

Shell Operations

Utilities

This section introduces the following utilities, listed in alphabetical order:

chsh

kill

ps

echo

nohup

sleep

Shell Commands

This section introduces the following shell commands, listed in alphabetical order:

echo

exit

tee

eval


kill

umask

exec

shift

wait




| Shell | Linux | UNIX |
|--------------|--------------------|-------------|
| Bash | sh or bash | bash |
| Korn shell | ksh, pdksh, or zsh | ksh |
| C shell | csh or tesh | csh |

Figure 5–1 Comparison of Linux and UNIX shell names.

| Shell | Full pathname |
|--------------|-------------------------|
| Bash | /bin/bash (or /bin/sh) |
| Korn | /bin/ksh |
| C | /bin/tesh (or /bin/csh) |

Figure 5–3 Common shell locations.



Shell Command: **echo** {arg}*


echo is a built-in shell command that displays all of its arguments to standard output. By default, it appends a newline to the output.

Figure 5–4 Description of the *echo* shell command.

Shell Metacharacters


| Symbol | Meaning |
|--------|---|
| > | Output redirection; writes standard output to a file. |
| >> | Output redirection; appends standard output to a file. |
| < | Input redirection; reads standard input from a file. |
| * | File substitution wildcard; matches zero or more characters. |
| ? | File substitution wildcard; matches any single character. |
| [...] | File substitution wildcard; matches any character between brackets. |

Figure 5–5 Shell metacharacters. (Part 1 of 2)



| | |
|---------------------------------|---|
| <code>`<i>command</i>`</code> | Command substitution; replaced by the output from <i>command</i> . |
| <code> </code> | Pipe symbol; sends the output of one process to the input of another. |
| <code>;</code> | Used to sequence commands. |
| <code> </code> | Conditional execution; executes a command if the previous one failed. |
| <code>&&</code> | Conditional execution; executes a command if the previous one succeeded. |
| <code>(...)</code> | Groups commands. |
| <code>&</code> | Runs a command in the background. |
| <code>#</code> | All characters that follow up to a newline are ignored by the shell and programs (i.e., a comment). |
| <code>\$</code> | Expands the value of a variable. |
| <code>\</code> | Prevents special interpretation of the next character. |
| <code><<<i>tok</i></code> | Input redirection; reads standard input from script up to <i>tok</i> . |

Figure 5-5 Shell metacharacters. (Part 2 of 2)



| Wildcard | Meaning |
|-----------------|--|
| * | Matches any string, including the empty string. |
| ? | Matches any single character. |
| [..] | Matches any one of the characters between the brackets. A range of characters may be specified by separating a pair of characters by a dash. |

Figure 5–6 Shell wildcards.



Figure 5-7 A simple pipeline.

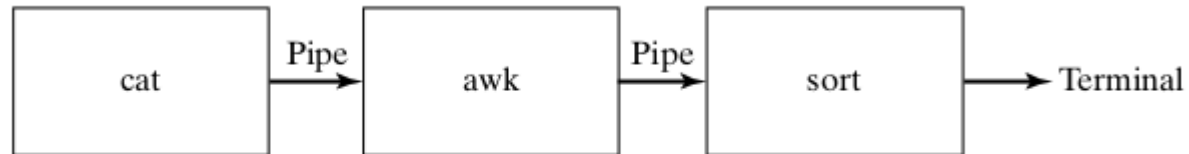


Figure 5-8 A pipeline that sorts.

Utility: **tee** -ia {fileName}+

The **tee** utility copies its standard input to the specified files and to its standard output. The **-a** option causes the input to be appended to the files rather than overwriting them. The **-i** option causes interrupts to be ignored.

Figure 5-9 Description of the **tee** command.

Shell Variables

| Name | Meaning |
|---------|---|
| \$HOME | The full pathname of your home directory. |
| \$PATH | A list of directories to search for commands. |
| \$MAIL | The full pathname of your mailbox. |
| \$USER | Your username. |
| \$SHELL | The full pathname of your login shell. |
| \$TERM | The type of your terminal. |

Figure 5–11 Predefined shell variables.

| Name | Meaning |
|----------|---|
| \$\$ | The process ID of the shell. |
| \$0 | The name of the shell script (if applicable). |
| \$1..\$9 | $\$n$ refers to the n th command-line argument (if applicable). |
| \$* | A list of all the command-line arguments. |

Figure 5–12 Special built-in shell variables.

Utility: **ps** -ef

ps generates a listing of process status information. By default, the output is limited to processes created by your current shell. The **-e** option instructs **ps** to include all running processes. The **-f** option causes **ps** to generate a full listing. The **-l** option generates a long listing. The meaning of each **ps** column is described in the text that follows.

Figure 5–13 Description of the **ps** command.

```
$ ps -ef          ...list all users' processes.
UID              PID  PPID  C STIME TTY          TIME CMD
root             1    0    0 Aug25 ?           00:00:02 init [5]
root             2    1    0 Aug25 ?           00:00:00 [ksoftirqd/0]
root             3    1    0 Aug25 ?           00:00:00 [events/0]
root             4    1    0 Aug25 ?           00:00:01 [kblockd/0]
root             5    1    0 Aug25 ?           00:00:00 [kapmd]
root             7    1    0 Aug25 ?           00:00:00 [pdflush]
root             8    1    0 Aug25 ?           00:00:05 [kswapd0]
root            742    1    0 Aug25 ?           00:00:00 syslogd -m 0
root            750    1    0 Aug25 ?           00:00:00 klogd -2
daemon          965    1    0 Aug25 ?           00:00:00 /usr/sbin/atd
root           1003   948    0 Aug25 ?           00:00:00 -:0
root           1078    1    0 Aug25 ?           00:00:00 crond
glass          1362  1282    0 Aug25 ?           00:03:01 magicdev
glass          1376    1    0 Aug25 ?           00:00:00 /usr/lib/gconfd
glass          1463  1282    0 Aug25 ?           00:00:00 kwrapper
glass          23708 23693    0 15:51 pts0       00:00:00 bash
root           24379 23708    0 16:41 pts0       00:00:00 su
root           24382 24379    0 16:41 pts0       00:00:00 bash
root           24805 24382    0 17:28 pts0       00:00:00 ps -ef
$ _
```

Utility/Shell Command: **kill** [*-signalId*] {*pid* }+

kill -l

kill sends the signal with code *signalId* to the list of numbered processes. *signalId* may be the number or name of a signal. By default, **kill** sends a TERM signal (number 15), which causes the receiving processes to terminate. To obtain a list of the legal signal names, use the **-l** option. To send a signal to a process, you must either own it or be a super-user. For more information about signals, refer to Chapter 12, “Systems Programming.”

Processes may protect themselves from all signals except the KILL signal (number 9). Therefore, to ensure a kill, send signal number 9 (note that sending a KILL will not allow a process to clean up and terminate normally, as many programs do when they receive a TERM signal).

Figure 5–18 Description of the **kill** command.