printer to print requests submitted by the **lp** command. The printer will not be defined as the system default print destination. If the user-defined class did not exist before, the wizard creates the class, but will not allow queueing of requests to the class as the print destination.

System administrators who prefer the command line interface to the System V print subsystem can configure the same print queue using the following command sequence.

First define the printer device:

```
# mkdev -c printer -t ibm4212 -s parallel -p ppa0 -w p
lp0 Available
```

Then define the printer queue:

```
# lpadmin -p prop24p -v /dev/lp0 -D "IBM Proprinter 24P" -c ASCII -I
simple -m standard -T proprinter
UX:lpadmin: WARNING: "/dev/lp0" is accessible by others.
    TO FIX: If other users can access it you may get
            unwanted output. If this is not what you
            want change the owner to "lp" and change the mode to 0600.
    Processing continues.
# accept prop24p
UX:accept: INFO: destination "prop24p" now accepting requests
# enable prop24p
```

The new printer can optionally be defined as the system default print destination and the */etc/hosts* file may be submitted as the first test for the System V local print queue:

```
# lpadmin -d prop24p
```

The **lpstat -t** command, entered immediately after the submission of the print request, gives comprehensive status information about the System V print subsystem:

```
# lp /etc/hosts;lpstat -t
request id is prop24p-1 (1 file)
scheduler is running
system default destination: prop24p
members of class ASCII:
    prop24p
device for prop24p: /dev/lp0
ASCII not accepting requests since Sat Feb 15 16:37:18 2003 -
    new destination
```

```
prop24p accepting requests since Sat Feb 15 16:37:38 2003
printer prop24p disabled since Sat Feb 15 16:37:17 2003. available.
    new printer
prop24p-1          root          1421    Sat Feb 15 16:38:31 2003
```

It was previously mentioned that under AIX 5.1 the System V print subsystem management tasks are currently not supported through the SMIT tool. However, some changes and additions have been made to account for the introduction of the System V print subsystem feature. The Print Spooling menu of the SMIT tool was changed to show that most of the menu choices that now exist are valid only for the AIX print subsystem. The AIX print subsystem menu items will still be displayed if the System V print subsystem is active, but they will not work properly, because most of the underlying AIX print subsystem commands and daemons are turned off or disabled in some manner by the `switch.prt.subsystem` script during the switch from the AIX to the System V print subsystem. In addition, one new menu item has been added at the bottom of the Print Spooling menu; it is valid for AIX and System V printing. The name of this item is Change/Show Current Print Subsystem and can be used for either displaying the current running print subsystem or for changing from one to the other.

Under AIX 5.2 System V print Spooling subsystem is fully supported under SMIT.

The Print Spooling menu of SMIT consists of two entries:

- AIX print spooling
- System V print spooling.

Under System V print spooling, the following options are available for selection:

- Manage Print Requests
- List Print Destinations
- Remove Destinations
- Manage Destinations

- Add Local Printer Configuration
- Add Remote Printer Configuration
- Add JetDirect Printer Configuration
- Manage Print Services
- Manage remote systems
- Change/Show Current print Subsystem.

Under the Manage Print requests menu the following options are available for selection:

- List Print Request
- List All Print Requests
- Move Print Request
- Move All Print Requests on Destination
- Cancel Print Request
- Cancel All Print Requests for User on Destination
- Cancel All Print Requests for User on All Destinations
- Cancel All Print Requests on Destination
- Cancel All Print Requests on All Destinations.

Under the Manage Destinations requests menu the following options are available for selection:

- List Print Destinations
- Change/Show Default Destination
- Accept Requests for Destination
- Reject Requests for Destination
- Enable Printer
- Disable Printer
- Change/Show Default Print Priority for All Destinations



- Remove Destination.

## TERMINFO AND SUPPORTED PRINTER TYPES

Since System V printing depends heavily on extracting information from the terminfo database to configure and initialize printers, one file has been added that contains the terminfo definitions for all the printers supported by this subsystem. The name of the file is `svprint.ti`, and it is located in the `/usr/lib/terminfo` directory. The file is compiled and stored in the respective terminfo directories at install time. Currently, on my system about 80 printer types are defined in this file.

Since many printers can be supported by the same terminfo file, the list of printers that are officially supported by System V printing is much larger. In addition, many printer manufacturers support their own printers for System V and send the support out with the printers. This greatly increases the total number.

The list of manufacturers includes, but is not limited to, HP, Canon, Epson, Oki, Panasonic, Unisys, IBM Printer Division, and Lexmark International. In later releases, more printers will be supported and shipped with AIX.

## SWITCHING BETWEEN AIX AND SYSTEM V PRINT SERVICE

AIX provides a command, accessible through SMIT and the Web-based System Manager, that will allow a system administrator to display the current active print subsystem and to switch between the active and inactive one. The command is intended to be executed only by the Web-based System Manager or SMIT, but will work from the command line with the proper permissions. That command, located in `/usr/sbin`, is **switch.prt** **[ -s *print\_subsystem* ] [ -d ]**.

The valid values for the *print\_subsystem* keyword are AIX and SystemV.

Running the command with the **-d** flag will display the current print subsystem; if you do not specify any flags, a brief help

message is displayed on the screen:

```
# swi tch.prt
Usage: [-s AIX | SystemV ] [-d]
-s switches to AIX print system or SystemV print system.
-d displays current subsystem.
```

For security reasons, the **switch.prt** command serves as a front-end to the script `/usr/sbin/switch.prt.subsystem`, which actually does the real work.

## REFERENCES

- 1 *Printing for Fun and Profit under AIX 5L*, SG24-6018, IBM Corporation.
- 2 *AIX 5L Differences Guide Version 5.2 Edition*, SG24-5765-02, IBM Corporation.
- 3 *AIX 5L Version 5.1 Guide to Printers and Printing*, IBM Corporation.

---

*Alex Polak*  
*System Engineer*  
*APS (Israel)*

© Xephon 2003

---

## Illustrated usage of various shell commands and shell features

This article contains a number of examples of shell commands and features, which I hope others will find useful.

### AWK

**awk** can be used to print specific fields, separated by white space(s), from a record:

```
RECORD="Ari f      Zaman"
```

```
echo "${RECORD}" | awk {' print $1' }
echo "${RECORD}" | awk {' print $2' }
echo "${RECORD}" | awk {' print $1' }
echo "${RECORD}" | awk {' print "fname =$1" | name=$2' }
```

**awk** can be used to print specific fields, separated by a field separator, from a record:

```
RECORD="Ari f | Zaman"
echo "${RECORD}" | awk -F|" {' print $1' }
```

**awk** can be used to print and format specific fields, separated by a field separator, from a record:

```
RECORD="Ari f | Zaman"
echo "${RECORD}" | awk -F|" {' print("%-15s %-30s\n", $1, $2)' }
Ari f          Zaman
```

## SED

**sed** can be used to extract a file extension from a file name:

```
ALLOWWD_FILE_EXT=" c pc sql sh"
FILE_NAME="dai ly_acct_update. pc"
FILE_EXT=' echo "${FILE_NAME}" | sed s/.*\.\.//'
```

**sed** can be used to extract a file name without the file extension:

```
FILE_NAME_WITHOUT_EXT=' echo "${FILE_NAME}" | sed s/[^a-z].*//'
```

**sed** can be used to get rid of all trailing spaces from a file:

```
FILE_BEFORE=/u1/zamana/trai ling_spaces. dat
FILE_AFTER=/u1/zamana/no_trai ling_spaces. dat

sed s/ ' *.$// ${FILE_BEFORE} > ${FILE_AFTER}
```

**sed** can be used to delete all empty lines in a file:

```
FILE_BEFORE=/u1/zamana/trai ling_spaces. dat
FILE_AFTER=/u1/zamana/no_trai ling_spaces. dat

sed s/ ' *.$// ${FILE_BEFORE} > ${FILE_AFTER}
```

## SORT AND AWK

**sort** can be used with **awk**:

```
FILE=/u1/zamana/a.dat
```

```
Zamana A |12061957|  
Tamou A |11021954|  
Brown J |21061957|  
Bond J |14061959|  
Khan A |10061960|
```

To generate a list of names ordered by surname in ascending order:

```
cat ${FILE} | awk -F'|' {'print $1'} | sort
```

To generate a list of names ordered by surname in descending order:

```
cat ${FILE} | awk -F'|' {'print $1'} | sort -r
```

## CSPLIT

**csplit** can be used to split a text file from Mark1(BEGIN) and Mark2 (END). For example:

```
FILE_NAME=/u1/zamana/a.dat  
XXXXXXXXXX  
YYYYYYYYYYY  
BEGIN  
RRRRRRRRRR  
DDDDDDDDDD  
TTTTTTTTTTTTTTTTTTTT  
END  
GGGGGGGGGGGGGG  
TTTTTTTTTTTT  
RRRRRRRRRRRR
```

```
csplit ${FILE_NAME} /BEGIN/ /END/+1
```

Or:

```
MARK1="BEGIN"  
MARK2="END"
```

```
csplit ${FILE_NAME} /${MARK1}/ /${MARK2}/+1
```

This will produce the following three files.

xx00:

```
XXXXXXXXXX
```

Yyyyyyyyyyyy

xx01:

```
BEGIN
Rrrrrrrrrrr
Dddddddddd
Tttttttttttttttttttt
END
```

xx02:

```
Gggggggggggg
Ttttttttttt
Rrrrrrrrrrrr
```

Example 2:

```
FILE_NAME=/u1/zamana/a.dat
```

```
BEGIN
Rrrrrrrrrrr
Dddddddddd
Tttttttttttttttttttt
END
Gggggggggggg
Ttttttttttt
Rrrrrrrrrrrr
```

```
csplit -f file ${FILE_NAME} /${MARK1}/ /${MARK2}/+1
```

This will produce the following three files.

file00:

empty

file01:

```
BEGIN
Rrrrrrrrrrr
Dddddddddd
Tttttttttttttttttttt
END
```

File03:

```
Gggggggggggg
Ttttttttttt
Rrrrrrrrrrrr
```

Notes:

- 1 **-f** option can be used to specify the name that will be appended to construct full file names (eg file01, file02, file03).
- 2 **csplit** always produces three files as follows:
  - from the beginning of the file to the first marker.
  - from the first marker to the second marker.
  - from the second marker to the end of file.
- 3 Depending on the location of markers, the first and the third files can be empty.

## SPLIT

**split** can be used to split a text file into a number of pieces, each containing 1000 lines:

```
FILE_NAME="/u1/zamana/name.dat"
FILE_PREFIX="name"
split -l 1000 ${FILE_NAME} ${FILE_PREFIX}
```

Notes:

- 1 If the file contains less than 1000 lines, only one file called nameaa will be created.
- 2 If the file contains more than 1000 lines, each file containing 1000 lines will be created with names like nameaa, nameab, nameac, etc.

**split** can be used to split a text file into a number of pieces, each containing 1024 bytes of data:

```
split -b 1024 ${FILE_NAME} ${FILE_PREFIX}
```

## CKSUM

**cksum** can be used to detect any changes in a text file:

```
FILE_NAME=/u1/zamana/a.dat
CHECKSUM_BEFORE='cksum ${FILE_NAME} | awk {' print $1 }'
```

```
CHECKSUM_AFTER='cksum ${FILE_NAME} | awk {' print $1 }'
```

```
if [ $CHECKSUM_BEFORE -eq $CHECKSUM_AFTER ]
```

```

then
    echo "File not changed"
else
    echo "file has changed "
fi

```

## BASENAME

**basename** can be used to extract a file name from a full file name:

```

FULL_FILE_NAME="/u1/user/zamana/temp/update.txt"
FILE_NAME='basename ${FULL_FILE_NAME}'

```

## DIRNAME

**dirname** can be used to extract a directory name from a full file name:

```

FULL_FILE_NAME="/u1/user/zamana/temp/update.txt"
CUR_DIR_NAME='dirname ${FULL_FILE_NAME}'
PARENT_DIR_NAME='dirname ${CURDIR_NAME}'

```

## EXPR

**expr** can be used to find the length of a string:

```

KEY="ABC001"
KEY_LEN='expr "${KEY}" : '.*''

```

**expr** can be used to validate a numeric field:

```

KEY=1234A
if expr "${KEY}" : '.*[a-zA-Z]' > /dev/null
then
    echo "Key Must be numeric"
else
    echo "Key is ok"
fi

```

**expr** can be used to validate an alpha field:

```

KEY=ABC1
if expr "${KEY}" : '.*[0-9]' > /dev/null
then
    echo "Key Must be character based"
else
    echo "Key is ok"

```

fi

## RM

**rm** can be used to remove files with names that begin with – (dash):

```
FILE=/u1/zamana/a.log
```

```
rm ./${FILE}
```

## KILL

**kill** can be used to kill a background job:

```
kill -<signal_no> %<job_no>
```

```
kill -9 %1
```

## EXEC

**exec** can be used to accept input from a terminal while reading from a file:

```
FILE=/u1/zamana/a.dat
```

For example:

```
exec 3<&0
cat $FILE | while read LINE
do
    exec 4<&0
    exec 0<&3
    echo "Enter the key: \c"
    read KEY
    0<&4
done
```

## Notes:

- 1 At the beginning of the script we save the input file descriptor (0) to 3 (exec 3<&0).
- 2 Start to read from the file. At this stage the input file descriptor (0) is mapped on to the file being read from.



- 3 Save this mapped input file descriptor to 4 (4<&0).
- 4 Restore the original input file descriptor (eg terminal) (0<&3).
- 5 Read from terminal.
- 6 Restore reading from the file (0<&4).

**exec** can be used to find files with .log extension and remove them:

```
find *.log -exec rm -f {} \;
```

## BC

**bc** can be used to add two real numbers with two decimal place precision within a shell script:

```
NUM1=1.234
NUM2=2.345

RESULT='bc <<!'
scale=2
$NUM1 + $NUM2
quit
!
'
```

## STRINGS

**strings** can be used to extract text from an executable file:

```
FILE=/u1/zamana/a.exe
strings ${FILE}
```

## NOHUP

**nohup** can be used to start a job in the background as a terminal detached process:

```
$nohup a.exe &
```

Note: if the terminal is switched off, the job will still be running.

## NICE

**nice** can be used to start a job with a higher priority (increased by 4):

```
ni ce    -4    my_j ob. exe
```

**nice** can be used to start a job with a lower priority (decreased by 4):

```
ni ce    +4    my_j ob. exe
```

Note: the higher the **nice** value the lower the priority, and hence +4 will decrease the priority by 4.

## EVAL

**eval** can be used to define a variable that will reference another variable:

```
De fi ne Va ri ab l es ( )
{
ORA_ERROR="\${ERR_MSG}"
SYS_ERROR="\${ERR_MSG}"
}
Di s pl ay Me s s a ge ( )
{
IN_MSG=' eval  echo $1'
echo "${IN_MSG}"
}
ERR_MSG="Table does not exist "
Di s pl ay Me s s a ge    " ${ORA_ERROR}"
ERR_MSG="Can not execute"
Di s pl ay Me s s a ge    "${SYS_ERROR}"
```

### Notes:

- 1 After the initial assignment, variable ORA\_ERROR and SYS\_ERROR contains the string \${ERR\_MSG}.
- 2 When the variable ERR\_MSG is assigned a value, you must use the **eval** command to reference the value contained in a variable that has been assigned to another variable, as shown in DisplayMessage ().

## STTY

**stty** can be used to accept a password from a terminal without echoing:

```
stty    -echo

echo    "Enter Password: \c"
read   PWD
stty    echo
```

## KSH

**ksh** can be used to find the length of a variable:

```
VAR="My String"
VAR_LEN=${#VAR}
echo  "Variable length is ${VAR_LEN}"
```

## SUBSTRING PROCESSING

To change a file extension:

```
PROG_NAME=my_prog.sh
NEW_PROG_NAME="${PROG_NAME%.sh}.sql"

CUR_FILE_EXT=".sh"
TRANSPOSED_FILE_EXT=".sql"
NEW_PROG_NAME="${PROG_NAME%${CUR_FILE_EXT}}${TRANSPOSED_FILE_EXT}"
```

To extract a file name (ie leaving out the file extension):

```
PROG_NAME=my_prog.sh
PROG_NAME_WITHOUT_EXT="${PROG_NAME%.sh}"
```

To define a variable to have a lower case value:

```
typeset -l    LNAME
echo "Enter last name"
read  LNAME
echo  "${LNAME}"
```

To convert the contents of a variable from upper/mixed to lower case:

```
LNAME=""
echo "Enter last name"
read  LNAME
typeset -l    LNAME
```

```
echo "${LNAME}
```

To define a variable to have an upper case value:

```
typeset -u LNAME
echo "Enter last name"
read LNAME
echo "${LNAME}
```

To convert the contents of a variable from lower/mixed to upper case:

```
LNAME=""
echo "Enter last name"
read LNAME
typeset -u LNAME
echo "${LNAME}
```

To define a variable as an integer:

```
typeset -i DOB
echo "Enter data of birth (ddmmyyy )"
read DOB
echo "${DOB}
```

Note: if the user tries to input a non-numeric character, **ksh** will display an error and exit.

To define a variable as read-only:

```
typeset -r PROG_NAME='basename $0'
echo "${PROG_NAME}
```

Note: the read-only variable must be assigned in the declaration line. Any attempt subsequently to re-assign a value to this variable will result in an error.

To define a variable to hold a left-justified string:

```
typeset -L LNAME
echo "Enter last name"
read LNAME
echo "${LNAME}"
LNAME=" zaman"
typeset -L LNAME
echo "${LNAME}"
```

To set a default value for a variable:

```
VAR1=""
```

```

VAR2="X"
VAR3=${VAR1: -X}
echo ${VAR3}
VAR3=${VAR1: -${VAR2}}
echo ${VAR3}
VAR3=${VAR1: =${VAR2}}
echo ${VAR3}

```

Note: in all cases, if \$VAR1 is null then X will be assigned to \$VAR3

## BUILT-IN SHELL VARIABLE \$LINENO

The variable \$LINENO is set by the shell to an integer number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets LINENO, the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of LINENO is unspecified.

For example:

```

#
# This is an example
#
echo $LINENO      # l i n e n o = 4
#
#
#
func1 ()
{
#
# This is a function
#
sleep 3
echo $LINENO      # l i n e n o = 6
}
#
#
func1
echo "Success"
echo $LINENO      # l i n e n o = 21

```

Note: in order to make this idea useful, we need to modify the

script as follows:

```
#
# This is an example
#
SCRIPT=example.ksh
FUNC_NAME=""
#
echo "${SCRIPT}: ${LINENO}"
#
#
#
func1 ()
{
FUNC_NAME="func1"
#
# This is a function
#
sleep 3
echo "${FUNC_NAME}: ${LINENO}"

}
#
#
func1
echo "Success"
echo "${SCRIPT}: ${LINENO}"
```

at

**at** can be used to execute or run 100 jobs, submitting 10 jobs at a time:

```
AT_QUEUE_DEF="/usr/lib/cron/queuedefs"
QUEUE_NAME="a"
SCRIPT_DIR="/u1/script"
cd $SCRIPT_DIR
for SCRIPT in `ls -1`
do
    at -f $SCRIPT -q $QUEUE_NAME now
done
```

Note: there is an entry in `AT_QUEUE_DEF` file for queue `QUEUE_NAME` as follows:

a. 10j 160

This entry in the file specifies that the queue, a, for **at** jobs, can

have up to 10 jobs running simultaneously; those jobs will be run with a **nice** value of 1. If a job cannot be run, wait for 60 seconds before trying to reschedule it

In the script we submit all the jobs to **at** but **at** controls the queue according to the queue definition found for `#{QUEUE_NAME}` in `#{AT_QUEUE_DEF}` file.

To vary the number of simultaneous jobs to be run, edit the queue definition in the file to change the number before the letter j.

---

*Arif Zaman*  
*ETL Developer (UK)*

© Xephon 2003

---

## Terminal capabilities

### TERMINFO FILES

Each and every terminal type intended to be attached to the system must be defined within a file contained in the `/usr/share/lib/terminfo` directory. This directory is a database that describes terminals and defines their capabilities and methods of operation. These definitions are used by various programs, such as the **vi** editor.

The information defined in **terminfo** includes initialization sequences, padding requirements, cursor positioning parameters, and other command sequences that control various terminal attributes such as function key definitions.

The **terminfo** database consists of a series of ASCII files containing the various capabilities for all terminal types from a particular manufacturer. For example, the **ibm.ti** file contains information on all IBM terminals. Similarly, the **wyse.ti** file contains information on all Wyse terminals.

It is possible to edit and modify these source files, but, before a

**terminfo** source file can be used, it must be compiled using the **tic** command. For example:

```
tic wyse.ti
```

will search for the **wyse.ti** file in the current directory and produce compiled files for each terminal defined in **wyse.ti**. These will be stored in the directory */usr/share/lib/terminfo/w* with a separate file for each terminal type, such as **wyse60**, **wyse120** etc.

A program that wants to make use of the terminal capability database selects an entry according to the value of the **TERM** environment variable. It is also possible to set the **TERMINFO** variable with an absolute pathname to point to a terminal description that is not in the standard **terminfo** directory.

Terminal capabilities are of three types:

- Boolean – indicate that the terminal has a particular feature. Boolean capabilities are true if the corresponding name is included in the terminal description.

Examples of this type are **am** for automatic margins, and **ul** for overstrike with underline character. If any Boolean capability is not specified, programs will assume that the terminal does not have that feature.

- Numeric – indicate the size or value of a particular terminal characteristic.

Numeric capabilities are followed by the **#** character and then the value. Thus **cols#80** indicates that the terminal has 80 columns, and **lines#24** that it has 24 lines.

- String – indicate an escape sequence that can be used to perform particular terminal operations. They define how a command is issued to the terminal.

The format of a string capability is the name followed by an equals sign, followed by a command sequence. For example, **cuu1=^K** specifies that the **Ctrl K** sequence will move the cursor up one line.



These terminal-dependent capabilities in **terminfo** are made available for use by the shell with the **tput** command. The output of **tput** is a string if the attribute is of type *string*, or an integer if the attribute is of type *integer*. If the attribute is of type *Boolean*, the command simply sets the exit value to 0 for true, or 1 for false, and produces no other output.

As an example, for most terminal types you can echo the **clear** screen sequence for the current workstation or window with:

```
tput clear
```

The same result can actually be achieved with the **clear** command on its own, although this is not supported for vt100 emulation. Similarly, to display the number of columns, enter:

```
tput cols
```

You can also display your current terminal type using:

```
tput longname
```

## EXAMPLE OF TEXT DISPLAYING CAPABILITIES

Consider the following example, which serves to show how text displayed on the screen can be highlighted, underlined etc:

```
$ vi texttest
```

```
bold=$(tput smso)           # start stand out mode
endbold=$(tput rmso)        # end stand out mode
under=$(tput smul)          # start underline mode
endunder=$(tput rmul)       # end underline mode
blink=$(tput blink)         # start blink mode
endall=$(tput sgr0)         # disable all attributes
```

```
print "${bold}Standout mode$endbold"
print "${under}Underline mode$endunder"
print "${blink}Blink mode"
print "${under}Blink and underline mode$endall"
```

The first group of commands defines a number of variables that are used to set and unset terminal attributes. The second group uses these variables, along with the **print** command, to display text in various modes. You should be aware that the **tput** command may produce unexpected results on some terminals,

or no changes at all. For example, **tput blink** may just produce a brighter version of stand out mode rather than blink the text.

Once an attribute has been enabled it remains in effect until specifically disabled, either with the corresponding command to disable it, or by using the **disable all** command:

```
tput sgr0
```

When you log out of your current shell, all attributes will be automatically reset to their default values.

If you intend to use these capabilities on a regular basis, it is probably best to define **bold**, **endbold**, etc in your *.profile* file, and then export them so that they are available in your environment; alternatively you can use your *.kshrc* file.

## MENU EXAMPLE

Consider the following simple program, which uses a menu:

```
$ vi menutest
#!/bin/ksh
# this function displays the menu
f_displaymenu()
{
    clear
    print "\n\n\n\t\t\tSample Text Attributes"
    print "\n\t\t\t1. Display bold text"
    print "\n\t\t\t2. Display underlined text"
    print "\n\t\t\t0. Exit from program"
    print "\n\t\t\tEnter choice: \c"
}
# this function displays bold text
f_displaybold()
{
    clear
    print "\n\n\n\n\t\t\t$(tput smso)This is an example \
of bold text$(tput rmso)"
    print "\n\t\t\tPress ENTER to return to menu"
}
# this function displays underlined text
f_displaypunder()
{
    clear
    print "\n\n\n\n\t\t\t$(tput smul)This is an example \
of underlined text$(tput rmul)"
}
```

```

        print "\n\t\t\t\t\tPress ENTER to return to menu"
    }
    # this function displays an invalid choice message
    f_invalid()
    {
        print "\n\n\t\t\t\t\t$(tput smso)Invalid Choice$(tput sgr0)\c"
        sleep 2
    }

    # this is the main program loop
    while true
    do
        f_dispmenu
        read answer
        case $answer in
            1)
                f_displaybold
                read answer ;;
            2)
                f_displayunder
                read answer ;;
            0)
                clear
                exit ;;
            *)
                f_invalid ;;
        esac
    done

```

This menu program uses functions to display the menu sections. When the menu is displayed, the system waits for the user to make a selection and then runs the appropriate function.

The script requires a **sleep** statement in the **f\_invalid** function to allow the user time to read the error message before the script automatically re-displays the menu.

The script in itself is not particularly useful but merely serves to show how a simple menu script can be constructed.

## USING TERMINAL RAW MODE

Normally when you enter characters at a command prompt, the operating system buffers them into lines, performs editing in the buffer to erase characters that have been backspaced over, deletes lines that have been killed, processes end-of-file

characters, and only executes the line when it receives an end-of-line or carriage return (ie line termination character).

Under most operating conditions, your terminal will be set up so that it processes only a line of input characters when it receives a carriage return. A terminal can be set to raw mode, however, so that it reads only one character at a time. In this mode, when a single character is typed, it is returned immediately, and the read is terminated by the input of the single character instead of a line followed by a carriage return.

When in raw mode, no checking is made for the special characters that handle *erase*, *kill*, and *end-of-file*, and thus all characters are valid for input. It is not therefore possible to trap signals such as **Ctrl D** and **Ctrl C** in a program using raw mode.

A terminal can be set to the raw mode with the command:

```
stty raw -echo
```

where the **-echo** option prevents the character you enter from being echoed back to the screen.

We can make use of this capability in the **menutest** example by using the above command in conjunction with the **dd** command, so that the program needs to accept only a single character for input.

The **menutest** program can now be amended, with amendments and additions shown in italics. Remember that the new version expects a single character, so should you make an error in the script you may not be able to cancel the program with **Ctrl C**, nor suspend it with **Ctrl Z**. Ideally you should be working in a Windows environment so that you can start a second session in order to kill the script in the event of an error.

```
#!/bin/ksh
# this function gets a single keystroke
f_get_key()
{
    stty raw -echo
    KEY=$(dd count=1 2>&-)
    stty -raw echo
}
```

```

# this function displays the menu
f_dispmenu()
{
    clear
    tput cnorm
    print "\n\n\n\t\t\tSample Text Attributes"
    print "\n\t\t\t1. Display bold text"
    print "\n\t\t\t2. Display underlined text"
    print "\n\t\t\t0. Exit from program"
    print "\n\t\t\tEnter choice: \c"
}
# this function displays bold text
f_dispbold()
{
    clear
    tput civis
    print "\n\n\n\n\t\t\t$(tput smso)This is an example \
of bold text$(tput rmso)"
    print "\n\t\t\tPress ENTER to return to menu"
}
# this function displays underlined text
f_dispunder()
{
    clear
    tput civis
    print "\n\n\n\n\t\t\t$(tput smul)This is an example \
of underlined text$(tput rmul)"
    print "\n\t\t\tPress ENTER to return to menu"
}
# this function displays an invalid choice message
f_invalid()
{
    tput civis
    print "\n\n\n\t\t\t$(tput smso)Invalid Choice$(tput sgr0)\c"
    sleep 2
}
# this is the main program loop
while true
do
    dispmenu
    f_get_key
    case $KEY in
    1)
        f_dispbold
        f_get_key;;
    2)
        f_dispunder
        f_get_key;;
    0)
        clear

```

```

        tput cnorm
        exit ;;
*)
        f_inval id;
esac
done

```

In the **f\_get\_key** function, the second **stty** command resets the terminal for normal operation so that the program will continue to execute and display menus and messages in the correct way. The strange looking line:

```
KEY=$(dd count=1 2>&-)
```

waits for a single keystroke, specified by the **count=1** option to **dd**. It tells **dd** to receive just a single character at a time and allocate it to the variable **KEY**. The value of **count** can be changed so that **count=3**, for example, will accept three characters, possibly generated by a function key.

If you do not turn off standard error with **2>&-**, then every time you execute the **dd** command you will receive the messages:

```

0+1 records in
0+1 records out

```

The main program loop has been changed so that each **read answer** has been replaced with **f\_get\_key**, and the **case \$answer in** with **case \$KEY in**.

Another enhancement has been made to the program to give it a slightly more professional, or cosmetic, look and feel. Previously, when a menu selection was made, the text would be displayed and the cursor would appear at the start of the line after the last line of text, waiting for the next key entry. Because we have designed the program to accept single key strokes without waiting for a carriage return we do not need to have the cursor displayed on the screen after the bold or underlined text has been displayed.

A cursor can be made invisible by using the **tput civis** command, and this has been used in the **f\_dispbold** and **f\_dispunder** functions and when displaying the **Invalid Choice** message.

The cursor needs to be visible, ie back to normal, when the main menu is displayed and when the program exits. This is made possible by using the **tput cnorm** commands at the start of the **f\_dispmenu** function, and when the **exit** choice has been selected from the main menu.

## THE SELECT COMMAND

While on the subject of menus, let us consider another built-in shell construct, **select**, which can be used to generate simple menus. You should be aware that this command is available only in the Korn shell. The syntax of the command is similar to that of the **for** loop:

```
select variable in word_list
do
    loop_body
done
```

Like **for**, if you omit the **in word\_list**, the command will default to **\$@**. Unlike **for**, however, the commands in **loop\_body** are not executed the number of times that **variable** occurs in **word\_list**. What **select** does instead is to generate a simply formatted menu containing each of the words in **word\_list** preceded by a number.

**select** will then prompt you for a number, using the **\$PS3** prompt, which by default is set to **#?**. Since this is not a particularly meaningful prompt, most scripts using **select** change this to something the user can understand.

When you enter your menu choice, the item on the menu is available in **variable** and the selected number is stored in the built-in variable **REPLY**; both variables are normally referenced inside the body of the loop. **select** does not check that you entered a valid number, and so you must have appropriate checks within the body of the construct. You must also have some means of breaking out of the loop since this feature is not part of the **select** statement.

So that you can understand this construct, let us consider the

following example. Suppose that you have a system containing a number of applications, and that your users need differing terminal types to run their applications. Or perhaps they log in using some Windows/X Windows terminal emulation, or from ASCII terminals. Each terminal type that can be used potentially creates a number of problems for you as a system administrator, and to make life easier for yourself you create the following script, which we have named **setterm**.

```
$ vi setterm
#!/bin/ksh
trap '' INT TSTP
PS3="Select terminal type: "
TERMS="vt100 \
vt220 \
ibm3151 \
aixterm \
dtterm \
xterm"
set -- $TERMS
select term in $TERMS
do
    if [[ $REPLY > $# ]] || [[ $REPLY < 1 ]] || [[ -z $term ]]
    then
        print Invalid choice
        continue
    fi
    TERM=$term
    print TERM has been set to $TERM
    break
done
```

When the script is executed, a user can select the correct terminal type from the menu, which will look like the following:

```
1) vt100
2) vt220
3) ibm3151
4) aixterm
5) dtterm
6) xterm
Select terminal type:
```

For users with little Unix knowledge, the code in this script should be included in their **.profile** files so that the terminal type can be automatically selected when the user logs in. We have included



a **trap** statement to ensure that the script cannot be interrupted with **Ctrl C**, or suspended with **Ctrl Z** (the **TSTP** signal), so that users are forced to select a terminal type. Other users may be required to run **setterm** themselves after logging in so that the **trap** statement may not be necessary since we expect their Unix knowledge to be sufficient to know what they are doing.

We have modified the **PS3** prompt to something a little more meaningful, and we have placed the terminal types on separate lines for readability. You must ensure that there are spaces before each backslash.

In any script such as this, we would normally expect to find a **case** statement to check that whatever we type in is a valid choice. This would be reasonable if there were a small number of choices, but should the list of menu items grow very large, then we would want to avoid as much typing as possible. If we also wanted to add items to the menu list at a later date, we would have to add stanzas to our **case** statement. Being inherently lazy, we want to avoid all this typing, and to overcome this we have used the construction as shown, which we will now explain.

We want to be able to check that the choice of menu item we enter is non-null, numeric, and within the range of menu items displayed. We know that the first item on our menu list will always start at 1, but we can never be sure what number the last item is going to be. As you will know, the **set --** command can be used to split the contents of a variable to **\$1**, **\$2**, etc, and by using this command with **\$TERMS**, not only will it do this, but it will also reset the **\$#** variable, ie the number of arguments, to show us how many items **TERMS** contains.

Now that we know our range is from **1** to **\$#**, we check the validity of our selection using the test:

```
[[ $REPLY > $# ]] || [[ $REPLY < 1 ]]
```

As you can see, this is not a normal construction to check numeric choices, where we would use **-gt** and **-lt**. Instead we are using string comparisons. The reason for this is that should a non-numeric character be entered, then, from a string point of

view, it could not possibly fall within our numeric range, would fail the test, and the script would print the error message. If we had only the standard numeric tests then we would require further checks to see if our answer was non-numeric.

The remaining part of our test, `|| [[ -z $term ]]`, is not there to check, as you might expect, when someone just hits the *Enter* key, since this particular check is actually made by **select** itself. If the menu items range from 1 to 6, say, and someone enters 10, then in a string comparison 10 is *less than* 6 and *greater than* 1 and so an error message would not be produced from this comparison alone. However, there is no menu item corresponding to the value 10 and so **term** would be set to null, which means that it would fail the final test and the error message would be produced.

When a user enters a number within the range, the script exits with the **break** statement.

---

*Tonto Kowalski*  
*Guru (UAE)*

© Xephon 2003

---

# AIX news

---

ORSYP has announced availability of Dollar Universe Publisher and Dollar Universe Reporter, new associated modules for its job scheduling software suite, Dollar Universe. The new modules enable enterprises to handle IT operations more efficiently, through complete knowledge management of operations and accurate reporting.

ORSYP's Dollar Universe product suite offers a cross-platform solution for automating IT functions across heterogeneous IT environments, in particular those relying on being able to batch process large amounts of data within an allotted timeframe. The addition of the new modules to the product suite simplifies change management and gives them tighter control over mission-critical operations.

Dollar Universe runs on AIX, Linux, MVS, and a host of other platforms.

For further information contact:  
ORSYP, Castle Court, 41 London Road,  
Reigate, Surrey RH2 9RJ, UK.  
Tel: (0173) 7735 021.  
URL: [www.orsyp.co.uk](http://www.orsyp.co.uk).

\* \* \*

Veritas has announced a new version of its Global Data Manager, which now manages and monitors back-up and recovery processes for both Veritas NetBackup and Backup Exec software in distributed environments.

It now provides a single point of management for all back-up and recovery processes

running on different operating systems and hardware platforms. For NetBackup and Backup Exec, it makes possible centralized management of both data centre and remote back-up and recovery from a common dashboard and provides consolidated real-time visibility of all back-up operations.

Administrators are given a dashboard view of multiple data protection processes spread across a global enterprise and are given the means to control back-up processes and policies from a single point.

For further information contact:  
VERITAS, 251 New Karner Road, Suite  
401, Albany, NY 12205, USA.  
Tel: (518) 690 0019.  
URL: <http://www.veritas.com>.

\* \* \*

Mainsoft is to broaden the availability of its Visual MainWin 5 product by adding seven new Unix operating systems to its list of supported platforms.

Specifically, it will add support for Solaris 9, 2.6, and 2.7, HP-UX 11.22, AIX 5.2 and 4.3.3, and Linux 8.0 to existing support for Solaris 8, HP-UX 11.00 and 11.11, AIX 5.1, and Linux 7.3.

For further information contact:  
Mainsoft, 3850 North First Street, San Jose,  
CA 95134, USA.  
Tel: (408) 544 1400.  
URL: <http://www.mainsoft.com/products/mainwin.html>.

\* \* \*



**xephon**