KOÇ UNIVERSITY

COMP 350
Introduction to DevOps Tools and Techniques

Lecture 4
Permissions & Processes

Hakan Ayral

Multiuser System

- Unix (and Linux) is not only a ***multitasking*** system but also a ***multiuser*** system.
- While a typical computer will likely have only one keyboard and monitor, it can still be used by more than one user.
  - For example, if a computer is attached to a network or the Internet, remote users can log in via ssh (secure shell) and operate the computer.
  - In fact, remote users can execute graphical applications and have the graphical output appear on a remote display. The X Window System supports this as part of its basic design.
- The **multiuser capability** of Linux **is a feature that is deeply embedded into the design of the operating system**.
  - Considering the environment in which Unix was created, this makes perfect sense.
    - Before computers were “personal,” they were large, expensive, and centralized.
    - A typical university computer system, for example, consisted of a large central computer located in one building and terminals located throughout the campus, each connected to the large central computer.
    - The computer would support many users at the same time.
- In order to make this practical, a method had to be devised to protect the users from each other.
  - The actions of one user can not be allowed to crash the computer, nor could one user interfere with the files belonging to another user.

Some Commands

- id—Display user identity.
- chmod—Change a file’s mode.
- umask—Set the default file permissions.
- su—Run a shell as another user.
- sudo—Execute a command as another user.
- chown—Change a file’s owner.
- chgrp—Change a file’s group ownership.
- passwd—Change a user’s password.

Owners, Group Members, and Everybody Else

- In the Unix security model, a user *owns* files and directories.
- When a user owns a file or directory, the user has control over its access.
- Users can, in turn, belong to a ***group*** consisting of one or more users who are given access to files and directories by their owners.
- In addition to granting access to a group, an owner may also grant some set of access rights to everybody, which in Unix terms is referred to as the *world*.
- To find out information about your identity, use the ***id*** command:

User ID, group ID

```
hayral@Computer1:~$
hayral@Computer1:~$ id
uid=1000(hayral) gid=1000(hayral) groups=1000(hayral),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),114(lpadmin),134(sambashare)
hayral@Computer1:~$
```

- When user accounts are created, users are assigned a number called a ***user ID***, or ***uid***.
- This is then, for the sake of the humans, mapped to a **username**.
- The user is assigned a ***primary group ID,*** or ***gid***, and may belong to additional groups.
- User/group ID numbering may show differences from distribution to distribution:
  - Fedora starts its numbering of regular user accounts at 500, while Ubuntu starts at 1000.
  - You can see that the Ubuntu user belongs to a lot more groups.
    - This has to do with the way Ubuntu manages privileges for system devices and services.

Where is the user database stored?

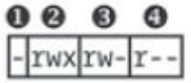
- Like many things in Linux, it comes from a couple of text files.
- User accounts are defined in the ***/etc/passwd*** file, and groups are defined in the ***/etc/group*** file.
- When user accounts and groups are created, these files are modified along with ***/etc/shadow***, which holds information about the user's password.
- For each user account, the ***/etc/passwd*** file defines the user (login) name, the uid, the gid, the account's real name, the home directory, and the login shell.
- If you examine the contents of ***/etc/passwd*** and ***/etc/group***, you will notice that besides the regular user accounts there are accounts for the superuser (uid 0) and various other system users.

Read, Write, Execute

- Access rights to files and directories are defined in terms of **read access**, **write access**, and **execution access**.
- If we look at the output of the ls command, we can get some clue as to how this is implemented:

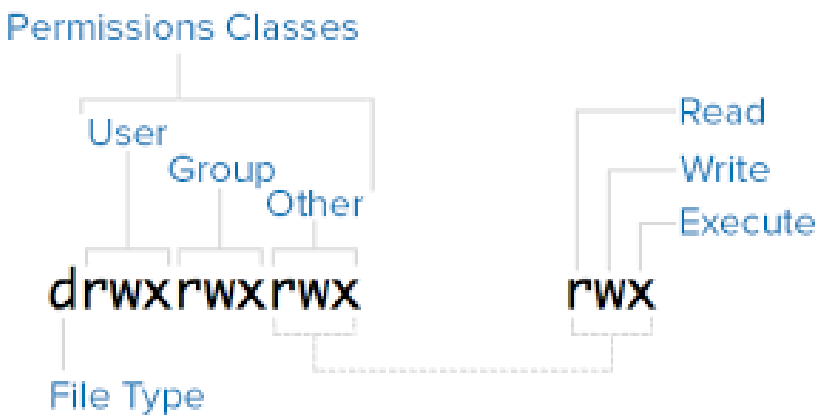
```
hayral@Computer1:~/Desktop/test$
hayral@Computer1:~/Desktop/test$ ls -l
total 44
-rw-rw-r-- 1 hayral hayral   164 Mar 10 11:18 after.txt
-rw-rw-r-- 1 hayral hayral 18868 Mar 10 11:18 before.txt
-rw-rw-r-- 1 hayral hayral 18868 Mar 10 11:17 ls-usrbin.txt
hayral@Computer1:~/Desktop/test$
```

- The first 10 characters of the listing are the *file attributes* (see Figure 9-1).
  - The first of these characters is the *file type*.
    - Table 9-1 lists the file types you are most likely to see (there are other, less common types too).
  - The remaining nine characters of the file attributes, called the *file mode,* represent the read, write, and execute permissions for the file's owner, the file's group owner, and everybody else.

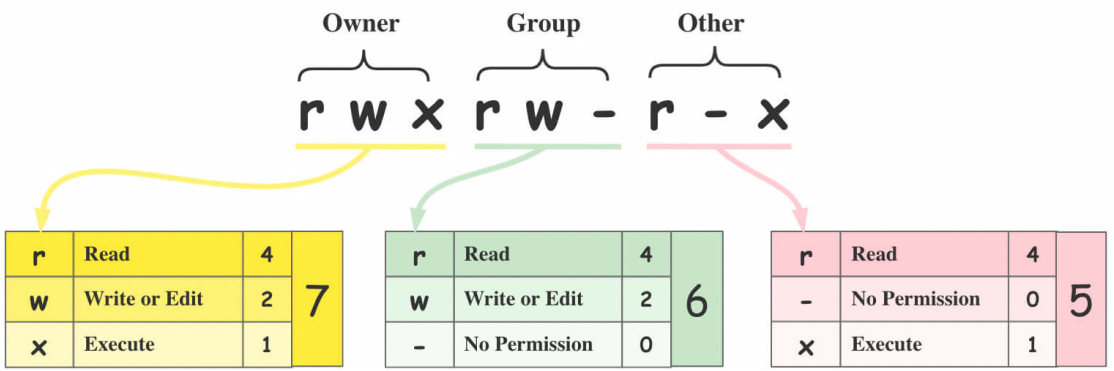


❶ File type (see Table 9-1)
❷ Owner permissions (see Table 9-2)
❸ Group permissions (see Table 9-2)
❹ World permissions (see Table 9-2)

Figure 9-1: Breakdown of file attributes



Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



File Types and Permissions

Table 9-1: File Types

Attribute	File Type
-	A regular file.
d	A directory.
l	A symbolic link. Notice that with symbolic links, the remaining file attributes are always rwxrwxrwx and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A *character special file*. This file type refers to a device that handles data as a stream of bytes, such as a terminal or modem.
b	A *block special file*. This file type refers to a device that handles data in blocks, such as a hard drive or CD-ROM drive.

Table 9-2: Permission Attributes

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered; e.g., `cd directory`.

Table 9-3: Permission Attribute Examples

File Attributes	Meaning
-rwx------	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
-rw-------	A regular file that is readable and writable by the file's owner. No one else has any access.
-rw-r--r--	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world readable.
-rwxr-xr-x	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
-rw-rw----	A regular file that is readable and writable by the file's owner and members of the file's owner group only.
lrwxrwxrwx	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
drwxrwx---	A directory. The owner and the members of the owner group may enter the directory and create, rename, and remove files within the directory.
drwxr-x---	A directory. The owner may enter the directory and create, rename, and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete, or rename files.

chmod—Change File Mode

- To change the mode (permissions) of a file or directory, the ***chmod*** command is used.
- Only the file's owner or the superuser can change the mode of a file or directory.
- ***chmod*** supports two distinct ways of specifying mode changes: octal number representation and symbolic representation.
- We will cover octal number representation first.
  - **Octal Representation**
    - With octal notation we use octal numbers to set the pattern of desired permissions.
  - Since each digit in an octal number represents three binary digits, this maps nicely to the scheme used to store the file mode. Table 9-4 shows what we mean.

Table 9-4: File Modes in Binary and Octal

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

chmod—Change File Mode

- By using three octal digits, we can set the file mode for the owner, group owner, and world.

```
hayral@Computer1:~/Desktop/test$
hayral@Computer1:~/Desktop/test$ ls -l
total 8
-rw-rw-r-- 1 hayral hayral 22 Mar 14 21:06 data2.txt
-rw-rw-r-- 1 hayral hayral 22 Mar 14 21:08 data.txt
hayral@Computer1:~/Desktop/test$ chmod 600 data.txt
hayral@Computer1:~/Desktop/test$ ls -l
total 8
-rw-rw-r-- 1 hayral hayral 22 Mar 14 21:06 data2.txt
-rw------- 1 hayral hayral 22 Mar 14 21:08 data.txt
hayral@Computer1:~/Desktop/test$
```

- By passing the argument 600, we can set the permissions of the owner to read and write while removing all permissions from the group owner and world.
- Though remembering the octal-to-binary mapping may seem inconvenient, you will usually have to use only a few common ones:

7 (rwx), 6 (rw-), 5 (r-x), 4 (r--), and 0 (---)

chmod—Change File Mode

- **Symbolic Representation**
- chmod also supports a symbolic notation for specifying file modes.
- Symbolic notation is divided into three parts: whom the change will affect, which operation will be performed, and which permission will be set.
- To specify who is affected, a combination of the characters ***u***, ***g***, ***o***, and ***a*** is used, as shown in Table 9-5.
- If no character is specified, *all* will be assumed.
- The operation may be "**+**" indicating that a permission is to be added, "**-**" indicating that a permission is to be taken away, or "=" indicating that only the specified permissions are to be applied and that all others are to be removed.
- Permissions are specified with the **r**, **w**, and **x** characters.
- Table 9-6 lists some examples of symbolic notation.

Table 9-5: chmod Symbolic Notation

Symbol	Meaning
u	Short for *user* but means the file or directory owner.
g	Group owner.
o	Short for *others* but means world.
a	Short for *all*; the combination of *u*, *g*, and *o*.

Table 9-6: chmod Symbolic Notation Examples

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. Equivalent to a+x.
o-rw	Remove the read and write permissions from anyone besides the owner and group owner.
go=rw	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or world previously had execute permissions, remove them.
u+x,go=rx	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

umask—Set Default Permissions

- The ***umask*** command controls the default permissions given to a file when it is created.
- It uses octal notation to express a ***mask*** of bits to be removed from a file's mode attributes.

```
hayral@Computer1:~/Desktop/test$
hayral@Computer1:~/Desktop/test$ umask
0002
hayral@Computer1:~/Desktop/test$ touch test.txt
hayral@Computer1:~/Desktop/test$ ls -l
total 0
-rw-rw-r-- 1 hayral hayral 0 Mar 14 22:50 test.txt
hayral@Computer1:~/Desktop/test$
```

- On the example above we run the ***umask*** command without an argument to see the current value.
- It shows the value 0002, which is the octal representation of our mask.,
- We then create a new file *test.txt* and observe its permissions.
  - We see that both the owner and group get read and write permissions (rw),
  - everyone else gets only read permission (r).
  - World does not have write permission because of the value of the mask.

umask—Set Default Permissions

- Let's repeat the example, this time setting the mask ourselves:

```
hayral@Computer1:~/Desktop/test$
hayral@Computer1:~/Desktop/test$ umask 0000
hayral@Computer1:~/Desktop/test$ touch test2.txt
hayral@Computer1:~/Desktop/test$ ls -l
total 0
-rw-rw-rw- 1 hayral hayral 0 Mar 14 22:54 test2.txt
-rw-rw-r-- 1 hayral hayral 0 Mar 14 22:50 test.txt
hayral@Computer1:~/Desktop/test$
```

- When we set the mask to 0000 (effectively turning it off), we see that the file is now world writable.
- To understand how this works, let's look at octal numbers again.
- If we expand the mask into binary and then compare it to the attributes, we can see what happens:
  - observe that where the 1 appears in our mask, an attribute was removed—in this case, the world write permission.
  - That's what the mask does: everywhere a 1 appears in the binary value of the mask, an attribute is unset.

Original file mode	--- rw- rw- rw-
Mask	000 000 000 010
Result	--- rw- rw- r--

Changing identities

- At various times, we may find it necessary to take on the identity of another user.
- Often we want to gain superuser privileges to carry out some administrative task, but it is also possible to “become” another regular user to perform such tasks as testing an account.
- There are three ways to take on an alternate identity:
  - Log out and log back in as the alternate user.
  - Use the ***su*** command.
  - Use the ***sudo*** command.
- We will skip the first technique because we know how to do it and it lacks the convenience of the other two.
- From within your own shell session, the ***su*** command allows you to assume the identity of another user and either start a new shell session with that user’s ID or issue a single command as that user.
- The ***sudo*** command allows an administrator to set up a configuration file called ***/etc/sudoers*** and define specific commands that particular users are permitted to execute under an assumed identity.

su—Run a Shell with Substitute User and Group IDs

- The su command is used to start a shell as another user.
- The command syntax looks like this:

su [-[l]] [*user*]

- If the -l option is included, the resulting shell session is a *login shell* for the specified user.
  - This means that the user's environment is loaded and the working directory is changed to the user's home directory. (This is usually what we want for ***su***)
- If the user is not specified, the superuser is assumed.
- Notice that (strangely) the -l may be abbreviated as -, which is how it is most often used.

sudo—Execute a Command as Another User

- The sudo command is like su in many ways but has some important additional capabilities.
- The administrator can configure sudo to allow an ordinary user to execute commands as a different user (usually the superuser) in a very controlled way.
- In particular, a user may be restricted to one or more specific commands and no others.
- Another important difference is that the use of sudo does not require access to the superuser's password.
- To authenticate using sudo, the user enters his own password.
- After entering the command, we are prompted for our password (not the superuser's), and once the authentication is complete, the specified command is carried out.
- One important difference between ***su*** and ***sudo*** is that ***sudo*** does not start a new shell, nor does it load another user's environment.

chown—Change File Owner and Group

- The chown command is used to change the owner and group owner of a file or directory.
- Superuser privileges are required to use this command.
- The syntax of chown looks like this:

chown [*owner*][:[*group*]] *file*...

- chown can change the file owner and/or the file group owner depending on the first argument of the command.
- Table 9-7 lists some examples:

Table 9-7: chown Argument Examples

Argument	Results
`bob`	Changes the ownership of the file from its current owner to user *bob*.
`bob:users`	Changes the ownership of the file from its current owner to user *bob* and changes the file group owner to group *users*.
`:admins`	Changes the group owner to the group *admins*. The file owner is unchanged.
`bob:`	Change the file owner from the current owner to user *bob* and changes the group owner to the login group of user *bob*.

chown—Change File Owner and Group

- Let's say that we have two users: ***janet***, who has access to superuser privileges, and ***tony***, who does not.
- User ***janet*** wants to copy a file from her home directory to the home directory of user ***tony***.
- Since user ***janet*** wants ***tony*** to be able to edit the file, ***janet*** changes the ownership of the copied file from ***janet*** to ***tony***:

```
[janet@linuxbox ~]$ sudo cp myfile.txt ~tony
Password:
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
 -rw-r--r-- 1 root  root  8031 2012-03-20 14:30 /home/tony/myfile.txt
[janet@linuxbox ~]$ sudo chown tony: ~tony/myfile.txt
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
 -rw-r--r-- 1 tony  tony  8031 2012-03-20 14:30 /home/tony/myfile.txt
```

- Here we see user ***janet*** copy the file from her directory to the home directory of user ***tony***.
- Next, ***janet*** changes the ownership of the file from ***root*** (a result of using sudo) to ***tony***.
- Using the trailing colon in the first argument, ***janet*** also changed the group ownership of the file to the login group of ***tony***, which happens to be group ***tony***.
- Notice that after the first use of sudo, ***janet*** was not prompted for her password?
  - This is because sudo, in most configurations, "trusts" you for several minutes (until its timer runs out).

chgrp—Change Group Ownership

- In older versions of Unix, the ***chown*** command changed only file ownership, not group ownership.
- For that purpose a separate command, ***chgrp***, was used.
- It works much the same way as ***chown***, except for being more limited.
- You are not likely to use this command often.

Changing user password

- The last topic we'll cover in this chapter is setting passwords for yourself (and for other users if you have access to superuser privileges).
- To set or change a password, the passwd command is used. The command syntax looks like this:

passwd [*user*]

- To change your password, just enter the passwd command.
  - You will be prompted for your old password and your new password.
- The ***passwd*** command will try to enforce use of "strong" passwords.
  - This means it will refuse to accept passwords that are too short, are too similar to previous passwords, are dictionary words, or are too easily guessed.
- If you have superuser privileges, you can specify a username as an argument to the ***passwd*** command to set the password for another user.
- Other options are available to the superuser to allow account locking, password expiration, and so on.

Processes

- Modern operating systems are usually *multitasking*, meaning that they create the illusion of doing more than one thing at once by rapidly switching from one executing program to another.
  - Even if you have a multicore processor the number of processes are always higher than the number of cores; so the processes share available cores over very small time intervals which gives the illusion of them working simultaneously without interruption.
- The Linux kernel manages this through the use of *processes*.
- Processes are how Linux organizes the different programs waiting for their turn at the CPU.
- Sometimes a computer will become sluggish, or an application will stop responding.
  - In this chapter, we will look at some of the tools available at the command line that let us examine what programs are doing and how to terminate processes that are misbehaving.

Processes and related commands

- ps—Report a snapshot of current processes.
- top—Display tasks.
- jobs—List active jobs.
- bg—Place a job in the background.
- fg—Place a job in the foreground.
- kill—Send a signal to a process.
- killall—Kill processes by name.
- shutdown—Shut down or reboot the system.

How a Process Works

- When a system starts up, the kernel initiates a few of its own activities as processes and launches a program called **init**.
- **init**, in turn, runs a series of shell scripts (located in ***/etc***) called *init scripts*, which start all the system services.
- Many of these services are implemented as ***daemon programs***, programs that just sit in the background and do their thing without having any user interface.
- So even if we are not logged in, the system is at least a little busy performing routine stuff.
- The fact that a program can launch other programs is expressed in the process scheme as a ***parent process*** producing a ***child process***.
- The kernel maintains information about each process to help keep things organized.
- For example, each process is assigned a number called a ***process ID (PID)***.
- PIDs are assigned in ascending order, with **init** always getting PID 1.
- The kernel also keeps track of the memory assigned to each process, as well as the processes' readiness to resume execution.
- Like files, processes also have owners and user IDs, effective user IDs, and so on.

- Typical process tree from Linux Mint

Process Name	ID	User	Virtual Memory	Resident Memor	Shared Memory	% CPU	CPU Time	Memory
systemd	1	root	99,6 MiB	9,6 MiB	7,2 MiB	0	0:03.99	2,4 MiB
NetworkManager	447	root	333,2 MiB	18,7 MiB	16,2 MiB	0	0:01.61	2,5 MiB
networkd-dispatcher	459	root	41,1 MiB	18,0 MiB	11,4 MiB	0	0:00.48	6,6 MiB
colord	1147	colord	245,7 MiB	14,1 MiB	9,0 MiB	0	0:00.41	5,1 MiB
csd-printer	1218	hayral	336,8 MiB	13,8 MiB	11,9 MiB	0	0:00.10	1,8 MiB
systemd-journald	273	root	66,6 MiB	12,0 MiB	10,6 MiB	0	0:01.21	1,4 MiB
udisksd	470	root	386,2 MiB	11,9 MiB	9,8 MiB	0	0:00.41	2,1 MiB
systemd-resolved	423	system	23,8 MiB	11,7 MiB	8,0 MiB	0	0:00.46	3,6 MiB
cups-browsed	527	root	176,2 MiB	11,3 MiB	9,7 MiB	0	0:00.21	1,6 MiB
polkitd	461	root	233,5 MiB	9,9 MiB	7,0 MiB	0	0:00.70	2,9 MiB
upowerd	1127	root	248,9 MiB	9,7 MiB	8,5 MiB	0	0:00.34	1,1 MiB
ModemManager	551	root	234,4 MiB	9,4 MiB	7,8 MiB	0	0:00.43	1,6 MiB
accounts-daemon	440	root	238,9 MiB	8,1 MiB	7,1 MiB	0	0:00.46	1020,0 KiB
systemd	889	hayral	18,3 MiB	7,8 MiB	7,0 MiB	0	0:00.49	848,0 KiB
gnome-terminal-server	1471	hayral	455,7 MiB	38,5 MiB	30,1 MiB	0	0:02.45	8,4 MiB
bash	1479	hayral	13,2 MiB	5,3 MiB	3,8 MiB	0	0:00.28	1,5 MiB
goa-daemon	1231	hayral	536,2 MiB	32,1 MiB	26,2 MiB	0	0:00.25	5,8 MiB
evolution-calendar-factory	1366	hayral	829,9 MiB	28,5 MiB	24,8 MiB	0	0:00.35	3,7 MiB
evolution-addressbook-facl	1385	hayral	660,3 MiB	27,7 MiB	24,2 MiB	0	0:00.26	3,4 MiB
evolution-source-registry	1348	hayral	456,0 MiB	24,4 MiB	20,9 MiB	0	0:00.27	3,5 MiB
pulseaudio	898	hayral	1,1 GiB	14,8 MiB	11,0 MiB	0	0:00.76	3,8 MiB
gvfs-udisks2-volume-monit	1180	hayral	384,8 MiB	10,0 MiB	8,4 MiB	0	0:00.24	1,7 MiB
goa-identity-service	1241	hayral	313,8 MiB	9,2 MiB	8,0 MiB	0	0:00.10	1,3 MiB
gvfs-afc-volume-monitor	1215	hayral	312,0 MiB	8,7 MiB	7,7 MiB	0	0:00.27	996,0 KiB
at-spi-bus-launcher	1063	hayral	302,5 MiB	7,9 MiB	7,1 MiB	0	0:00.08	896,0 KiB
dbus-daemon	1068	hayral	7,3 MiB	4,1 MiB	3,6 MiB	0	0:00.21	472,0 KiB
gvfsd	912	hayral	236,7 MiB	7,6 MiB	6,7 MiB	0	0:00.21	912,0 KiB
gvfsd-trash	1439	hayral	312,6 MiB	10,0 MiB	8,8 MiB	0	0:00.08	1,2 MiB
gvfs-gphoto2-volume-moni	1250	hayral	235,1 MiB	6,7 MiB	6,0 MiB	0	0:00.07	688,0 KiB
gvfsd-fuse	917	hayral	373,1 MiB	6,6 MiB	5,8 MiB	0	0:00.07	868,0 KiB
gvfsd-metadata	1448	hayral	161,2 MiB	6,6 MiB	5,9 MiB	0	0:00.08	648,0 KiB
gvfs-mtp-volume-monitor	1243	hayral	232,8 MiB	6,3 MiB	5,8 MiB	0	0:00.06	564,0 KiB
at-spi2-registryd	1071	hayral	159,1 MiB	6,3 MiB	5,7 MiB	0	0:00.41	640,0 KiB

Viewing Processes with *ps*

- The most commonly used command to view processes (there are several) is ***ps***.
- The ***ps*** program has a lot of options, but in it simplest form it is used like this:

```
hayral@Computer1:~$
hayral@Computer1:~$ ps
    PID TTY          TIME CMD
   1479 pts/0    00:00:00 bash
   1657 pts/0    00:00:00 ps
hayral@Computer1:~$
```

- The result in this example lists two processes: process 1479 and process 1657, which are ***bash*** and ***ps*** respectively.
- As we can see, by default ***ps*** doesn't show us very much, just the processes associated with the current terminal session.
- To see more, we need to add some options, but before we do that, let's look at the other fields produced by ***ps***.
  - TTY is short for *teletype* and refers to the *controlling terminal* for the process. (very old terminology due to historical reasons)
  - The TIME field is the amount of CPU time consumed by the process.
    - As we can see, neither process makes the computer work very hard.

?

Now, this is a teletype

- A **teleprinter** (**teletypewriter**, **teletype** or **TTY**) is an electromechanical device that can be used to send and receive typed messages through various communications channels, in both point-to-point and point-to-multipoint configurations.
- Initially they were used in telegraphy, which developed in the late 1830s and 1840s as the first use of electrical engineering, though teleprinters were not used for telegraphy until 1887 at the earliest.
- The machines were adapted to provide a user interface to early mainframe computers and minicomputers, sending typed data to the computer and printing the response.
- Some models could also be used to create punched tape for data storage (either from typed input or from data received from a remote source) and to read back such tape for local printing or transmission.

source: https://en.wikipedia.org/wiki/Teletype_Model_37 and https://en.wikipedia.org/wiki/Teleprinter

Now, this is a teletype

Siemens t37h (1933) without cover

A Teletype Model 33 ASR with paper tape reader and punch, as used for early modem-based computing

source: https://en.wikipedia.org/wiki/Teleprinter

Viewing Processes with *ps*

- Adding the x option (note that there is no leading dash) tells ps to show all of our processes regardless of what terminal (if any) they are controlled by.
- The presence of a ? in the TTY column indicates no controlling terminal.
- Using this option, we see a list of every process that we own.
- Since the system is running a lot of processes, ***ps*** produces a long list.
- It is often helpful to pipe the output from ***ps*** into ***less*** for easier viewing.
- Some option combinations also produce long lines of output, so maximizing the terminal emulator window may be a good idea, too.

```
hayral@Computer1:~$ ps x
    PID TTY      STAT   TIME COMMAND
    889 ?        Ss     0:00 /lib/systemd/systemd --user
    890 ?        S      0:00 (sd-pam)
    898 ?        S<sl   0:00 /usr/bin/pulseaudio --daemonize=no --log-target=journal
    899 ?        Ssl    0:01 cinnamon-session --session cinnamon
    908 ?        Ss     0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofo
    912 ?        Ssl    0:00 /usr/libexec/gvfsd
    917 ?        Sl     0:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_wr
    940 ?        S      0:00 /usr/bin/VBoxClient --clipboard
    941 ?        Sl     0:00 /usr/bin/VBoxClient --clipboard
    951 ?        S      0:00 /usr/bin/VBoxClient --seamless
    953 ?        Sl     0:00 /usr/bin/VBoxClient --seamless
    959 ?        S      0:00 /usr/bin/VBoxClient --draganddrop
    960 ?        Sl     0:11 /usr/bin/VBoxClient --draganddrop
    963 ?        S      0:00 /usr/bin/VBoxClient --vmsvga
    966 ?        Sl     0:00 /usr/bin/VBoxClient --vmsvga
   1051 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/im-launch cinnamon-session-c
   1063 ?        Ssl    0:00 /usr/libexec/at-spi-bus-launcher
   1068 ?        S      0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/a
   1071 ?        Sl     0:00 /usr/libexec/at-spi2-registryd --use-gnome-session
   1078 ?        SLl    0:00 /usr/bin/gnome-keyring-daemon --start --components=ssh
   1088 ?        Sl     0:00 /usr/libexec/csd-background
   1090 ?        Sl     0:01 /usr/libexec/csd-power
   1092 ?        Sl     0:00 /usr/libexec/csd-xrandr
   1093 ?        Sl     0:00 /usr/libexec/csd-color
   1098 ?        Sl     0:00 /usr/libexec/csd-xsettings
   1100 ?        Sl     0:00 /usr/libexec/csd-mouse
   1103 ?        Sl     0:00 /usr/libexec/csd-wacom
   1104 ?        Sl     0:00 /usr/libexec/csd-cursor
   1105 ?        Sl     0:00 /usr/libexec/csd-automount
   1106 ?        Sl     0:00 /usr/libexec/csd-orientation
   1109 ?        Sl     0:00 /usr/libexec/csd-a11y-keyboard
   1114 ?        Sl     0:00 /usr/libexec/csd-housekeeping
   1117 ?        Sl     0:01 /usr/libexec/csd-sound
   1119 ?        Sl     0:00 /usr/libexec/csd-clipboard
   1120 ?        Sl     0:00 /usr/libexec/csd-media-keys
   1125 ?        Sl     0:00 /usr/libexec/csd-screensaver-proxy
   1128 ?        Sl     0:01 /usr/libexec/csd-keyboard
   1130 ?        Sl     0:00 /usr/libexec/csd-print-notifications
   1131 ?        Sl     0:00 /usr/libexec/csd-a11y-settings
   1145 ?        Sl     0:00 /usr/libexec/dconf-service
   1180 ?        Ssl    0:00 /usr/libexec/gvfs-udisks2-volume-monitor
   1213 ?        Sl     0:00 cinnamon-launcher
   1215 ?        Ssl    0:00 /usr/libexec/gvfs-afc-volume-monitor
   1218 ?        Sl     0:00 /usr/libexec/csd-printer
   1223 ?        Ssl    0:00 /usr/libexec/gvfs-goa-volume-monitor
```

Viewing Processes with *ps*

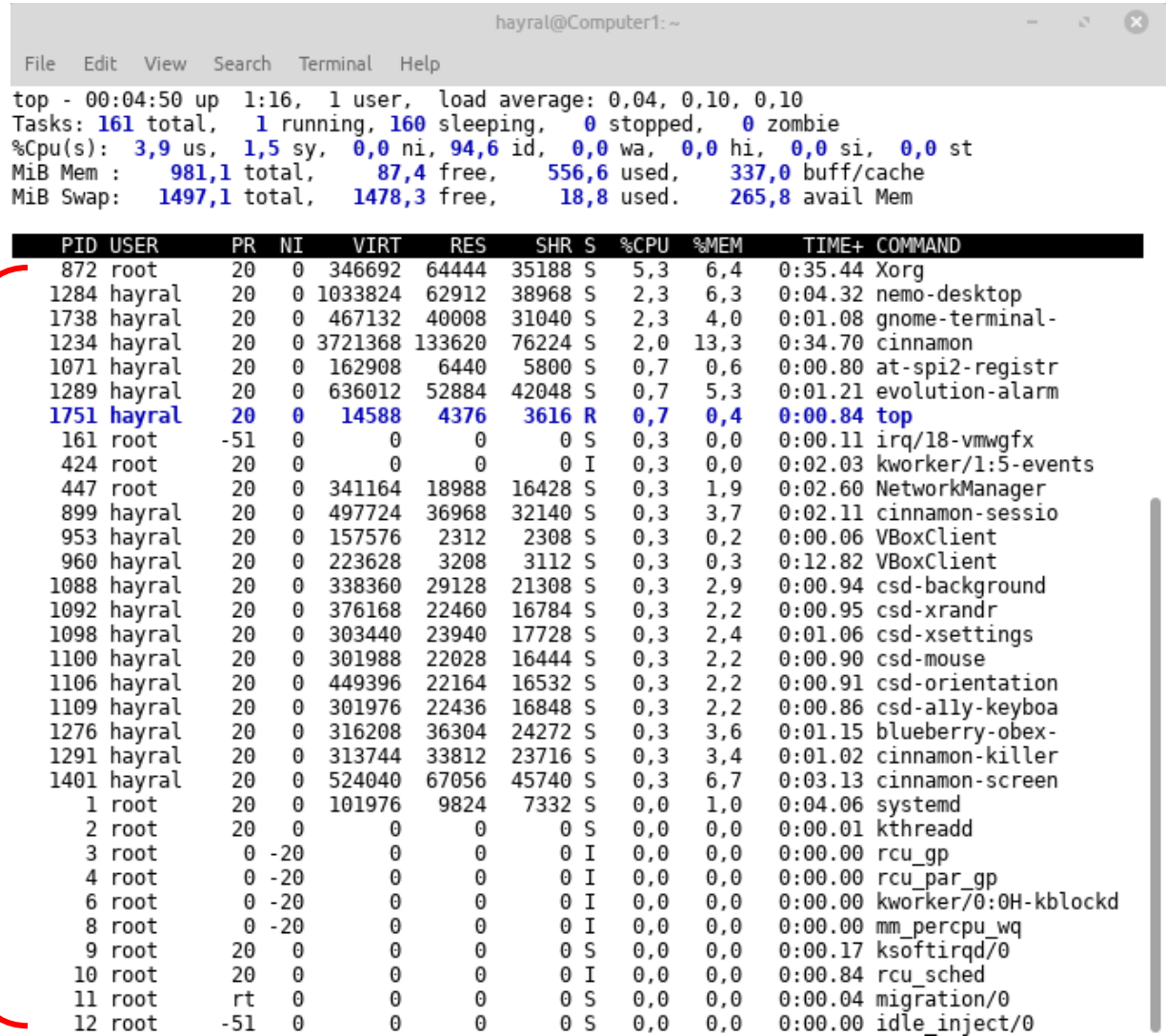
- A new column titled STAT has been added to the output when we used the parameter -x.
- STAT is short for *state* and reveals the current status of the process, as shown in Table 10-1.

Table 10-1: Process States

State	Meaning
R	Running. The process is running or ready to run.
S	Sleeping. The process is not running; rather, it is waiting for an event, such as a keystroke or network packet.
D	Uninterruptible sleep. Process is waiting for I/O such as a disk drive.
T	Stopped. Process has been instructed to stop (more on this later).
Z	A defunct or "zombie" process. This is a child process that has terminated but has not been cleaned up by its parent.
<	A high-priority process. It's possible to grant more importance to a process, giving it more time on the CPU. This property of a process is called *niceness*. A process with high priority is said to be less nice because it's taking more of the CPU's time, which leaves less for everybody else.
N	A low-priority process. A process with low priority (a nice process) will get processor time only after other processes with higher priority have been serviced.

Viewing Processes Dynamically with top

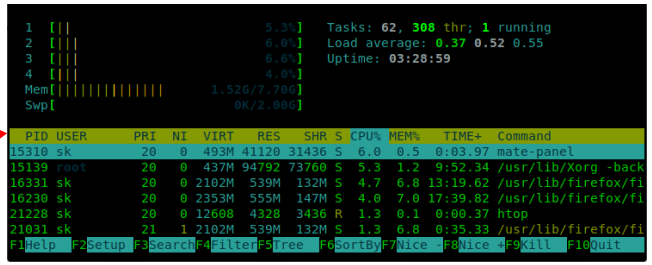
- While the ps command can reveal a lot about what the machine is doing, it provides only a snapshot of the machine's state at the moment the ps command is executed.
- To see a more dynamic view of the machine's activity, we use the top command.
- The top program displays a continuously updating (by default, every 3 seconds) display of the system processes listed in order of process activity.
- Its name comes from the fact that the top program is used to see the "top" processes on the system.
- The top display consists of two parts:
  - a system summary at the top of the display, followed by
  - a table of processes sorted by CPU activity.

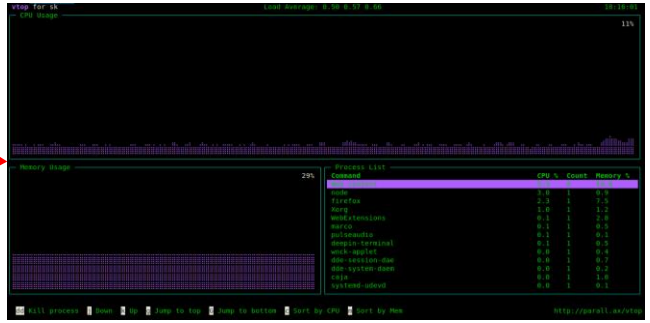


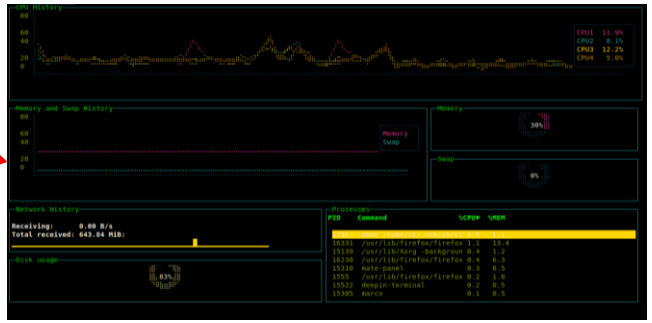
Some alternatives to default top program

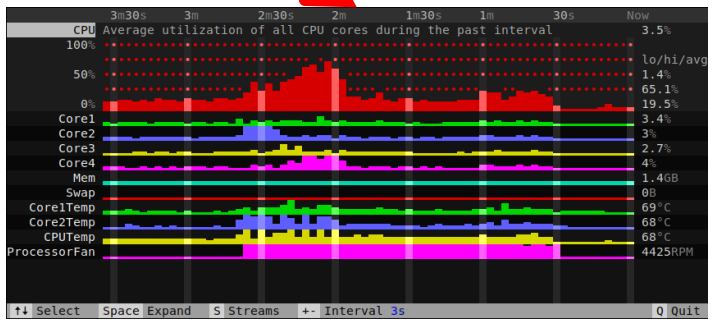
- Htop
- Vtop
- Gtop
- Gotop
- Ptop
- Hegemon
- Bashtop
- Bpytop

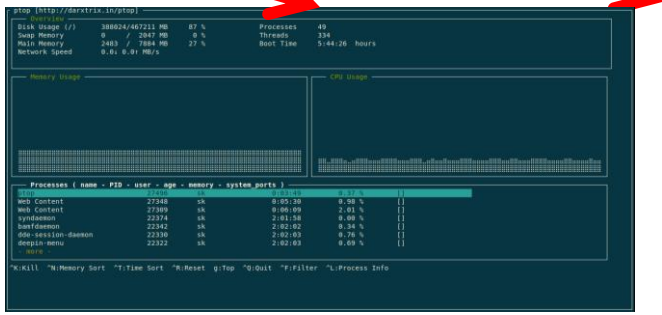
(you have to install these, they don't come preinstalled)

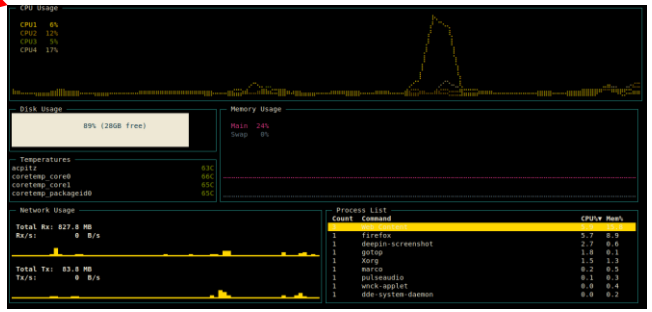






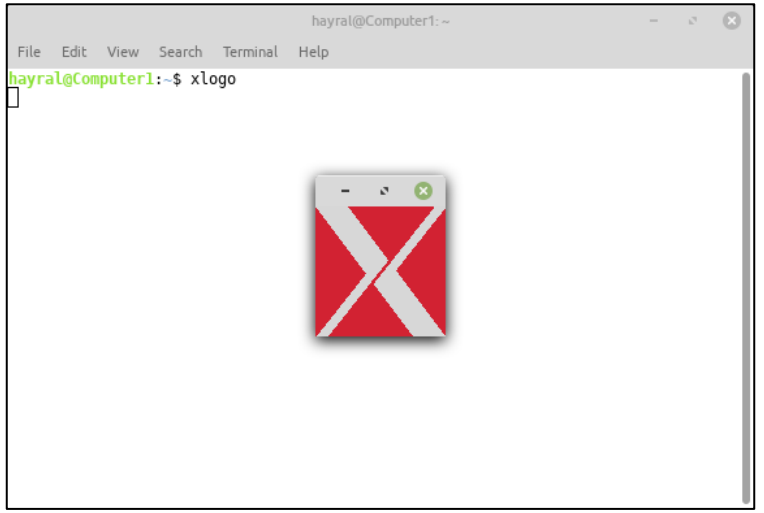






Controlling Processes

- For our experiments, we're going to use a little program called `xlogo`.
- The xlogo program is a sample program supplied with the X Window System (the underlying engine that makes the graphics on our display go), which simply displays a resizable window containing the X logo.
- After we enter the command, a small window containing the logo should appear on the screen.
- On some systems, `xlogo` may print a warning message, but it may be safely ignored.
- We can verify that `xlogo` is running by resizing its window. If the logo is redrawn in the new size, the program is running.
- Notice how our shell prompt has not returned?
  - This is because the shell is waiting for the program to finish.
  - If we close the `xlogo` window, the prompt returns.



Interrupting a Process

- Let's observe what happens when we run `xlogo` again.
- First, enter the `xlogo` command and verify that the program is running.
- Next, return to the terminal window and press CTRL-C.
- In a terminal, pressing CTRL-C ***interrupts*** a program.
- This means that we politely asked the program to terminate.
- After we pressed CTRL-C, the `xlogo` window closed and the shell prompt returned.
- Many (but not all) command-line programs can be interrupted by using this technique.

Putting a Process in the Background

- Let's say we wanted to get the shell prompt back without terminating the xlogo program.
- We'll do this by placing the program in the ***background***.
- Think of the terminal as having a ***foreground*** (with stuff visible on the surface, like the shell prompt) and a background (with hidden stuff below the surface).
- To launch a program so that it is immediately placed in the background, we follow the command with an ampersand character (&).
- After the command was entered, the xlogo window appears and the shell prompt returns, but some numbers are printed too.
  - This message is part of a shell feature called ***job control***.
  - With this message, the shell is telling us that we have started job number 1 ([1]) and that it has PID 1677.
  - If we run ***ps***, we can see our process:

```
hayral@Computer1:~$ xlogo&
[1] 1677
hayral@Computer1:~$ ps
    PID TTY          TIME CMD
   1559 pts/0    00:00:00 bash
   1677 pts/0    00:00:00 xlogo
   1681 pts/0    00:00:00 ps
hayral@Computer1:~$
```

Putting a Process in the Background

- The shell's job control facility also gives us a way to list the jobs that have been launched from our terminal.
- Using the jobs command, we can see the following list:

```
hayral@Computer1:~$
hayral@Computer1:~$ jobs
[1]+  Running                 xlogo &
hayral@Computer1:~$
```

- The results show that we have one job, numbered 1, that it is running, and that the command was **xlogo &**.

Returning a Process to the Foreground

- A process in the background is immune from keyboard input, including any attempt to interrupt it with a CTRL-C.
- To return a process to the foreground, use the ***fg*** command, as in this example:

```
hayral@Computer1:~$
hayral@Computer1:~$ jobs
[1]+  Running                 xlogo &
hayral@Computer1:~$ fg %1
xlogo
```

- The command ***fg*** followed by a percent sign and the job number (called a ***jobspec***) does the trick. If we have only one background job, the jobspec is optional.
- To terminate xlogo, type CTRL-C.

Stopping (Pausing) a Process

- Sometimes we'll want to stop a process without terminating it.
- This is often done to allow a foreground process to be moved to the background.
- To stop a foreground process, type CTRL-Z.
- Let's try it. At the command prompt, type xlogo, press the ENTER key, and then type CTRL-Z:

```
hayral@Computer1:~$
hayral@Computer1:~$ xlogo
^Z
[1]+  Stopped                 xlogo
hayral@Computer1:~$
```

- After stopping xlogo, we can verify that the program has stopped by attempting to resize the xlogo window.
  - We will see that it appears quite dead.

Stopping (Pausing) a Process

- We can either restore the program to the foreground, using the ***fg*** command, or move the program to the background with the ***bg*** command:

```
hayral@Computer1:~$
hayral@Computer1:~$ xlogo
^Z
[1]+  Stopped                 xlogo
hayral@Computer1:~$ bg %1
[1]+ xlogo &
hayral@Computer1:~$
```

- As with the ***fg*** command, the jobspec is optional for ***bg*** too, if there is only one job.
- Moving a process from the foreground to the background is handy if we launch a graphical program from the command but forget to place it in the background by appending the trailing &.

Signals

- The ***kill*** command is used to “kill” (terminate) processes.
- This allows us to end the execution of a program that is behaving badly or otherwise refuses to terminate on its own.
- Here’s an example:

```
hayral@Computer1:~$
hayral@Computer1:~$ xlogo &
[1] 1691
hayral@Computer1:~$ kill 1691
hayral@Computer1:~$
[1]+  Terminated              xlogo
hayral@Computer1:~$
```

- We first launch xlogo in the background.
- The shell prints the jobspec and the PID of the background process.
- Next, we use the kill command and specify the PID of the process we want to terminate.
- We could also have specified the process using a jobspec (for example, %1) instead of a PID.

Signals

- While this is all very straightforward, there is more to it.
- The kill command doesn't exactly "kill" processes; rather it sends them ***signals***.
- Signals are one of several ways that the operating system communicates with programs.
- We have already seen signals in action with the use of CTRL-C and CTRL-Z.
- When the terminal receives one of these keystrokes, it sends a signal to the program in the foreground.
  - In the case of CTRL-C, a signal called **INT** (Interrupt) is sent;
  - with CTRL-Z, a signal called **TSTP** (Terminal Stop) is sent.
- Programs, in turn, "listen" for signals and may act upon them as they are received.
- The fact that a program can listen and act upon signals allows it to do things like save work in progress when it is sent a termination signal.

Sending Signals to Processes with kill

- The most common syntax for the kill command looks like this:

```
kill [-signal] PID...
```

- If no signal is specified on the command line, then the TERM (Terminate) signal is sent by default.
- The kill command is most often used to send the signals shown in Table 10-4.

Table 10-4: Common Signals

Number	Name	Meaning
1	HUP	Hang up. This is a vestige of the good old days when terminals were attached to remote computers with phone lines and modems. The signal is used to indicate to programs that the controlling terminal has "hung up." The effect of this signal can be demonstrated by closing a terminal session. The foreground program running on the terminal will be sent the signal and will terminate. This signal is also used by many daemon programs to cause a reinitialization. This means that when a daemon is sent this signal, it will restart and reread its configuration file. The Apache web server is an example of a daemon that uses the HUP signal in this way.
2	INT	Interrupt. Performs the same function as the CTRL-C key sent from the terminal. It will usually terminate a program.
9	KILL	Kill. This signal is special. Whereas programs may choose to handle signals sent to them in different ways, including by ignoring them altogether, the KILL signal is never actually sent to the target program. Rather, the kernel immediately terminates the process. When a process is terminated in this manner, it is given no opportunity to "clean up" after itself or save its work. For this reason, the KILL signal should be used only as a last resort when other termination signals fail.
15	TERM	Terminate. This is the default signal sent by the kill command. If a program is still "alive" enough to receive signals, it will terminate.
18	CONT	Continue. This will restore a process after a STOP signal.
19	STOP	Stop. This signal causes a process to pause without terminating. Like the KILL signal, it is not sent to the target process, and thus it cannot be ignored.

Sending Signals to Processes with kill

- You may need to press the ENTER key a couple of times before you see the “terminated” message.
- Note that signals may be specified either by number or by name, including the name prefixed with the letters *SIG* :

```
hayral@Computer1:~$ xlogo &
[1] 1702
hayral@Computer1:~$ kill -int 1702
hayral@Computer1:~$
[1]+  Interrupt               xlogo
hayral@Computer1:~$
hayral@Computer1:~$ xlogo &
[1] 1703
hayral@Computer1:~$ kill -SIGINT 1703
hayral@Computer1:~$
[1]+  Interrupt               xlogo
hayral@Computer1:~$
```

- Remember, you can also use jobspecs in place of PIDs.
- In addition to the signals listed in Table 10-4, which are most often used with kill, other signals are frequently used by the system.
  - Table 10-5 lists the other common signals.
- A complete list of signals can be seen with the kill -l command

Table 10-5: Other Common Signals

Number	Name	Meaning
3	QUIT	Quit.
11	SEGV	Segmentation violation. This signal is sent if a program makes illegal use of memory; that is, it tried to write somewhere it was not allowed to.
20	TSTP	Terminal stop. This is the signal sent by the terminal when CTRL-Z is pressed. Unlike the STOP signal, the TSTP signal is received by the program but the program may choose to ignore it.
28	WINCH	Window change. This is a signal sent by the system when a window changes size. Some programs, like top and less, will respond to this signal by redrawing themselves to fit the new window dimensions.

```
hayral@Computer1:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
hayral@Computer1:~$
```

Sending Signals to Multiple Procs with killall

- It’s also possible to send signals to multiple processes matching a specified program or username by using the killall command. Here is the syntax:

```
killall [-u user] [-signal] name...
```

- To demonstrate, we will start a couple of instances of the xlogo program and then terminate them:

```
hayral@Computer1:~$
hayral@Computer1:~$ xlogo &
[1] 1722
hayral@Computer1:~$ xlogo &
[2] 1723
hayral@Computer1:~$ xlogo &
[3] 1724
hayral@Computer1:~$ killall xlogo
[1]   Terminated              xlogo
[2]-  Terminated              xlogo
[3]+  Terminated              xlogo
hayral@Computer1:~$
```

- Remember, as with kill, you must have superuser privileges to send signals to processes that do not belong to you.

More Process-Related Commands

- Since monitoring processes is an important system administration task, there are a lot of commands for it.
- Table 10-6 lists some to play with:

Table 10-6: Other Process-Related Commands

Command	Description
`pstree`	Outputs a process list arranged in a tree-like pattern showing the parent/child relationships between processes.
`vmstat`	Outputs a snapshot of system resource usage including memory, swap, and disk I/O. To see a continuous display, follow the command with a time delay (in seconds) for updates (e.g., `vmstat 5`). Terminate the output with CTRL-C.
`xload`	A graphical program that draws a graph showing system load over time.
`tload`	Similar to the `xload` program, but draws the graph in the terminal. Terminate the output with CTRL-C.