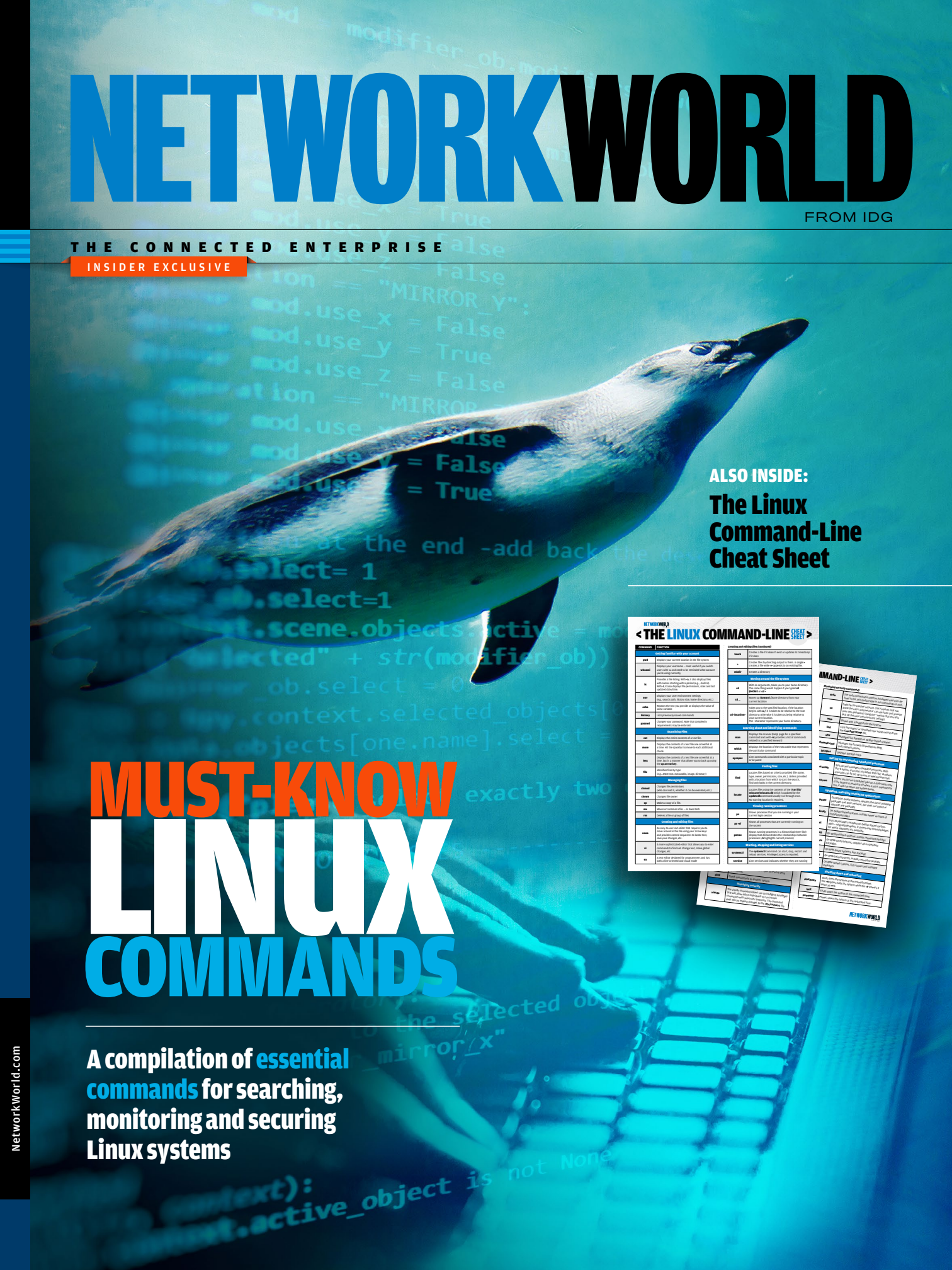
NETWORKWORLD
FROM IDG

THE CONNECTED ENTERPRISE

INSIDER EXCLUSIVE



MUST-KNOW LINUX COMMANDS

A compilation of essential commands for searching, monitoring and securing Linux systems

ALSO INSIDE:

The Linux Command-Line Cheat Sheet

IT TAKES SOME TIME WORKING with Linux commands before you know which one you need for the task at hand, how to format it and what result to expect, but it's possible to speed up the process.

With that in mind, we've gathered together some of the essential commands an aspiring power user needs to get started with Linux systems.

The breakdown starts with the rich options available for finding files on Linux – **find**, **locate**, **mlocate**, **which**, **whereis**, to name a few. Some overlap, but some are more efficient than others or zoom in on a narrow set of results. There's even a command – **apropos** – to find the right command to do what you want to do. This section will demystify searches.

Editor's NOTE

Our article on memory provides a wealth of options for discovering the availability of physical and virtual memory and ways to have that information updated at intervals to constantly measure whether there's enough memory to go around. We show how it's possible to tailor your requests so you get a concise presentation of the results you seek.

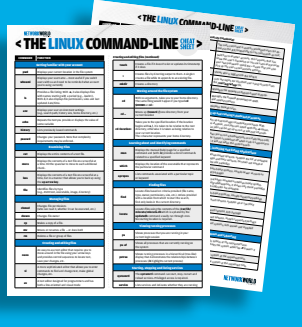
Two remaining articles in this package show how to monitor activity on Linux servers and how to set up security parameters on these systems.

As a bonus, our bundle of commands includes The Linux Command-Line Cheat Sheet, a concise summary of important commands suitable for printing out on two sides of a single sheet, laminating and keeping beside your keyboard.

Enjoy! ◆

TIM GREENE *is Executive Editor for Network World.*

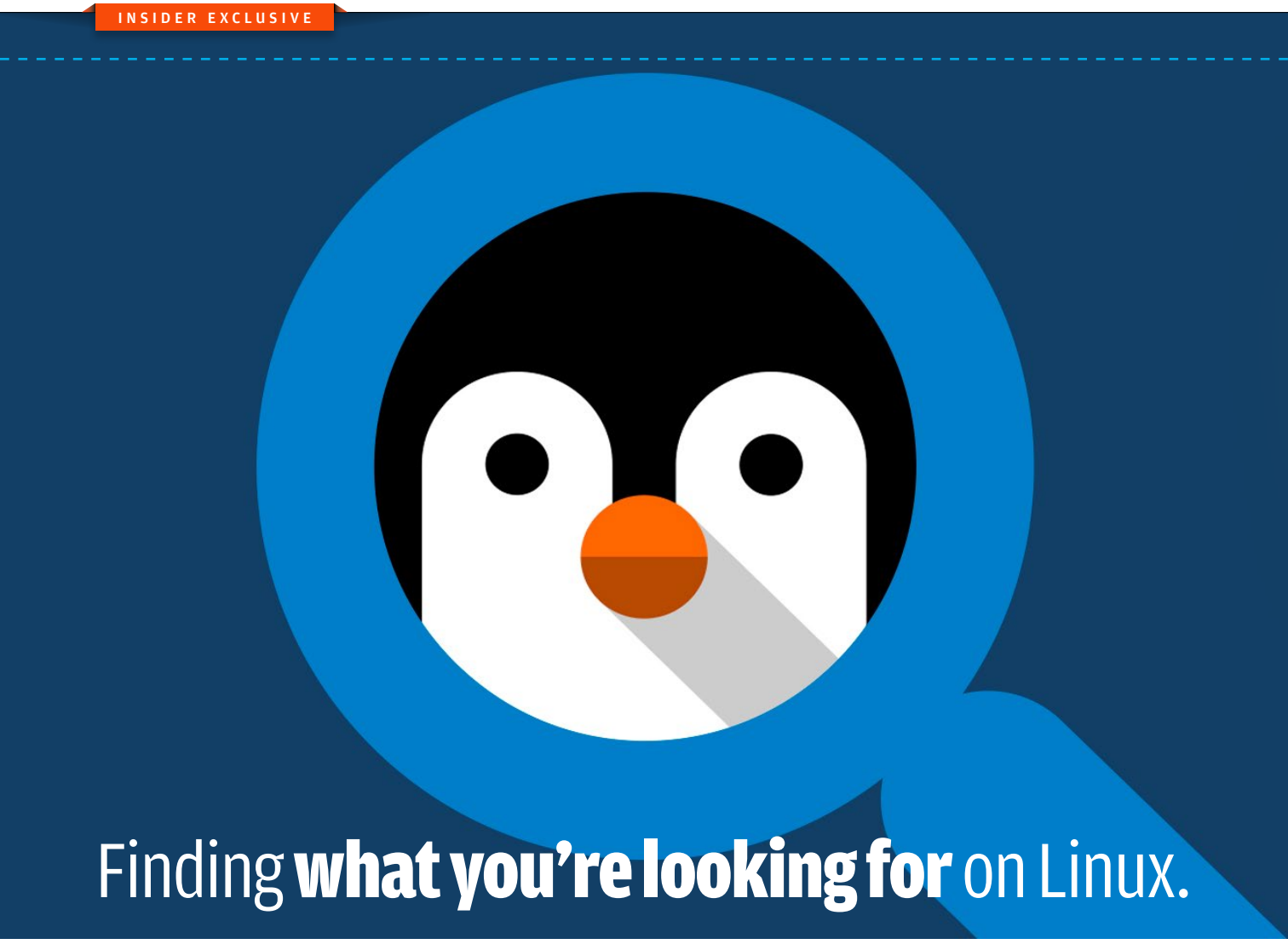
INSIDE



Download, print and laminate.

SANDRA HENRY-STOCKER *has been administering Unix systems for more than 30 years. You can contact her at bugfarm@gmail.com.*

INSIDER EXCLUSIVE



Finding **what you're looking for** on Linux.

How to use the **find**, **locate**, **mlocate**, **which**, **whereis**, **whatis** and **apropos** commands to find files on Linux systems

BY SANDRA HENRY-STOCKER

THE COMMANDS AVAILABLE ON LINUX for locating and identifying files are quite varied, but they're all very useful. Some commands have functionality that overlaps a bit, but they aren't identical and in fact may work differently and come up with different results. This is a description of a few, how to use them and what each offers uniquely.

```
$ find e*
empty
examples.desktop
```

The most obvious command in this category is undoubtedly the **find** command, which has become easier to use than it was years ago. It used to require a starting location for your search, but these days, you can also use find with just a file name or regular expression if you're willing to confine your search to the local directory.

SASHATIGAR / GETTY IMAGES

In this way, it works much like the **ls** command and isn't doing much of a search.

For more relevant searches, **find** requires a starting point and some criteria for your search unless you simply want it to provide a recursive listing of that starting point's directory. The command **find . -type f** will recursively list all regular files starting with the current directory while **find ~nemo -type f -empty** will find empty files in Nemo's home directory.

```
$ find ~nemo -type f -empty
/home/nemo/empty
```

locate

The name of the **locate** command suggests that it does basically the same thing as **find**, but it works entirely differently. Where the **find** command can select files based on a variety of criteria – name, size, owner, permissions, state (such as empty), etc., with a selectable depth for the search – the **locate** command looks through a file called /var/lib/mlocate/mlocate.db to find what you're looking for. That db file is periodically updated, so a **locate** of a file you just created will probably fail to find it. If that bothers you, you can run the **update.db** file and get the update to happen right away.

```
$ sudo updatedb
```

mlocate

The **mlocate** command works like the **locate** command and uses the same mlocate.db file as **locate**.

which

The **which** command works very differently than the **find** and **locate** commands. It uses your search path and checks each directory on it for an executable with the file name you're looking for. Once it finds one, it stops searching and displays the full path to that executable.

The primary benefit of the **which** command is that it answers the question, "If I enter this command, what executable file will be run?" It ignores files that aren't executable and doesn't list all executables on the system with that name – just the one that it finds first. If you wanted to find all executables that have some name, you could run a **find** command like this, but it might take considerably longer to run than the very efficient **which** command.

```
$ find / -name locate -perm -a=x 2>/dev/null
/usr/bin/locate
/etc/alternatives/locate
```

In this find command, we're looking for all executables (files that can be run by anyone) named "locate". We're also electing not to view all of the "Permission denied" messages that would otherwise clutter our screens.

whereis

The **whereis** command works a lot like the **which** command, but it provides more information. Instead of just looking for executables, it also looks for man pages and source files. Like the **which** command, it uses your search path ($PATH) to drive its search.

```
$ whereis locate
locate: /usr/bin/locate /usr/share/man/man1/locate.1.gz
```

whatis

The **whatis** command has its own unique mission. Instead of actually finding files, it looks for information in the man pages for the command you are asking about and provides the brief description of the command from the top of the man page.

```
$ whatis locate
locate (1)          - find files by name
```

If you ask about a script that you've just set up, it won't have any idea what you're referring to and will tell you so.

```
$ whatis cleanup
cleanup: nothing appropriate.
```

apropos

The **apropos** command is useful when you know what you want to do, but you have no idea what command you should be using to do it. If you were wondering how to locate files, for example, the commands "apropos find" and "apropos locate" would have a lot of suggestions to offer.

```
$ apropos find
File::IconTheme (3pm)                - find icon directories
File::MimeInfo::Applications (3pm)   - Find programs to open a file by mimetype
File::UserDirs (3pm)                 - find extra media and documents directories
```

```
find (1)                    - search for files in a directory hierarchy
findfs (8)                  - find a filesystem by label or UUID
findmnt (8)                 - find a filesystem
gst-typefind-1.0 (1)        - print Media type of file
ippfind (1)                 - find internet printing protocol printers
locate (1)                  - find files by name
mlocate (1)                 - find files by name
pidof (8)                   - find the process ID of a running program.
sane-find-scanner (1)       - find SCSI and USB scanners and their device files
systemd-delta (1)           - Find overridden configuration files
xdg-user-dir (1)            - Find an XDG user dir
$
$                            -apropos locate
blkid (8)                   - locate/print block device attributes
deallocvt (1)               - deallocate unused virtual consoles
fallocate (1)               - preallocate or deallocate space to a file
IO::Tty (3pm)               - Low-level allocate a pseudo-Tty, import constants.
locate (1)                  - find files by name
mlocate (1)                 - find files by name
mlocate.db (5)              - a mlocate database
mshowfat (1)                - shows FAT clusters allocated to file
ntfsfallocate (8)           - preallocate space to a file on an NTFS volume
systemd-sysusers (8)        - Allocate system users and groups
systemd-sysusers.service (8) - Allocate system users and groups
updatedb (8)                - update a database for mlocate
updatedb.mlocate (8)        - update a database for mlocate
whereis (1)                 - locate the binary, source, and manual page files for a...
which (1)                   - locate a command
```

SASHATIGAR / GETTY IMAGES

INSIDER EXCLUSIVE

How much memory is installed and being used on your Linux systems?

Several commands report on how much memory is installed and being used on Linux systems. You can be deluged with details or get a quick and easy answer, depending on the command you use. BY SANDRA HENRY-STOCKER

THERE ARE NUMEROUS WAYS TO get information on the memory installed on Linux systems and view how much of that memory is being used. Some commands provide an overwhelming amount of detail, while others provide succinct, though not necessarily easy-to-digest, answers. Here, we'll look at some of the more useful tools for checking on memory and its usage.

Before we get into the details, however, let's review a few basics. Physical memory and virtual memory are not the same. The latter includes disk space configured to be used as swap. Swap may include partitions set aside for this usage. Or files that are created to add to the available swap space when creating a new partition may not be practical. Some Linux commands provide information on both.

PANUWAT SIKHAM / BALOON111 / GETTY IMAGES

Swap expands memory by providing disk space that can be used to house inactive pages that are moved to disk when physical memory fills up.

One file that plays a role in memory management is **/proc/kcore**. This file looks like a normal (though extremely large) file, but it does not occupy disk space at all. Instead, it is a virtual file like all of the files in **/proc**.

```
$ ls -l /proc/kcore
-r--------. 1 root root 140737477881856 Jan 28 12:59 /proc/kcore
```

Interestingly, the two systems queried below do *not* have the same amount of memory installed, yet the size of **/proc/kcore** is the same on both. The first of these two systems has 4 GB of memory installed; the second has 6 GB.

```
system1$ ls -l /proc/kcore
-r--------. 1 root root 140737477881856 Jan 28 12:59 /proc/kcore

system2$ ls -l /proc/kcore
-r--------  1 root root 140737477881856 Feb  5 13:00 /proc/kcore
```

Explanations that claim the size of this file represents the amount of available virtual memory (maybe plus 4K) don't hold much weight. This number would suggest that the virtual memory on these systems is 128 terabytes! The number seems to represent instead how much memory a 64-bit systems might be capable of addressing – not how much is available on the system. Calculations of what 128 terabytes and that number, plus 4K, would look like are fairly easy to make on the command line:

```
$ expr 1024 \* 1024 \* 1024 \* 1024 \* 128
140737488355328

$ expr 1024 \* 1024 \* 1024 \* 1024 \* 128 + 4096
140737488359424
```

Another and more human-friendly command for examining memory is the **free** command. It gives you an easy-to-understand report on memory.

$ free

	TOTAL	USED	FREE	SHARED	BUFF/CACHE	AVAILABLE
Mem:	6102476	812244	4090752	13112	1199480	4984140
Swap:	2097148	0	2097148			

With the **-g** option, **free** reports the values in gigabytes.

```
$ free -g
              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:              5           0           3           0           1           4
Swap:             1           0           1
```

With the **-t** option, **free** shows the same values as it does with no options (don't confuse **-t** with terabytes!) but by adding a total line at the bottom of its output.

```
$ free -t
              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:        6102476      812408     4090612       13112     1199456     4983984
Swap:       2097148           0     2097148
Total:      8199624      812408     6187760
```

And, of course, you can choose to use both options.

```
$ free -tg
              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:              5           0           3           0           1           4
Swap:             1           0           1
Total:            7           0           5
```

You might be disappointed in this report if you're trying to answer the question "How much RAM is installed on this system?" This is the same system shown in the example above that was described as having 6GB of RAM. That doesn't mean this report is wrong, but that it's the system's view of the memory it has at its disposal.

The **free** command also provides an option to update the display every X seconds (10 in the example below).

```
$ free -s 10
              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:        6102476      812280     4090704       13112     1199492     4984108
Swap:       2097148           0     2097148

              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:        6102476      812260     4090712       13112     1199504     4984120
Swap:       2097148           0     2097148
```

With **-l**, the **free** command provides high and low memory usage.

```
$ free -l
              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:        6102476      812376     4090588       13112     1199512     4984000
Low:        6102476     2011888     4090588
High:             0           0           0
Swap:       2097148           0     2097148
```

Another option for looking at memory is the **/proc/meminfo** file. Like **/proc/kcore**, this is a virtual file and one that gives a useful report showing how much memory is installed, free and available. Clearly, free and available do not represent the same thing. MemFree seems to represent unused RAM. MemAvailable is an estimate of how much memory is available for starting new applications.

```
$ head -3 /proc/meminfo
MemTotal:                6102476 kB
MemFree:                 4090596 kB
MemAvailable:            4984040 kB
```

If you only want to see total memory, you can use one of these commands:

```
$ awk '/MemTotal/ {print $2}' /proc/meminfo
                         6102476
$ grep MemTotal /proc/meminfo
MemTotal:                6102476 kB
```

The **DirectMap** entries break information on memory into categories.

```
$ grep DirectMap /proc/meminfo
DirectMap4k:             213568 kB
DirectMap2M:            6076416 kB
```

DirectMap4k represents the amount of memory being mapped to standard 4k pages, while DirectMap2M shows the amount of memory being mapped to 2MB pages.

The **getconf** command is one that will provide quite a bit more information than most of us want to contemplate.

```
$ getconf -a | more
LINK_MAX                        65000
_POSIX_LINK_MAX                 65000
MAX_CANON                       255
_POSIX_MAX_CANON                255
MAX_INPUT                       255
_POSIX_MAX_INPUT                255
NAME_MAX                        255
_POSIX_NAME_MAX                 255
PATH_MAX                        4096
_POSIX_PATH_MAX                 4096
PIPE_BUF                        4096
SOCK_MAXBUF
_POSIX_ASYNC_IO
_POSIX_CHOWN_RESTRICTED         1
_POSIX_NO_TRUNC                 1
_POSIX_PRIO_IO
_POSIX_SYNC_IO
_POSIX_VDISABLE                 0
ARG_MAX                         2097152
ATEXIT_MAX                      2147483647
CHAR_BIT                        8
CHAR_MAX                        127
--More--
```

Pare that output down to something specific with a command like the one shown below, and you'll get the same kind of information provided by some of the commands above.

```
$ getconf -a | grep PAGES | awk 'BEGIN {total = 1} {if (NR == 1 || NR == 3) total *=$NF} END {print total
/1024" kB"}'
6102476 kB
```

That command calculates memory by multiplying the values in the first and last lines of output like this:

```
PAGESIZE                        4096            <==
_AVPHYS_PAGES                   1022511
_PHYS_PAGES                     1525619         <==
```

Calculating that independently, we can see how that value is derived.

```
$ expr 4096 \* 1525619 / 1024
6102476
```

Clearly that's one of those commands that deserves to be turned into an alias!

Another command with very digestible output is **top**. In the first five lines of **top**'s output, you'll see some numbers that show how memory is being used.

```
$ top
top - 15:36:38 up 8 days,     2:37,     2 users,     load average: 0.00, 0.00, 0.00
Tasks: 266 total,          1 running,          265 sleeping,          0 stopped,          0 zombie
%Cpu(s):    0.2 us,    0.4 sy,    0.0 ni,    99.4 id,    0.0 wa,    0.0 hi,    0.0 si,    0.0 st
MiB Mem :     3244.8 total,      377.9 free,     1826.2 used,     1040.7 buff/cache
MiB Swap:     3536.0 total,     3535.7 free,        0.3 used.     1126.1 avail Mem
```

And finally a command that will answer the question "So, how much RAM is installed on this system?" in a succinct fashion:

```
$ sudo dmidecode -t 17 | grep "Size.*MB" | awk '{s+=$2} END {print s / 1024 "GB"}'
6GB
```

Depending on how much detail you want to see, Linux systems provide a lot of options for seeing how much memory is installed on your systems and how much is used and available. ■

PANUWAT SIKHAM / BALOON111 / GETTY IMAGES

How to **monitor activity** on your **Linux server**

The **watch**, **top**, and **ac** commands provide some effective ways to **oversee what is happening on your Linux servers.**

BY SANDRA HENRY-STOCKER

LINUX SYSTEMS PROVIDE A NUMBER of commands that make it easy to report on system activity. In this article, we're going to look at three commands that are especially helpful.

There are many commands for examining system activity. The **watch** command allows you to run just about any command in a repetitive way and watch how the output changes. The **top** command is a better option for focusing on user processes and also loops in a way that allows you to see the changes as they happen, while the **ac** command examines user connect time.

GRAFISSIMO / MARCO_PIUNTI / GETTY IMAGES

The watch command

The **watch** command is one that makes it easy to repeatedly examine a variety of data on your system – user activities, running processes, logins, memory usage, etc. All the command really does is run the command that you specify repeatedly, each time overwriting the previously displayed output, but this lends itself to a very convenient way of monitoring what's happening on your system.

To start with a very basic and not particularly useful command, you could run **watch -n 5 date** and see a display with the current date and time that updates every 5 seconds. As you likely have guessed, the **-n 5** option specifies the number of seconds to wait between each run of the command. The default is 2 seconds. The command will run and update a display like this until you stop it with a **^c**.

```
Every 5.0s: date                                  butterfly: Wed Jan 23 15:59:14 2019

Wed Jan 23 15:59:14 EST 2019
```

As a more interesting command example, you can watch an updated list of whoever is logging into the server. As written, this command will update every 10 seconds. Users who log out will disappear from the current display and those who log in will come into view. If no one is logging in or out, the display will remain the same except for the time displayed.

```
$ watch -n 10 who
```

```
Every 10.0s: who                                  butterfly: Tue Jan 23 16:02:03 2019

shs          :0           2019-01-23 09:45 (:0)
dory         pts/0        2019-01-23 15:50 (192.168.0.5)
nemo         pts/1        2019-01-23 16:01 (192.168.0.15)
shark        pts/3        2019-01-23 11:11 (192.168.0.27)
```

If you just want to see how many users are logged in, you can get a user count along with load averages showing you how hard the system is working by having **watch** call the **uptime** command.

```
$ watch uptime
```

```
Every 2.0s: uptime                                butterfly: Tue Jan 23 16:25:48 2019

 16:25:48 up 22 days,  4:38,  3 users,  load average: 1.15, 0.89, 1.02
```

If you want to use **watch** to repeat a command that includes a pipe, you need to put the command between single quote marks like this command that every 5 seconds shows you how many processes are running:

```
$ watch -n 5 'ps -ef | wc -l'
```

```
Every 5.0s: ps -ef | wc -l                    butterfly: Tue Jan 23 16:11:54 2019

245
```

To watch memory usage, you might try a command like this one:

```
$ watch -n 5 free -m
```

```
Every 5.0s: free -m                           butterfly: Tue Jan 23 16:34:09 2019

              TOTAL        USED        FREE      SHARED  BUFF/CACHE   AVAILABLE
Mem:           5959         776        3276          12        1906        4878
Swap:          2047           0        2047
```

The top command

You could watch processes being run by one particular user with **watch**, but the top command provides a much better option.

If you want to watch one particular user's processes, **top** has an ideal option for you – the **-u** option:

```
$ top -u nemo
top - 16:14:33 up 2 days,  4:27,  3 users,  load average: 0.00, 0.01, 0.02
Tasks: 199 total,   1 running,  198 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   5959.4 total,   3277.3 free,    776.4 used,   1905.8 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   4878.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
23026 nemo      20   0   46340   7820   6504 S   0.0   0.1   0:00.05 systemd
23033 nemo      20   0  149660   3140     72 S   0.0   0.1   0:00.00 (sd-pam)
23125 nemo      20   0   63396   5100   4092 S   0.0   0.1   0:00.00 sshd
23128 nemo      20   0   16836   5636   4284 S   0.0   0.1   0:00.03 zsh
```

You not only see what processes the user is running, but the resources (CPU time and memory) that the process is consuming and how hard the system is working overall.

The ac command

If you'd like to see how much time each of your users is spending logged in, you can make use of the **ac** command. This requires installation of the **acct** (Debian) or **psacct** (RHEL, Centos, etc.) package.

The **ac** command has a number of options, but it pulls its data from the current **wtmp** file. Here's an example showing the total number of hours users were logged in recently:

```
$ ac
        total        1261.72
```

This command shows total hours by user:

```
$ ac -p
        shark           5.24
        nemo            5.52
        shs          1251.00
        total        1261.76
```

This ac command shows daily counts of how many hours users were logged in:

```
$ ac -d | tail -10
Jan 11          total           0.05
Jan 12          total           1.36
Jan 13          total          16.39
Jan 15          total          55.33
Jan 16          total          38.02
Jan 17          total          28.51
Jan 19          total          48.66
Jan 20          total           1.37
Jan 22          total          23.48
Today           total           9.83
```

Examining partitions on Linux systems

Linux systems provide many ways to look at disk partitions. Here's a look at commands you can use to display useful information – each providing a different format and with a different focus. **BY SANDRA HENRY-STOCKER**

LINUX SYSTEMS PROVIDE MANY WAYS to look at disk partitions. In this article, we'll look at a series of commands, each of which shows useful information but in different formats and with different focuses. Maybe one will make your favorites list.

lsblk

One of the most useful commands is the **lsblk** (list block devices) command that provides a very nicely formatted display of block devices and disk partitions. In the example below, we can see that the system has two disks, sda and sdb, and that sdb has both a very small (500MB) partition and a large one (465.3GB).

Disks and partitions (part) are clearly labeled, and the relationship between the disks and partitions is quite obvious. We also see that the system has a cdrom (sr0).

```
$ lsblk | grep -v loop
NAME        MAJ:MIN  RM      SIZE  RO TYPE MOUNTPOINT
sda           8:0     0     111.8G   0 disk
└─sda1        8:1     0     111.8G   0 part /
sdb           8:16    0     465.8G   0 disk
├─sdb1        8:17    0      500M    0 part
└─sdb2        8:18    0     465.3G   0 part /apps
sr0           11:0    1     1024M    0 rom
```

A block device is a device that the system reads and writes one block of data at a time, unlike, for example, character and serial devices. The command above is skipping over loop devices with the **grep -v** command.

cat /proc/partitions

Looking at the **partitions** file shows similar data, but not in as nearly a useful format. Partitions and disk sizes are reported in blocks. It does, however, show the major and minor device numbers. Note that we are again selecting lines (only those containing an "s") to omit loop devices.

```
$ cat /proc/partitions | grep s
MAJOR MINOR  #BLOCKS  NAME
    8     0  117220824 sda
    8     1  117219328 sda1
   11     0    1048575 sr0
    8    16  488386584 sdb
    8    17     512000 sdb1
    8    18  487872512 sdb2
```

df

The **df** (disk usage) command can provide a succinct report on disk partitions, but only mounted partitions. It shows file system sizes along with measurements of how much of the disk space is used. In this example, we're using **-h** (human-readable size measurements), **-T** (include partition type), and **-t** (select partitions of the specified type) options.

```
$ df -hTt ext4
Filesystem     Type      Size  Used Avail Use% Mounted on
/dev/sda1      ext4      110G  9.0G   95G   9% /
/dev/sdb2      ext4      457G   73M  434G   1% /apps
```

mount

Another command for listing file systems that are mounted along with their types and mount options is the **mount** command itself. This command shows us nothing about partition sizes, but the options are interesting.

```
$ mount | grep ^/dev
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/sdb2 on /apps type ext4 (rw,relatime,data=ordered)
```

In the above output, we see the partitions, mount points, file system type, and these options:

- rw – read/write
- relatime – access time (atime) will not be written to the disc during every access
- errors=remount-ro – file system will be remounted as read-only if errors are detected
- data=ordered – file data written to file system prior to metadata being committed to the journal

The remount-ro choice is specified in the /etc/fstab file. The **tune2fs** command allows us to adjust other file system parameters.

fdisk

The **fdisk** command provides ways to manipulate disk partitions, but with the **-l** option, we are only displaying partition descriptions on ext2, ext3 and ext4 file systems.

```
$ sudo fdisk -l /dev/sda1
Disk /dev/sda1: 111.8 GiB, 120032591872 bytes, 234438656 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
$ sudo sfdisk -l /dev/sda1
Disk /dev/sda1: 111.8 GiB, 120032591872 bytes, 234438656 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

parted

Like **sfdisk**, the **parted** command allows us to manipulate partitions but offers a **-l** option for displaying partition descriptions.

```
$ sudo parted -l
Model: ATA SSD2SC120G1CS175 (scsi)
Disk /dev/sda: 120GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  120GB   120GB   primary  ext4         boot

Model: ATA SAMSUNG HE502HJ (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  525MB   524MB   primary  ntfs         boot
 2      525MB   500GB   500GB   primary  ext4
```

blkid

The **blkid** command displays block-device attributes. It shows the 128-bit UUID and file system type, but it is not likely to be the most illuminating command for providing partition stats.

```
$ sudo blkid
/dev/sda1: UUID="235f8f0c-641a-4d83-b584-7280cfcd3faa" TYPE="ext4" PARTUUID="f63b5929-01"
/dev/sdb1: LABEL="System Reserved" UUID="08F0AF99F0AF8C0E" TYPE="ntfs"
PARTUUID="7e67ccf3-01"
/dev/sdb2: UUID="94a29e41-b282-4ca2-aacb-fbd91507dc27" TYPE="ext4" PARTUUID="7e67ccf3-02"
```

hwinfo

The **hwinfo** command probes system hardware and provides a nicely organized display of disks, partitions and, if available, other block devices. This short report displays only a fraction of the information the command will report. The **--block** and the **--short** options specify that we only want the command to provide summaries on block devices.

```
$ hwinfo --block --short
disk:
  /dev/sdb              SAMSUNG HE502HJ
  /dev/sda              SSD2SC120G1CS175
partition:
  /dev/sdb1             Partition
  /dev/sdb2             Partition
  /dev/sda1             Partition
cdrom:
  /dev/sr0              HL-DT-ST DVD+-RW GSA-H73N
```

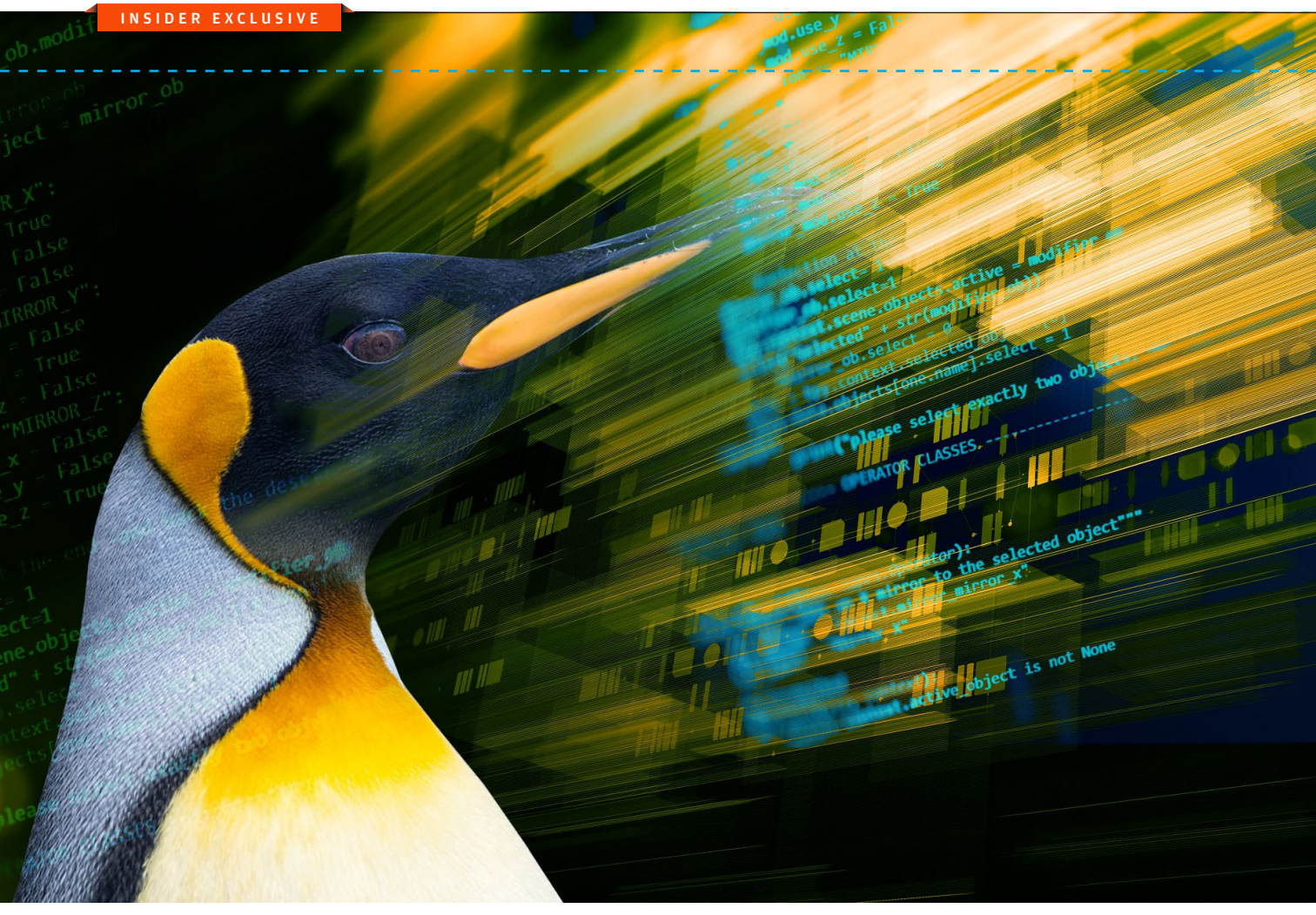
file

Even the **file** command can shed some light on partitions. In the example below, we see the partitions, file system types and the 128-bit UUID numbers.

```
$ sudo file -s /dev/sda1
/dev/sda1: Linux rev 1.0 ext4 filesystem data, UUID=235f8f0c-641a-4d83-b584-7280cfcd3faa (needs
journal recovery) (extents) (64bit) (large files) (huge files)
```

```
$ sudo file -s /dev/sdb2
/dev/sdb2: Linux rev 1.0 ext4 filesystem data, UUID=94a29e41-b282-4ca2-aacb-fbd91507dc27 (extents
(64bit) (large files) (huge files)
```

INSIDER EXCLUSIVE



22 essential Linux security commands

If you're looking after the security of Linux systems, these are the commands you must know. BY SANDRA HENRY-STOCKER

THERE ARE MANY ASPECTS TO security on Linux systems, from setting up accounts to ensuring that legitimate users have no more privilege than they need to do their jobs. This is look at some of the most essential security commands for day-to-day work on Linux systems.

sudo

Running privileged commands with **sudo** – instead of switching user to root – is one essential good practice as it helps to ensure that you only use root privilege when needed and limits the impact of mistakes. Your access to the **sudo** command depends on settings in the **/etc/sudoers** and **/etc/group** files.

```
$ sudo adduser shark
Adding user `shark' ...
Adding new group `shark' (1007) ...
Adding new user `shark' (1007) with group `shark' ...
Creating home directory `/home/shark' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for shark
Enter the new value, or press ENTER for the default
        Full Name []: shark
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

If you run **sudo** and ask who you are, for example, you'll get confirmation that you're running the command as root.

```
$ sudo whoami
root
```

visudo

If you manage the **sudo** setup for users, you also need to be comfortable with the **visudo** command.

The **visudo** command allows you to make changes to the /etc/sudoers file by opening the file in a text editor and checking your changes for syntax. Run the command with "sudo visudo" and make sure that you understand the syntax. Privileges can be assigned by user or by group. On most Linux systems, the **/etc/sudoers** file will already be configured with groups like those shown below that allow the privileges to be assigned to groups set up in the **/etc/group** file. In those cases, you don't need to use the **visudo** command at all – just be familiar with the groups that bestow root privileges in this way, and make your updates to the **/etc/group** file. Note that group names are preceded by the % sign.

```
%admin ALL=(ALL) ALL
%sudo ALL=(ALL:ALL) ALL
%wheel ALL=(ALL:ALL) ALL
```

You can probably display the group providing **sudo** access in your **/etc/group** file like this since it is probably one of these:

```
$ egrep “admin|sudo|wheel” /etc/group
sudo:x:27:shs,jdoe
```

The easiest way to give someone **sudo** privilege is to add them to the empowered group in **/etc/group**. However, that means that they can run any command as root. If you want some users to have root authority for a limited set of commands, such as adding and removing accounts, you can define the commands you want them to be able to run through a command alias like this:

```
Cmnd_Alias ACCT_CMDS = /usr/sbin/adduser, /usr/sbin/deluser
```

Then give the user or group the ability to run these commands using **sudo** with a command like one of these:

```
nemo ALL=(ALL) ACCT_CMDS
%techs  ALL=(ALL:ALL) ACCT_CMDS
```

The first line allows the user “nemo” to run the two (**adduser** and **deluser**) commands with sudo while the second assigns the same privileges to anyone in the “tech” group in the **/etc/group** file.

who and w

The **who** and **w** commands show you who is logged into the system, though **w** shows more information such as where they logged in from, when they logged in and how long they’ve been idle.

```
$ w
18:03:35 up 9 days, 22:48,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
joe      tty2     /dev/tty2        27Apr18  9days  7:34   0.09s /usr/lib/x86_64-linux
shs      pts/1    192.168.0.15     09:50    7.00s  0.28s  0.00s w
```

Use the “sudo update-alternatives – config editor” command if you don’t like the default editor that’s called into play when you run the **visudo** command. It will offer a number of editors as options and change your setting.

last

The **last** command shows you recent logins for users and is often helpful when you're trying to track down changes or other activity.

```
$ last nemo
nemo  pts/1  192.168.0.15  Wed May  2 07:01 - 08:29  (01:27)
wtmp begins Tue May  1 10:21:35 2018
```

Nemo hasn't logged in for a while. He might be on vacation (maybe fishing?) or have just recently left the company. This kind of information can be useful in deciding whether you need to follow up on this.

find

The **find** command is used for many types of searches. When it comes to security, you might find yourself looking for files that have no owners (no corresponding accounts) or are both world-writable and executable. **Find** commands are easy to compose but require some familiarity with its many options for defining what you're looking for. The first of these two commands will find files with no currently defined owners. The second will find files that likely anyone can both run and modify.

```
$ sudo find /home -nouser
$ sudo find / -perm -o=wx
```

Keep in mind that the **-o** in the second command refers to the "other" group – not the owner and not the group associated with the files.

file

The **file** command looks at a file and determines what kind of file it is based on its contents, not its name. Many files (like jpeg files) contain identifiers near the beginnings of the files that identify them. The ".jpg" file in the example below is clearly not a true jpeg file but an executable – in spite of its name.

```
jdoe@stinkbug:~$ ls -l
total 24
-rw-r--r-- 1  root  root      0  Apr  13  09:59  empty
-rwxr-xr-x 1  jdoe  jdoe  18840  May  10   17:39  myphoto.jpg
-rwx------ 1  jdoe  jdoe     24  May   2  07:06  trythis
jdoe@stinkbug:~$ file myphoto.jpg
myphoto.jpg: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/
ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=5d19f7a492405ea9b022a9aa8151f6fb4143633d,
stripped
```

which

The **which** command identifies the executable that will be run when you type its name. This won't always be what you think it is. If a Trojan has been inserted into the file system in a location that shows up in your search path before the legitimate one, it will be run instead. This is a good reason to ensure that your search path includes directories like /usr/bin before it adds less-standard locations and especially before "**.**" (the current directory).

```
$ which date
/usr/local/bin/date     <===     probably not what we wanted
```

You can check a user's search path by switching to the user and echoing it:

```
shs@stinkbug:~$ sudo su - nemo
nemo@stinkbug:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/snap/bin
```

Even if users' search paths are set up in a system file like **/etc/profile** or **/etc/bash.bashrc**, they may have been altered by local settings.

```
$ which date
/usr/local/bin/date     <===     probably not what we wanted
```

You can check a user's search path by switching to the user and echoing it:

```
shs@stinkbug:~$ sudo su - nemo
nemo@stinkbug:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/snap/bin
```

Even if users' search paths are set up in a system file like **/etc/profile** or **/etc/bash.bashrc**, they may have been altered by local settings.

ss

The **ss** command is a tool for investigating sockets and allows you to do things like display listening ports and active connections. Without adding some constraints, **ss** is going to display a lot more information than you probably want to look at. After all, many parts of the operating system communicate via sockets. If you want to generate a list of established connections or listening ports (i.e., services available to external systems), commands like these will prove very useful.

Established connections:

```
$ ss -t
State   Recv-Q Send-Q Local Address:Port     Peer Address:Port
ESTAB   0      224    192.168.0.20:ssh      192.168.0.15:56647
```

```
$ ss | grep ESTAB | grep tcp
tcp    ESTAB   0    64   192.168.0.20:ssh   192.168.0.15:64885
```

Listening ports:

```
$ ss -ltn
State       Recv-Q      Send-Q           Local Address:Port           Peer Address:Port
LISTEN      0                128                    *:22                        *:*
LISTEN      0                  5             127.0.0.1:631                       *:*
LISTEN      0                 50                    *:445                       *:*
LISTEN      0                 50                    *:139                       *:*
LISTEN      0                128                   *:5355                       *:*
LISTEN      0                128                   :::22                       :::*
LISTEN      0                  5                  ::1:631                      :::*
LISTEN      0                 50                  :::445                       :::*
LISTEN      0                128                  :::5355                      :::*
LISTEN      0                 50                  :::139                       :::*
```

Notice that port 631 (CUPS) is only listening on the loopback interface (127.0.0.1).

ufw

If you are running a firewall on your Linux system – an important step for controlling access to the system – the commands used to start/stop, enable/disable, modify and display the status or active rules are critical. Here are some sample commands for **ufw** – the "uncomplicated firewall" that you will find on many Ubuntu systems:

```
$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22                         ALLOW IN    192.168.0.0/24
```

This firewall is active and is allowing only connections from the local network to **ssh** in.

The following commands would 1) set up the rule shown above and 2) disable the firewall.

```
$ sudo ufw allow from 192.168.0.0/24 to any port 22
$ sudo ufw disable
```

iptables

It's also important to know how to list firewall rules for **iptables**. These commands will provide you with a complete list of netfilter rules:

```
sudo iptables -vL -t filter
sudo iptables -vL -t nat
sudo iptables -vL -t mangle
sudo iptables -vL -t raw
sudo iptables -vL -t security
```

ip

The **ip** command allows you to display information on your network interfaces. In the example below, we see the loopback and the public interface.

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
```

```
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
qlen 1000
    link/ether 00:1e:4f:c8:43:fc brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.20/24 brd 192.168.0.255 scope global dynamic enp0s25
        valid_lft 59794sec preferred_lft 59794sec
    inet6 fe80::f233:4f72:4556:14c2/64 scope link
        valid_lft forever preferred_lft forever
```

ip route

The **ip route** command will display your routing table:

```
$ ip route
default via 192.168.0.1 dev enp0s25 proto static metric 100
169.254.0.0/16 dev enp0s25 scope link metric 1000
192.168.0.0/24 dev enp0s25 proto kernel scope link src 192.168.0.20 metric 100
```

kill, pkill and killall

Unix and Linux systems offer a convenient selection of commands for terminating processes regardless of why you want them dead. You can kill by process ID or by name. You can kill singly or a group at a time. In any case, the various kill commands are at your disposal, and you should be ready to use them as needed. Examples include:

```
$ kill 1234
$ pkill bad
$ killall badproc
```

passwd

While the **passwd** command is probably an obvious one when it comes to system security, it's also one that shouldn't be omitted from any security-essentials list. Having a reasonable policy for password changes, especially as users come and go or change roles, is important.

The **passwd** command isn't just used to change passwords, however. You can also use it with **sudo** privilege to change other users' passwords, lock/unlock or expire accounts, check account status and change the settings that determine when a password expires or time password warnings.

Check the man page (man passwd) for details and use commands like these:

```
$ sudo passwd nemo      <== change nemo's password
$ sudo passwd -e dory   <== expire dory's password (forces her to reset it)
$ sudo passwd -i shark  <== disable shark's account
```

pwck

The **pwck** command does something of a sanity check on your **/etc/passwd** and **/etc/shadow** files – making sure required fields are present, files and directories exist, etc.

```
$ sudo pwck
user 'squash': directory '/home/squash' does not exist
user 'squash': program '/bin/bsh' does not exist
```

setfacl and getfacl

Don't let the easy display of rwxr-x--- type permissions lull you into imagining that this is all there is to file permissions on Linux systems. With **setfacl** and **getfacl** commands, you can give someone who isn't a file's owner and isn't a member of the associated group (and you don't want them to be) access to a file. Say you want Nemo to have read access to a file outlining your ufw setup instructions file but nothing more. Use a command like this to modify the access control list (ACL) for the file:

```
$ setfacl -m u:nemo:r ufw-setup
```

The **getfacl** command will then display that the change was made:

```
$ getfacl ufw-setup
# file: ufw-setup
# owner: shs
# group: shs
user::rwx
user:nemo:r--      <===
group::rw-                #effective:r--
mask::r--
other::---
```

sestatus and apparmor

The **sestatus** and **apparmor** commands can display the status of SELinux and apparmor tools that provide isolation between applications using mandatory access control. If you are using one or the other of these tools, you should know how to display its status.

sestatus

```
$ sudo sestatus
SELinux status:                 enabled
```

```
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

apparmor

```
$ sudo apparmor_status
apparmor module is loaded.
18 profiles are loaded.
18 profiles are in enforce mode.
   /sbin/dhclient
   /usr/bin/evince
   /usr/bin/evince-previewer
   /usr/bin/evince-previewer//sanitized_helper
   /usr/bin/evince-thumbnailer
   /usr/bin/evince-thumbnailer//sanitized_helper
   /usr/bin/evince//sanitized_helper
   /usr/lib/NetworkManager/nm-dhcp-client.action
   /usr/lib/NetworkManager/nm-dhcp-helper
   /usr/lib/connman/scripts/dhclient-script
   /usr/lib/cups/backend/cups-pdf
   /usr/lib/snapd/snap-confine
   /usr/lib/snapd/snap-confine//mount-namespace-capture-helper
   /usr/sbin/cups-browsed
   /usr/sbin/cupsd
   /usr/sbin/cupsd//third_party
   /usr/sbin/ippusbxd
   /usr/sbin/tcpdump
0 profiles are in complain mode.
3 processes have profiles defined.
3 processes are in enforce mode.
   /sbin/dhclient (705)
   /usr/sbin/cups-browsed (30173)
   /usr/sbin/cupsd (26828)
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

You should also know how to start and stop these tools.

```
$ sudo /etc/init.d/apparmor start
$ sudo /etc/init.d/apparmor stop
$ sudo /etc/init.d/apparmor restart
```

And, for **SELinux**, what the various modes represent:

```
enforcing -- SELinux security policy is enforced
permissive -- SELinux prints warnings instead of enforcing
disabled -- SELinux is fully disabled
```

Many commands on Linux systems can help you manage security. The descriptions above are meant to introduce these commands, but not to explain everything about how they work or can be used. ■

GUENTERGUNI / GETTY IMAGES

NETWORKWORLD
FROM IDG

THE LINUX COMMAND-LINE CHEAT SHEET

COMMAND	FUNCTION
Getting familiar with your account	
pwd	Displays your current location in the file system
whoami	Displays your username – most useful if you switch users with su and need to be reminded what account you're using currently
ls	Provides a file listing. With **-a**, it also displays files with names starting with a period (e.g., .bashrc). With **-l**, it also displays file permissions, sizes and last updated date/time.
env	Displays your user environment settings (e.g., search path, history size, home directory, etc.)
echo	Repeats the text you provide or displays the value of some variable
history	Lists previously issued commands
passwd	Changes your password. Note that complexity requirements may be enforced.
Examining Files	
cat	Displays the entire contents of a text file.
more	Displays the contents of a text file one screenful at a time. Hit the spacebar to move to each additional chunk.
less	Displays the contents of a text file one screenful at a time, but in a manner that allows you to back up using the **up arrow key**.
file	Identifies files by type (e.g., ASCII text, executable, image, directory)
Managing files	
chmod	Changes file permissions (who can read it, whether it can be executed, etc.)
chown	Changes file owner
cp	Makes a copy of a file.
mv	Moves or renames a file – or does both
rm	Deletes a file or group of files
Creating and editing files	
nano	An easy-to-use text editor that requires you to move around in the file using your arrow keys and provides control sequences to locate text, save your changes, etc.
vi	A more sophisticated editor that allows you to enter commands to find and change text, make global changes, etc.
ex	A text editor designed for programmers and has both a line-oriented and visual mode

Creating and editing files (continued)

touch	Creates a file if it doesn't exist or updates its timestamp if it does
>	Creates files by directing output to them. A single **>** creates a file while **>>** appends to an existing file.
mkdir	Creates a directory
Moving around the file system	
cd	With no arguments, takes you to your home directory. The same thing would happen if you typed **cd $HOME** or **cd ~**
cd ..	Moves up **(toward /)** one directory from your current location
cd <location>	Takes you to the specified location. If the location begins with **a /**, it is taken to be relative to the root directory; otherwise it is taken as being relative to your current location. The **~**character represents your home directory.
Learning about and identifying commands	
man	Displays the manual (help) page for a specified command and (with **-k**) provides a list of commands related to a specified keyword
which	Displays the location of the executable that represents the particular command
apropos	Lists commands associated with a particular topic or keyword
Finding files	
find	Locates files based on criteria provided (file name, type, owner, permissions, size, etc.). Unless provided with a location from which to start the search, find only looks in the current directory.
locate	Locates files using the contents of the **/var/lib/mlocate/mlocate.db** which is updated by the **updatedb** command usually run through cron. No starting location is required.
Viewing running processes	
ps	Shows processes that you are running in your current login session
ps -ef	Shows all processes that are currently running on the system
pstree	Shows running processes in a hierarchical (tree-like) display that demonstrates the relationships between processes (**-h** highlights current process)
Starting, stopping and listing services	
systemctl	The **systemctl** command can start, stop, restart and reload services. Privileged access is required.
service	Lists services and indicates whether they are running

< THE LINUX COMMAND-LINE CHEAT SHEET >

COMMAND	FUNCTION
Killing processes	
kill	Terminates a running process provided you have the authority to do so
killall	Terminates all processes with the provided name
pkill	Terminates a process based on its name
Identifying your OS release	
uname	Displays information on OS release in a single line of text
lsb_release	On Debian-based systems, this command displays information on the OS release including its codename and distributor ID
hostnamectl	Displays information on the system including hostname, chassis type, OS, kernel and architecture
Gauging system performance	
top	Shows running processes along with resource utilization and system performance data. Can show processes for one selected user or all users. Processes can be ordered by various criteria (CPU usage by default)
atop	Similar to top command but more oriented toward system performance than individual processes
free	Shows memory and swap usage — total, used and free
df	Display file system disk space usage
Managing users and groups	
useradd	Adds a new user account to the system. A username is mandatory. Other fields (user description, shell, initial password, etc.) can be specified. Home directory will default to **/home/username**.
userdel	Removes a user account from the system. The **-f** option runs a more forceful removal, deleting the home and other user files even if the user is still logged in.
groupadd	Adds a new user group to the system, updating the **/**etc/group.
groupdel	Removes a user group from the system
Examining network connections	
ip	Displays information on network interfaces
ss	Displays information on sockets. The **-s** option provides summary stats. The **-l** option shows listening sockets. The **-4** or **-6** options restrict output to IPv4 or IPv6 connections.
ping	Check connectivity to another system
Managing security	
visudo	The visudo command allows you to configure privileges that will allow select individuals to run certain commands with superuser authority. The command does this by making changes to the **/etc/sudoers** file.

Managing security (continued)

sudo	The sudo command is used by privileged users (as defined in the /etc/sudoersfile to run commands as root.
su	Switches to another account. This requires that you know the user's password or can use sudo and provide your own password. Using the **-** means that you also pick up the user's environment settings.
who	Shows who is logged into the system
last	Lists last logins for specified user using records from the **/var/log/wtmp** file.
ufw	Manages the firewall on Debian-based systems.
firewall-cmd	Manages the firewall (firewalld) on RHEL and related systems.
iptables	Displays firewall rules.
Setting up and running scheduled processes	
crontab	Sets up and manages scheduled processes. With the **-l** option, cron jobs are listed. With the **-e** option, cron jobs can be set up to run at selected intervals.
anacron	Allows you to run scheduled jobs on a daily basis only. If the system is powered off when a job is supposed to run, it will run when the system boots.
Updating, installing and listing applications	
apt update	On Debian-based systems, updates the list of available packages and their versions, but does not install or upgrade any packages
apt upgrade	On Debian-based systems, installs newer versions of installed packages
apt list	Lists all packages installed on Debian-based system. With — upgradableoption, it shows only those packages for which upgrades are available.
apt install	On Debian-based systems, installs requested package
yum update	On RPM-cased systems, updates all or specified packages
yum list	On RPM-based systems, lists package
yum install	On RPM-based systems, installs requested package
yum list	On RPM-based systems, lists known and installed packages
Shutting down and rebooting	
shutdown	Shuts down the system at the requested time. The **-H** option halts the system while the **-P** powers it down as well.
halt	Shuts down the system at the requested time.
poweroff	Powers down the system at the requested time.

NETWORKWORLD FROM IDG