

System Administration Toolkit: Process administration tricks

Skill Level: Introductory

[Martin Brown \(mc@mcslp.com\)](mailto:mc@mcslp.com)
Freelance Writer
Consultant

21 Feb 2006

Discover how to get the information you want on UNIX® processes. Knowing what is running on your UNIX system is one of the most basic requirements of any system administrator. The standard process list is useful, but often the information that it provides is not in the right format or doesn't contain exactly the processes or information you need. Being able to select specific types of data from the process list makes administration a lot easier. In this article, you'll examine how to extend that process further to improve the readability of the information, or provide summaries and information that are not easily obtainable elsewhere. You'll also look at methods for standardizing how to obtain process information across different UNIX platforms.

About this series

The typical UNIX® administrator has a key range of utilities, tricks, and systems he or she uses regularly to aid in the process of administration. There are key utilities, command-line chains, and scripts that are used to simplify different processes. Some of these tools come with the operating system, but a majority of the tricks come through years of experience and a desire to ease the system administrator's life. The focus of this series is on getting the most from the available tools across a range of different UNIX environments, including methods of simplifying administration in a heterogeneous environment.

Using ps

The `ps` command line tool lists running processes. The tool exists on all variants of

UNIX and, on the whole, works in the same basic fashion -- asking the kernel for a list of running processes and then reporting on the list of processes and their properties, such as memory usage, running time, and other details.

The `ps` tool is actually a very powerful utility, although many administrators probably rely on only one or two of the available options and, in their eyes, to drill down to the information they want. You can get more information out of the command by making use of the built-in command line options, and even more if you combine `ps` with commands through pipes to get access to exactly the information you want.

Listing all processes

The standard output from `ps` only lists your running processes, even when you are logged in as root. Depending on whether your UNIX environment is based on the BSD or AT&T, SysV UNIX base alters the primary command line options that you use to get information on other processes on the system, or changes the information that is displayed. Within a BSD-based UNIX environment, the output incorporates the process ID, terminal, status, time (duration of execution in CPU seconds used, rather than an indication of when the process was started), and the command executed, as shown in [Listing 1](#).

Listing 1. Listing process on BSD UNIX variants

```
$ ps
  PID  TT  STAT      TIME COMMAND
  391  p5  S        0:00.24 /bin/bash
 9165  p5  S+       0:00.50 emacs
  476  p6  S        0:01.03 /bin/bash
 9299  p6  S        0:00.09 xterm
 9319  p6  S        0:00.07 xterm
 9423  p6  S        0:00.12 ftp atuin
 9513  p6  R+       0:00.01 ps
 9301  p7  Ss+     0:00.01 /usr/X11R6/bin/luit
 9302  p8  Ss+     0:00.03 bash
 9321  p9  Ss+     0:00.01 /usr/X11R6/bin/luit
 9322  pa  Ss+     0:00.02 bash
```

Under an SVR4 environment, fewer columns are provided (you no longer get a process status), as seen in [Listing 2](#) below.

Listing 2. Listing process on BSD UNIX variants

```
$ ps
  PID  TTY      TIME  CMD
19915 pts/3    00:00:00 bash
29145 pts/3    00:00:00 emacs
32256 pts/3    00:00:00 emacs
26986 pts/3    00:00:00 xterm
31303 pts/3    00:00:00 ftp
31358 pts/3    00:00:00 ps
```

To get a list of all the processes running on the system, you need to use different command line options, according to the UNIX variant in use. Within a BSD UNIX, the command line option `-a` lists all user processes, including your own. However, the list will still not include processes without a controlling terminal (for example, those started at boot time, daemons, and those executing as part of a `cron` job). To get a list of all running processes, you must use the `-A` command line option (see [Listing 3](#)).

Listing 3. Listing all processes on BSD systems

```
$ ps -A
  PID  TT  STAT      TIME COMMAND
    1  ??  S<s      0:15.47 /sbin/launchd
   23  ??  Ss       0:00.02 /sbin/dynamic_pager -F /private/var/vm/swapfile
   27  ??  Ss       0:00.95 kexstd
   49  ??  Ss       0:05.17 /usr/sbin/configd
   50  ??  Ss       0:01.89 /usr/sbin/coreaudiod
   51  ??  Ss       0:04.40 /usr/sbin/diskarbitrationd
   52  ??  Ss       0:00.08 /usr/sbin/memberd -x
   53  ??  Ss       0:02.80 /usr/sbin/securityd
   55  ??  Ss      11:03.59 /usr/sbin/notifyd
   57  ??  Ss       0:01.13 /usr/sbin/DirectoryService
...
 8051  p2  S+       0:00.61 ssh root@bear
  292  p3  Ss       0:00.02 bash
  372  p3  S+       0:00.42 ssh admin@atuin
  312  p4  Ss+      0:00.03 bash
  332  p5  Ss       0:00.03 bash
  391  p5  S        0:00.24 /bin/bash
 9165  p5  S+       0:00.50 emacs
  352  p6  Ss       0:00.04 bash
  476  p6  S        0:01.04 /bin/bash
 9299  p6  S        0:00.09 xterm
 9319  p6  S        0:00.07 xterm
 9423  p6  S        0:00.14 ftp atuin
 9520  p6  R+       0:00.01 ps -A
 9301  p7  Ss+      0:00.01 /usr/X11R6/bin/luit
 9302  p8  Ss+      0:00.03 bash
 9321  p9  Ss+      0:00.01 /usr/X11R6/bin/luit
 9322  pa  Ss+      0:00.02 bash
```

The `-A` command line option is equivalent to using both `-a` and `-x` options, where `-a` shows processes with controlling terminals and `-x` shows processes without controlling terminals.

Under SVR4 variants, it is the `-e` command line option that shows you all running processes, irrespective of whether or not they have a controlling terminal. It is equivalent, in terms of the processes displayed, to the BSD `-A` option. You can see a sample of the output in [Listing 4](#).

Listing 4. Process listings in SVR4 environments

```
$ ps -e
  PID TTY          TIME CMD
    0 ?           15:24 sched
    1 ?           0:00  init
```

```

 2 ?          0:00 pageout
 3 ?          0:00 fsflush
308 ?          0:00 devfsadm
 7 ?          0:06 svc.star
 9 ?          0:10 svc.conf
506 ?          0:00 htt_serv
260 ?          0:00 rpcbind
259 ?          0:00 cron
 52 ?         0:00 dhcpagen
282 console    0:00 ttymon
267 ?          0:00 lockd
264 ?          0:00 statd
 90 ?         0:00 sysevent
...
462 ?          0:00 smcboot
464 ?          0:00 smcboot
463 ?          0:00 smcboot
473 ?          0:00 htt
552 ?          0:00 in.telne
527 ?          0:00 dmispd
548 ?          0:01 snmpd

```

Where the output differs is in the columns of information that are displayed, but you can fix that by specifically selecting the columns that you want.

Listing specific information

The `ps` tool includes a number of standard sets of columns that are displayed. For example, within SVR4, it is common to use `ps -ef` for more extensive information about the processes listed, including the parent process ID, processor utilization, start time, and a more detailed command line, as shown here in [Listing 5](#).

Listing 5. Extending the output

```

ps -ef
  UID    PID  PPID  C   STIME TTY          TIME CMD
  root     0     0   0 15:56:26 ?           15:24 sched
  root     1     0   0 15:56:26 ?           0:00 /sbin/init
  root     2     0   0 15:56:26 ?           0:00 pageout
  root     3     0   0 15:56:26 ?           0:00 fsflush
  root    308     1   0 15:57:09 ?           0:00 devfsadmd
  root     7     1   0 15:56:29 ?           0:06 /lib/svc/bin/svc.startd
...
  root    562     1   0 15:58:17 ?           0:00 /usr/lib/sendmail -bd -q15m
  root    576    555   1 16:01:47 pts/1       0:00 ps -ef
  root    416     1   0 15:57:14 ?           0:00 /usr/sbin/syslogd
  smmsp   561     1   0 15:58:17 ?           0:00 /usr/lib/sendmail -Ac -q15m
...
  root    552    283   0 15:57:47 ?           0:00 /usr/sbin/in.telnetd
  root    527     1   0 15:57:22 ?           0:00 /usr/lib/dmi/dmispd
  root    548     1   0 15:57:24 ?           0:01 /usr/sfw/sbin/snmpd

```

For BSD environments, you normally add the `-l` option, which produces "long" output from each process, such as [Listing 6](#).

Listing 6. Showing more process detail in BSD

```

$ ps -al
UID PID  PPID  CPU  PRI  NI  VSZ   RSS  WCHAN  STAT  TT  TIME    COMMAND
0  9165  391   0   31   0  57896  6376  -      S+   p5  0:00.50  emacs
501 352   349   0   31   0  27784   52   -      Ss   p6  0:00.04  bash
0  476   352   0   31   0  27784   600  -      S    p6  0:01.05  /bin/bash
0  9299  476   0   31   0  44988  1880  -      S    p6  0:00.09  xterm
0  9319  476   0   31   0  44988  1888  -      S    p6  0:00.07  xterm
0  9423  476   0   31   0  27504   488  -      S    p6  0:00.15  ftp atuin
0  9540  476   0   31   0  27384   504  -      R+   p6  0:00.01  ps -axl
0  9301  9299   0   31   0  27332   452  -      Ss+  p7  0:00.01  /usr/X11R6/bin/luit
0  9302  9301   0   31   0  27784   888  -      Ss+  p8  0:00.03  bash
0  9321  9319   0   31   0  27332   452  -      Ss+  p9  0:00.01  /usr/X11R6/bin/luit
0  9322  9321   0   31   0  27784   888  -      Ss+  pa  0:00.02  bash

```

The problem with these options is that, although they show more information, the information is not always particularly useful, or it contains information that you don't want cluttering up the display when you are looking for a particular process.

Fortunately, all versions of `ps` also include the ability to specify what columns are displayed. You can use this to great effect, either only picking out the information you want or, in a heterogeneous environment, you can use it to create a standardized output across a range of different UNIX environments.

To use this feature, use the `-o` command line option and list the individual columns that you want to display as a comma-separated list. Although there are differences between the ranges of columns available, many of them are standard across different UNIX versions. For example, `pid`, `ppid` (parent pid), `command`, `RSS` (resident set size or memory usage), and `priority` are available in all variants.

In use, you can use `-o` to pick the columns and the display order. For example, to get the `pid`, `ppid`, and `command`, you would use `-opid,ppid,command`, as shown within BSD in [Listing 7](#), or `-opid,ppid,comm` within SVR4, as seen in [Listing 8](#).

Listing 7. Selecting specific columns in BSD

```

$ ps -o pid,ppid,command
PID  PPID  COMMAND
391  332  /bin/bash
9165  391  emacs

```

Listing 8. Selecting specific columns in SVR4

```

$ ps -opid,ppid,comm
PID  PPID  COMMAND
555  552  -sh
622  555  ps

```

Once you have selected the columns you like, you might then want to select how the information is ordered. The default for `ps` is to sort the list of processes by the process ID, but this can have the effect of hiding the very information you are looking

for. When looking for memory hungry processes, it is more useful to display the output listed in order of the memory used, rather than process ID.

Some `ps` variants support this option by using a command line option. BSD variants sort by memory usage when the `-m` option is used, or by CPU usage when using `-r`. Many SVR4 variants don't have a specific solution for this, but you can produce similar results in both environments by combining `ps` with `sort`. For example, to get a list of processes sorted by the amount of CPU they are using, you might use the command in [Listing 9](#) within BSD.

Listing 9. Tracking usage by CPU in BSD

```
$ ps -A -o pid,%cpu,command|sort -n +1
...
358 0.1 ftp
11425 0.1 /bin/bash
28684 0.3 trivial-rewrite -n rewrite -t unix -u
356 0.4 ssh
354 0.5 as
23988 1.1 emacs
136 14.6 cclplus
26306 23.6 cpp
```

Within SVR4, you need to change `%cpu` for `pcpu`, but otherwise the command is identical (see [Listing 10](#)).

Listing 10. Tracking usage by CPU in SVR4

```
$ ps -e -o pid,pcpu,comm|sort -n +1
...
3 0.1 fsflush
555 0.1 -sh
627 0.2 sort
628 0.2 ps
```

This command chain works, because you have specifically ordered your process list columns and can therefore sort on those columns to get the information you really want. If you want to look for processes by a different criteria, there are other solutions available.

Listing specific processes

Obviously, once you get a list of running processes, you might want to list specific processes. An obvious way to do this is to combine the output of `ps` with `grep` to extract the information you want. On some UNIX variants, you might have access to a specific utility for doing this, such as `pgrep`, but often it can be just as effective to use `grep` if you are looking for specific commands:

```
$ ps -ef|grep bash
```

The `ps` command also supports showing processes according to more specific criteria, such as user ID, parent process, or controlling terminal. For example, the `-U` command line option limits the list of processes shown to those with the specified user name. For example, to show the processes currently owned by the root user, see [Listing 11](#).

Listing 11. Listing processes by user

```
$ ps -U root
PID TTY          TIME CMD
  0 ?             15:24 sched
  1 ?             0:00 init
  2 ?             0:00 pageout
  3 ?             0:02 fsflush
 308 ?             0:00 devfsadm
  7 ?             0:06 svc.star
...
 552 ?             0:00 in.telne
 527 ?             0:00 dmispd
 629 pts/1         0:00 ps
 548 ?             0:01 snmpd
```

To get all of the processes on a specified terminal, use `-t`, as shown here in [Listing 12](#).

Listing 12. Listing by terminal

```
$ ps -t 3
PID TTY          TIME CMD
19915 pts/3         00:00:00 bash
29145 pts/3         00:00:00 emacs
32256 pts/3         00:00:00 emacs
```

Once you have this information, you might want to use it to perform some operations on the processes.

Signaling multiple processes

One of the most common commands to run once you have found the process you are looking for is `kill`. It sends one or more processes a specific signal. In the case of a daemon that has started multiple threads or child processes, you can also try sending a signal to the parent process to signal all processes. However, this doesn't work with all daemons and applications.

Obviously, you want to avoid having to manually pick out the processes. Some UNIX variants come with a tool called `pkill`, which can send the same signal to processes matching a particular pattern or other criteria, such as those on a

particular terminal, process group, user ID, or group ID list.

You can simulate the basic operation of sending a signal to multiple processes matching a particular command pattern by chaining together the `ps`, `grep`, `awk`, `xargs` and `kill` commands. For example, to send the `kill` signal to all of the processes with "httpd" in their command, you might use:

```
$ ps -e -opid,command |grep httpd|awk '{print $1}'|xargs kill -9
```

It is easier to understand if you dissect the individual elements:

```
$ ps -e -opid,command
```

This shows a list of all the running processes (in an SVR4 system, use `-A` for BSD). It shows only the process ID and command that was being executed. You don't need any other information, and using more detailed output might introduce text that would otherwise match your search:

```
$ ps -e -opid,command |grep httpd
```

This extracts only processes that have `httpd` in the command (since the only other information being generated from the process list is the process ID):

```
$ ps -e -opid,command |grep httpd|awk '{print $1}'
```

By using `awk`, you filter the output printing out of only the first argument, the process ID:

```
$ ps -e -opid,command |grep httpd|awk '{print $1}'|xargs kill -9
```

The `xargs` command takes a list of space-delimited items (spaces that include returns, linefeeds, tabs, and one or more spaces) and formats it as a list of arguments supplied to the given command -- in this case, the `kill` command.

It is best to place this into a script with a suitable name (`pkill` or `killbyname`). You can set the script to accept two arguments, the signal and the text to be matched, and even account for OS differences, as shown in [Listing 13](#).

Listing 13. Killing processes by command string

```
#!/bin/sh
```

```

HOSTTYPE=`uname -s`

SIGNAL=$1
STRING=$2

if [ -z "$1" -o -z "$2" ]
then
    echo Usage: $0 signal string
    exit 1
fi

case $HOSTTYPE in
    Darwin|BSD)
        ps -a -opid,command | grep $STRING | awk '{ print $1; }' | xargs kill $SIGNAL
        ;;
    Linux|Solaris|AIX|HP-UX)
        ps -e -opid,command | grep $STRING | awk '{ print $1; }' | xargs kill $SIGNAL
        ;;
esac

```

The same basic technique shown here can be used for other similar collations.

Calculating memory usage

The `ps` tool provides two columns that haven't been covered yet. The RSS column provides the "resident set size" of a process; this is the amount of physical memory used by the process and a good indication of how much real memory a given process is using. The VSZ column details the total amount of memory being used by a process, including what is allocated for internal storage, but often swapped to disk. Both of these columns are common to the majority of `ps` variants.

Determining the two figures gives a good idea of memory usage. If you combine the output from `ps` with `grep` to select specific processes and `awk` to calculate the totals, you can get a good idea of how much physical and virtual memory is being used up by single application, or an application and its children.

For example, to determine the amount of real and virtual memory being used by the `bash` process, you could use the command in [Listing 14](#).

Listing 14. Calculating memory usage with `ps` and `awk`

```

$ ps -A -o rss,vsz,command|grep bash | \
  awk '{rss += $1; vsz += $2 } END { print "Real: ",rss, "Virtual: ",vsz }'
Real: 4004 Virtual: 305624

```

This can be particularly useful when you are diagnosing memory and swap usage problems.

Using a job control compatible shell

It is fairly common for a typical system administrator to have more than one or two specific tasks on the go at any one time. Although you might have more than one connection open to a server at any time -- whether that is multiple terminal windows (through xterm, for example) or other terminals, or remotely through SSH or Telnet -- there are times when within your active shell or environment you want to control or monitor multiple processes.

All shells support the option to run a command automatically in the background by appending the ampersand (&) to the end of the command. But sometimes you want to put an interactive application, such as an editor into the background so that you can run a shell command and then return to your editor session.

The ability to control background processes in this way is called job control and a standard feature on both the Korn and C shells, and also open source shells, such as bash and zsh.

For basic job control within the shell every time you start a command to run in the background, the command (which can be any valid command line, even inline scripts) is given a job reference ID.

```
$ find / -name "core" >/tmp/corelist 2>&1 &
[3] 11957
```

You can get a list of the running background jobs using the `jobs` command, as shown in [Listing 15](#).

Listing 15. Using the jobs command

```
$ jobs
[1]-  Stopped          emacs MCSLP/Intranet/News.pm
[2]+  Stopped          emacs MCSLP/Intranet/Media.pm
[3]   Running          find / -name "core" >/tmp/corelist 2>&1 &
```

In the listing, the second `emacs` command is denoted with the + sign. This indicates that the shell considers it to be the currently active job. The `find` command you started previously is not the active job, because it does not require interaction (although it generates output, it doesn't require input to continue), and therefore is not the active process. The first `emacs` process is denoted with -- to indicate that the shell considers this the previously active command. You can reference these jobs using the %+ and %- strings, respectively.

You can switch any of these running jobs to be the foreground process by typing `fg` followed by the job number, or one of the job strings (%+, %-). If you omit a reference, the shell switches to the currently active job.

To suspend the currently running process, press **Control-Z**. You can reconfigure

this using:

```
ftp>
[3]+ Stopped          ftp atuin
```

This works with many different commands and applications. It should also work for the majority of simple commands you run in the shell, such as `ls` or `find`. Note that the job is marked as **Stopped**. This means that the execution of the command has been paused. To switch the command into a background process, use the `bg` command. Like `fg`, `bg` either takes a job reference or defaults to the currently active job with no arguments. If the command is one that requires input (an editor, FTP, and so forth), then the next time you press return after the `bg` command, you will be warned that the process has been temporarily suspended (see [Listing 16](#)).

Listing 16. Warning that process has been temporarily suspended

```
$ bg
[3]+ ftp atuin &
$
[3]+ Stopped
```

Note that if your background command generates output and you haven't redirected it, it will be output to the screen, even if you have placed the command in the background through job control .

In a busy environment, job control becomes one of the easiest ways to manage and control multiple background jobs, or to quickly drop out of editors and other commands back to your active shell (as opposed to running a new shell).

If you want to have access to a job control shell, or anything other than the standard shell when you are in single user or a recovery mode on a UNIX system, then you must make sure that a statically linked copy of the shell is available within one of the standard binary directories, such as `/bin` or `/sbin` (this directory is specifically for statically linked binaries of applications). This will ensure that the shell is available when secondary filesystems have not been loaded.

Running processes reliably in the background

You will occasionally want to run a script, utility, or command line in the background. However, most systems terminate a command running in the background if the user is disconnected, or they log out, which can be opposite of your intentions if you want to log in, start a command, and log out again.

Such termination can be particularly frustrating if you have to restart or otherwise re-initialize background or daemon processes that do not automatically daemonize

themselves, or that rely on a separate script to start and manage the execution of the of a daemon application. The MySQL `mysqld_safe` script is a good example of a script that works in this way.

To prevent the application from automatically terminating when you log out, use the `nohup` command as a prefix to the command line or tool you want to run, like this:

```
$ nohup find/ -name core
```

Unless you specifically redirect the output of your commands, `nohup` automatically writes the standard output and standard error to a file called `nohup.out` in the current directory.

You can output to your own file just by using standard redirection, but remember to redirect both the output and error streams, for example:

```
$ nohup find/ -name core >/tmp/corefind.out 2>&1
```

I find that I almost automatically use `nohup` to run any command that I think will last longer than two or three minutes, even if I am running the command on the console. This probably has as much to do with the automatic redirection of the output than its ability to prevent termination during a connection failure.

Summary

Using the techniques shown in this article, you should be able to quickly find the UNIX processes you want, and the information about them. If you are in a heterogeneous environment, then you can also use the command line options to standardize the output you generate, making it easier for you to find the processes you want.

Resources

Learn

- IBM has a Redbook on AIX® for Sun Solaris Administrators that contains useful information on the differences between AIX from the perspective of Solaris system administrators. You can read an extract on the [IBM Redbooks site](#).
- [System Administration Toolkit](#): Check out other parts in this series.
- [HP-UX 11i System Administration Handbook and Toolkit](#), by Marty Poniowski (Prentice Hall PTR, ISBN: 0-13-060081-3) contains information not only about HP-UX, but also compares the HP-UX commands and generated output to other UNIX operating systems.
- [Bash](#) is an alternative shell to the standard Bourne shell with similar syntax, but an expanded range of features, including aliasing, job control and auto-completion of file and directory names.
- [Zsh](#) provides similar capabilities to Bash but also includes a mechanism for extending certain components of the shell, particularly the auto completion mechanism.
- [AIX® and UNIX articles](#): Check out other articles written by Martin Brown.
- Search the AIX and UNIX library by topic:
 - [System administration](#)
 - [Application development](#)
 - [Performance](#)
 - [Porting](#)
 - [Security](#)
 - [Tips](#)
 - [Tools and utilities](#)
 - [Java technology](#)
 - [Linux](#)
 - [Open source](#)
- [AIX and UNIX](#): The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- [New to AIX and UNIX](#): Visit the New to AIX and UNIX page to learn more about

AIX and UNIX.

- [AIX 5L™ Wiki](#): A collaborative environment for technical information related to AIX.
- [Safari bookstore](#): Visit this e-reference library to find specific technical resources.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- [IBM trial software](#): Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the [developerWorks blogs](#) and get involved in the developerWorks community.
- Participate in the AIX and UNIX forums:
 - [AIX 5L -- technical forum](#)
 - [AIX for Developers Forum](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant](#)
 - [Performance Tools -- technical](#)
 - [Virtualization -- technical](#)
 - [More AIX and UNIX forums](#)

About the author

Martin Brown

Martin Brown has been a professional writer for more than seven years. He is the author of numerous books and articles across a range of topics. His expertise spans myriad development languages and platforms -- Perl, Python, Java™, JavaScript, Basic, Pascal, Modula-2, C, C++, Rebol, Gawk, Shellscript, Windows®, Solaris, Linux®, BeOS, Mac OS X and more -- as well as Web programming, systems management, and integration. He is a Subject Matter Expert (SME) for Microsoft and regular contributor to ServerWatch.com, LinuxToday.com, and IBM developerWorks. He is also a regular blogger at Computerworld, The Apple Blog, and other sites. You

can contact him through [his Web site](#).

Trademarks

IBM, AIX, and AIX 5L are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.