Computer Architecture

Lab 5 (Users, Groups, and Security)

1- Overview

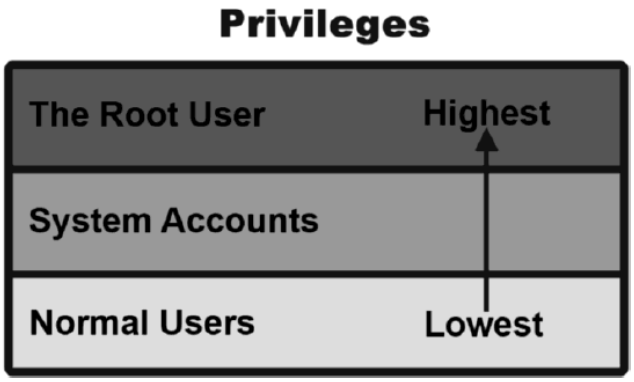
This chapter covers the most common commands related to users, groups, and security. It will also discuss topics like account creation/deletion, file and directory permissions, and other user/security related commands.

Commands covered in this lesson:

Command	Purpose
`chmod`	Change file and directory permissions.
`chown`	Change the owner of a file or directory.
`chgrp`	Change the group of files and directories.
`umask`	Display/set a user's default file creation mask.
`su`	Switch user accounts.
`sudo`	Run a single command as a different user.
`id`	Display information about a user's identity.
`groups`	Display which groups a user belongs to.
`who`	Display who is logged into the system.
`whoami`	Display the current user's identity
`w`	Display detailed information about users logged in to the system.
`last` `lastb`	Display the last successful/failed user logins.
`lastlog`	Display the most recent user login information.
`finger`	Display information a about user account.
`passwd`	Change passwords.
`useradd` `userdel`	Create/delete user accounts.
`adduser` `deluser`	Create/delete user accounts on Linux systems.
`groupadd` `groupdel`	Add/remove a group.
`usermod` `groupmod`	Modify user and group account settings.
`wall`	Broadcast a message to all users on the system.
`ulimit`	Display/set system resource limits.

2- ***Types of Accounts***

There are several types of user accounts used on Unix, Linux, and BSD systems. The graphic below illustrates the user security model used on most systems.



Unix/Linux/BSD user security model

By default, normal users and the programs they execute are given the least amount of privileges on the system. System accounts have slightly elevated privileges and are used to run system services (like a web server or FTP server). The root account has unrestricted administrative access to the entire system.

- Groups

Groups are used to simplify the management of system security. Users can be a member of one or more groups. All users are part of at least one group by default; this group is known as the user's primary group.

- File and Directory Permissions

File and directory permissions are managed using a set of nine "flags". The following example describes permissions found on a typical file.

-	rwx	r-x	r-x
	1 2 3	4 5 6	7 8 9
File Type	User (Owner)	Group	Other (Everyone)

Within these nine flags, three sets of permissions are specified:

- User (AKA owner) permissions
- Group permissions
- Other (i.e. everyone else)

There are four types of permissions that can be used to control access to a file or directory. The following table describes each permission.

Symbolic	Meaning
r	Read
w	Write
x	Execute
-	No access

Example of directory permissions

In most cases, the owner of a file will always have full read/write access to that file. Execute permission is a special flag used for programs, scripts, and directories to indicate they are executable.

The example below displays basic file permissions.

```
$ ls -l ShoppingList.txt
-rw-r--r-- 1 nick users 254 2009-06-01 15:35 ShoppingList.txt
```

Output of the ls -l command displaying file permissions

	User	Group	Other
Symbolic	rw-	r--	r--
Meaning	Read & Write	Read only	Read only

Example of file permissions

The next example demonstrates directory permissions. Directory permissions work the same as file permissions except they are used to control access to directories.

```
$ ls -ld finance/
drwxr-x--- 2 root finance 4096 2009-06-12 09:48 finance/
```

Output of the ls -ld command displaying directory permissions

Each file and directory has its own set of permissions. Permissions are not inherited from the parent directory. Additionally, directories require execute permission in order to be accessible, as shown in the previous example.

- **passwd**

Purpose: Change passwords.

Usage syntax: passwd [OPTIONS] [USER]

```
$ passwd
Enter new UNIX password: ******
Retype new UNIX password: ******
passwd: password updated successfully
```

Changing the current user's password

The **passwd** command changes a user's password. Executing passwd with no arguments changes the password for the current user as shown in the above example. The root user can change other user's passwords by specifying a username as demonstrated in the next example.

```
# passwd nick
Enter new UNIX password: ******
Retype new UNIX password: ******
passwd: password updated successfully
```

Changing a specific user's password

Common usage examples:	
passwd	Set the password for the current user
passwd [USER]	Set the password for the specified user
passwd -e [USER]	Force a user to change their password at the next login
passwd -l [USER]	Lock the specified user account
passwd -u [USER]	Unlock the specified user account
passwd -S [USER]	Display the status of the specified user account

➢ su

Purpose: Switch user accounts.

Usage syntax: su [OPTIONS] [USER]

```
$ whoami
nick
$ su
Password: ******
# whoami
root
```

Using the su command to switch from a normal user to the root user

The **su** command (short for Switch User) allows you to login as another user without having to first log out of the system. In the above example su is used by a normal user to switch to the root user account. Notice that when you become the root user your shell prompt changes from **$ to #**. As the root user you can now run commands that require elevated privileges.

By default, executing **su** with no arguments switches to the root user account. A user name can be specified with **su** to become a different user as shown in the next example.

```
$ whoami
nick
$ su steve
Password: ******
$ whoami
steve
```

Using su to switch to another user

Common usage examples:	
`su`	Switch to the root user account
`su -`	Switch to the root user account and load root's profile
`su [USERNAME]`	Switch to the specified username

➢ sudo

Purpose: Run a single command as a different user.

Usage syntax: sudo [OPTIONS] [COMMAND]

```
$ whoami
nick
$ sudo whoami
[sudo] password for nick: *******
root
$ whoami
nick
```

Using sudo to run a command as the root user

The **sudo (Super User Do)** command allows you to run a single command as another user. It is most commonly used to execute commands that require root privileges. In this example **whoami** is executed as root via the **sudo** command. Using the **sudo** command is the recommended way to run commands that require elevated privileges as it limits the amount of time spent with root privileges. This greatly helps prevent disasters such as accidental deletion of important system files.

Note	*User (or group) accounts must be listed in the* `/etc/sudoers` *file in order to execute commands as root with* **sudo**.

Common usage examples:

`sudo [COMMAND]`	Run the specified command as root
`sudo -u [USER] [COMMAND]`	Run a command as the specified user
`sudo !!`	Run the last command as root

- To enable the root account, give it a password: **$ sudo passwd**
- To disable the root account: **$ sudo passwd -l root**
- To temporarily lock a user account: **$ sudo passwd -l username**
- To temporarily unlock a user account: **$ sudo passwd -u username**

3- Managing Users and Groups

There are four main user administration files:

- **/etc/passwd** – Keeps the user account and password information. This file holds the majority of information about accounts on the Unix system.

- **/etc/shadow** – Holds the encrypted password of the corresponding account. Not all the systems support this file.
- **/etc/group** – This file contains the group information for each account.
- **/etc/gshadow** – This file contains secure group account information.

Check all the above files using the **cat** command. The next command is important to manage file and directory permission.

➢ chmod

Purpose: Change file and directory permissions.

Usage syntax: `chmod [OPTIONS] [MODE] [DIRECTORY/FILE]`

```
# chmod 664 ShoppingList.txt
# ls -l ShoppingList.txt
-rw-rw-r-- 1 root root 23 2009-05-27 22:31 ShoppingList.txt
```

Using the chmod command to change file permissions

The chmod command sets permissions on files and directories. By default, permissions are specified in numerical (octal) format such as 664 as shown in the above example. In octal form, three digits are used to represent owner, group, and everyone else's permissions. The first number represents the owner's permissions, the second number is the group's permissions, and the third number is for everyone else. The table below provides a cross reference of symbolic and octal permissions.

Permission	Symbolic	Octal
Read	r	4
Write	w	2
Execute	x	1
None	-	0

Permissions cross reference

The sum of the octal permissions becomes what is known as the mode. The valid modes are described in the following table.

Mode	Octal	Symbolic	Effective Permission
7	4+2+1	rwx	Read/Write/Execute
6	4+2	rw-	Read/Write
5	4+1	r-x	Read/Execute
4	4	r--	Read
0	0	---	None

Mode cross reference

The combination of 3 modes determines the permissions for the file. A mode of 664 would create rw-rw-r-- permissions giving read/write access to the user and group, and read only to everyone else. The concept of permissions on Unix, Linux, and BSD systems can be hard to grasp as first.

Example 1: A mode of 660 would provide read/write access to the owner and group and no access to everyone else.

```
# chmod 660 MyFile
# ls -l MyFile
-rw-rw---- 1 root sales 23 2009-05-27 22:31 MyFile
```

Result of a 660 mode

Example 2: A mode of 755 would provide full access to the owner and read/execute access for the group and everyone else.

```
# chmod 755 MyProgram.sh
# ls -l MyProgram.sh
-rwxr-xr-x 1 root sales 23 2009-05-27 22:31 MyProgram.sh
```

Result of a 755 mode

Example 3: A mode of 600 would provide read/write access to the owner and no access to everyone else.

```
# chmod 600 MyFile
# ls -l MyFile
-rw------- 1 root sales 23 2009-05-27 22:31 MyFile
```

Result of a 600 mode

Example 4: A mode of 775 applied to a directory would provide read/write/execute access to the owner and group and read only access to everyone else.

```
$ chmod 775 MyDirectory
$ ls -ld test
drwxrwxr-x 2 root sales 4096 2009-05-27 16:04 MyDirectory
```

Result of a 775 mode applied to a directory

Common usage examples:	
`chmod [MODE] [FILE]`	Change the permissions on the specified file
`chmod [MODE] -R [DIR]`	Recursively change the permissions on all files

The following table lists out commands to **create and manage accounts and groups**:

	Command & Description of Users account and Groups
1.	**Useradd / Userdel :**Adds / Deletes accounts to the system
2.	**adduser / deluser :** Adds/ Deletes user (**username**)accounts to the system
3.	**groups** : Display which groups a user belongs to
4.	**Groupadd / groupdel :** Adds / Removes groups to the system
5.	**groupmod :** Modifies group attributes
6.	**Usermod :** Modifies user account attributes
7.	**Chown** : Change the owner of a file or directory
8.	**Chgrp** : Change the group of a file or directory
9.	**Umask** : Display/set a user's default file creation mask

➢ useradd / userdel

Purpose: Create/delete user accounts.

Usage syntax: useradd [OPTIONS] [USER]

```
# useradd -m steve
# passwd steve
Enter new UNIX password: ******
Retype new UNIX password: ******
passwd: password updated successfully
```

Adding a user account to the system (and setting their password)

The **useradd** command creates new user accounts. In the above example, executing **useradd -m steve** creates a basic login account for a user. The **-m** option is used to automatically create a home directory for the specified user (recommended). The password is then set for the new user using the previously discussed passwd command.

The **userdel** command deletes user accounts from the system. The next example demonstrates using **userdel** to remove an account. The optional **-r** option is used to have the system automatically delete the specified user's home directory after removing their account.

Usage syntax: `userdel [OPTIONS] [USER]`

```
# userdel -r steve
```

Removing a user account

Common usage examples:

`useradd [USER]`	Create the specified user account
`useradd -m [USER]`	Automatically create a home directory for the user
`userdel [USER]`	Delete the specified user account
`userdel -r [USER]`	Delete a user's account and their home directory

- **adduser / deluser**

Purpose: Create/delete user accounts on Linux systems.

Usage syntax: adduser [OPTIONS] [USER]

```
# adduser mike
Adding user 'mike' ...
Adding new group 'mike' (1002) ...
Adding new user 'mike' (1002) with group 'mike' ...
Creating home directory '/home/mike' ...
Copying files from '/etc/skel' ...
Enter new UNIX password: ******
Retype new UNIX password: ******
passwd: password updated successfully
Changing the user information for mike
Enter the new value, or press ENTER for the default
        Full Name []: Mike Smith
        Room Number []: Computer Room
        Work Phone []: 555-1212
        Home Phone []:
Is the information correct? [y/N] y
```

Creating a user with the adduser command

The **adduser** command is a *user-friendly frontend* for the previously discussed **useradd** command. It simplifies the creation of user accounts on Linux systems by prompting for necessary information when creating accounts (rather than having to specify a number of command line options). The above example demonstrates the typical usage of the **adduser** command. The **deluser** command deletes user accounts as shown in the next example.

- **Usage syntax:** deluser [OPTIONS] [USER]

```
# deluser mike
Removing user 'mike' ...
Warning: Removing group 'mike', since no other user is part of it.
Done.
```

Removing a user with the deluser command

Common usage examples:

adduser [USER]	Create a user account
deluser [USER]	Remove a user account

➢ groups

Purpose: Display which groups a user belongs to.

Usage syntax: groups [OPTIONS] [USER]

```
# groups
root
```

Displaying group information for the current user

The groups command displays a user's group membership. Executing groups with no options displays the current user's groups, as shown in the above example. A user name can be used with the groups command to display the specified user's group membership as shown in the next example.

```
# groups nick
nick adm dialout cdrom plugdev lpadmin admin sambashare
```

Displaying group information for the specified user

Common usage examples:

groups	Display the current user's group membership
groups [USER]	Display group membership for the specified user

➢ groupadd / groupdel

Purpose: Add/remove a group.

Usage syntax: groupadd [GROUP]

Usage syntax: `groupadd [GROUP]`

```
# groupadd accounting
```

Creating a new group with groupadd

The groupadd command creates new group accounts. Groups are helpful in managing access to files and directories in a multiuser environment. In the above example a new group called accounting is created. The resulting group entry in the /etc/group file is displayed below.

```
# grep accounting /etc/group
accounting:x:1002:
```

Displaying a group entry in the /etc/group file

The groupdel command deletes groups from the system. The next example demonstrates using the groupdel command to remove the previously created accounting group from the system.

Usage syntax: `groupdel [GROUP]`

```
# groupdel accounting
```

Deleting a group using groupdel

Common usage examples:

`groupadd [GROUP]`	Create a new group
`groupdel [GROUP]`	Delete a group

- **usermod / groupmod**

Purpose: Modify user and group account settings.

Usage syntax: usermod [OPTIONS] [USER]

```
# usermod -aG sales nick
```

Changing a user's group membership using the usermod command

The usermod command modifies user account settings. In the above example, the **-aG option** is used to add the user nick to the sales group. You can also use the usermod command to change a user's home directory location using **the -d option or default shell using -s.**

The groupmod command modifies groups. Its primary purpose is to rename a group. In the next example the accounting group is renamed to finance using **groupmod -n**.

Usage syntax: groupmod [OPTIONS] [GROUP]

```
# groupmod accounting -n finance
# grep finance /etc/group
finance:x:1002:
```

Renaming a group using the groupmod command

Common usage examples:	
`usermod -s [SHELL] [USER]`	Change a user's default shell
`usermod -d [DIR] [USER]`	Change a user's home directory location
`usermod -aG [GROUP] [USER]`	Add a user to the specified group
`groupmod [OLD] -n [NEW]`	Rename the specified group

➢ chown

Purpose: Change the owner of a file or directory.

Usage syntax: chown [OPTIONS] [USER:GROUP][DIRECTORY/FILE]

```
# ls -l ShoppingList.txt
-rw-r--r-- 1 nick nick 23 2009-05-27 22:31 ShoppingList.txt
# chown root ShoppingList.txt
# ls -l ShoppingList.txt
-rw-r--r-- 1 root nick 23 2009-05-27 22:31 ShoppingList.txt
```

Using the chown command to change the owner of a file

The chown command changes the owner of a file or directory. In the above example, the owner of the ShoppingList.txt file is changed from nick to root.

The next example demonstrates changing both the owner and group of a file using the chown command.

```
# ls -l ShoppingList.txt
-rw-r--r-- 1 root nick 23 2009-05-27 22:31 ShoppingList.txt
# chown nick:sales ShoppingList.txt
# ls -l ShoppingList.txt
-rw-r--r-- 1 nick sales 23 2009-05-27 22:31 ShoppingList.txt
```

Using the chown command to change the owner and group of a file

After executing the chown nick: sales ShoppingList.txt command, the ShoppingList.txt file is updated with the owner of nick and the group of sales.

Common usage examples:	
`chown [USER] [FILE]`	Change the owner of a file
`chown [USER]:[GROUP] [FILE]`	Change the owner and group of a file
`chown -R [USER] [DIR]`	Recursively change the owner on all files in the specified directory

➢ chgrp

Purpose: Change the group of files and directories.

Usage syntax: chgrp [OPTIONS] [GROUP] [DIRECTORY/FILE]

```
# ls -l ShoppingList.txt
-rw-r--r-- 1 root root  23 2009-05-27 22:31 ShoppingList.txt
# chgrp sales ShoppingList.txt
# ls -l ShoppingList.txt
-rw-r--r-- 1 root sales 23 2009-05-27 22:31 ShoppingList.txt
```

Using the chgrp command to change the group of a file

The chgrp command changes the group of a file or directory. In the above example the chgrp command is used to change the group from root to sales on the ShoppingList.txt file.

➢ umask

Purpose: Display/set a user's default file creation mask.

Usage syntax: umask [OPTIONS] [MODE]

```
$ umask
022
```

Displaying the current user's umask

umask controls a user's default file creation mask. This determines the permissions that will be assigned to newly created files and directories. To determine the file/directory creation mode the umask value is subtracted from 777 for directories and 666 for files. For example, a umask of 022 would create effective permissions of 644 (rw-r--r--) for files and 755 (rwxr-xr-x) for

directories. On some systems the **-S option** can be used to display a more user friendly symbolic output of the umask value, as shown in the next example.

```
$ umask -S
u=rwx,g=rx,o=rx
```

Displaying the umask in symbolic notation

The umask command can also be used to change the umask value as displayed in the next example.

```
$ umask 077
```

Setting the umask value

In this example, a umask value of 077 would create effective permissions of 600 (rw-------) for files and 700 (rwx------) for directories.

4- User information and Security

➢ id

Purpose: Display information about a user's identity.

Usage syntax: id [OPTIONS] [USER]

```
# id
uid=0(root) gid=0(root) groups=0(root)
```

Displaying user and group information for the current user

The **id** command displays user and group information for the specified user. Executing id with no options displays the current user's information as displayed in the above example.

```
# id nick
uid=1000(nick) gid=1000(nick) groups=4(adm), 20(dialout),24(cdrom),
46(plugdev), 106(lpadmin), 121(admin), 122(sambashare), 1000(nick)
```

Displaying user and group information for a specific user

Note	*Unix, Linux, and BSD systems assign a numerical UID (**User ID**) and GID (**Group ID**) for each user and group on the system. A user friendly name is also assigned to each UID and GID which is displayed in parenthesis next to each ID number. This information is stored in* `/etc/passwd` *for users and* `/etc/group` *for groups.*

Common usage examples:

`id`	Display the current user's ID information
`id [USER]`	Display user and group information for the specified user

➢ who / whoami

Purpose: Display who is logged into the system.

```
$ who
root     tty2         2010-05-17 11:32
nick     tty1         2010-05-17 11:31
nick     pts/0        2010-05-17 08:40 (10.10.1.251)
dave     pts/1        2010-05-17 12:32 (10.10.1.188)
mike     pts/2        2010-05-17 14:28 (10.10.1.167)
lisa     pts/3        2010-05-17 14:50 (10.10.1.204)
nick     pts/4        2010-05-17 15:33 (10.10.1.251)
```

Output of the who command

The who command displays information about users currently logged in to the system. The default output of the who command displays the username, terminal ID, and date/time the user logged in as shown in the above example.

The **whoami** command displays the username of the current user. This is helpful to verify which user's environment and security privileges are available when switching between different accounts.

Usage syntax: whoami

```
$ whoami
nick
```

Using whoami to display the name of the current user

- **w**

Purpose: Display detailed information about users logged in to the system.

Usage syntax: w [OPTIONS] [USER]

```
$ w
 15:39:12 up 4 days,  6:09,  5 users,  load average: 0.06, 0.05, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
nick     pts/0    10.10.1.251      08:40    1:10    0.18s  0.15s -bash
dave     pts/1    10.10.1.188      12:32    0.00s   0.14s  0.14s vim
mike     pts/2    10.10.1.167      14:28    0.00s   0.12s  0.12s tail
lisa     pts/3    10.10.1.204      14:50    9.00s   0.14s  0.14s -bash
nick     pts/4    10.10.1.251      15:33    0.00s   0.15s  0.01s w
```

Output of the w command

The **w command** shows detailed information about users logged into the system. It is similar to the previously discussed who command except it provides additional information such as the user's last login time, how long they have been idle, and what program they are currently running. The **w command** also displays a system summary line that shows the host's uptime, number of connected users, and samples of system load averages for the past 1, 5, and 15 minutes.

- **last / lastb**

Purpose: Display the last successful/failed user logins.

Usage syntax: last [OPTIONS] [USER]

```
$ last
nick     pts/1        192.168.1.50  Sat May 22 13:42   still logged in
nick     pts/0        192.168.1.50  Sat May 22 13:40   still logged in
nick     tty7         :0            Sat May 22 13:40   still logged in
reboot   system boot  2.6.32-21-ge  Sat May 22 13:40 - 14:02  (00:21)
nick     pts/0        :0.0          Thu May 20 21:33 - 21:36  (00:03)
nick     pts/0        :0.0          Thu May 20 21:30 - 21:33  (00:02)
root     pts/0        :0.0          Thu May 20 21:27 - 21:30  (00:03)
nick     tty7         :0            Thu May 20 21:25 - crash (1+16:14)
...
```

Output of the last command

The **last command** displays the login and logout times for each user on the system. It also shows information about system shutdowns and restarts as

shown in the above example. The **lastb** command displays failed login attempts and shown in the next example.

Usage syntax: `lastb [OPTIONS] [USER]`

```
$ lastb
nick     tty1                      Sat May 22 13:59 - 13:59  (00:00)
nick     tty1                      Thu May 20 21:27 - 21:27  (00:00)
root     tty1                      Thu May 20 15:27 - 15:27  (00:00)

btmp begins Sat May 20 13:59:28 2010
```

Output of the lastb command

Common usage examples:

`last`	Display the last user login information
`last -[NUMBER]`	Display the specified number of logins
`last [USER]`	Display the last logins for the specified user
`lastb`	Display failed login attempts
`lastb -[NUMBER]`	Display the specified number of failed login attempts
`lastb [USER]`	Display failed login attempts for the specified user

➢ lastlog

Purpose: Display the most recent user login information.

Usage syntax: lastlog [OPTIONS]

```
$ lastlog | more
Username         Port     From           Latest
root             tty2                    Sat May 30 11:32:33 -0500 2009
dave             tty3                    Sat May 30 10:22:51 -0500 2009
nick             pts/0    10.10.1.251    Sat May 30 11:31:51 -0500 2009
steve                                    **Never logged in**
bin                                      **Never logged in**
sync                                     **Never logged in**
lp                                       **Never logged in**
...
```

Output of the lastlog command

The **lastlog command** displays the most recent user login time and dates for every user on the system. Executing lastlog with no options displays the last login information for all users, as shown in the above example. This output is similar to the last command except lastlog only displays the most current login activity where last displays all available login events.

The **-u option** can be used to display the last login for a specific user as demonstrated in the next example.

```
$ lastlog -u nick
Username         Port     From           Latest
nick             pts/0    10.10.1.251    Sat May 30 11:31:51 -0500 2009
```

Displaying the last login for a specific user

Common usage examples:	
`lastlog`	Display the last login information for all users
`lastlog -u [USER]`	Display the last login information for the specified user

➢ finger

Purpose: Display information about a user account.

Usage syntax: finger [OPTIONS] [USER]

```
$ finger nick
Login: nick                             Name: Nick Marsh
Directory: /home/nick                   Shell: /bin/bash
On since Sat May 30 11:32 (CDT) on tty2   10 minutes 32 seconds idle
New mail received Mon May 17 16:08 2010 (CDT)
     Unread since Tue Apr 13 08:43 2010 (CDT)
No Plan.
```

Using the finger command to display information about a user account

The finger command displays information about user accounts. It shows details about the user's shell, home directory, and other helpful information. The example above demonstrates the typical user information displayed when using the finger command.

➢ wall

Purpose: Broadcast a message to all users on the system.

Usage syntax: wall [FILE]

```
# wall
Anyone want some tacos?

<CTRL + D>
```

Using the wall command to send a message to all users logged into the system

The wall command sends a message to all users currently logged into the system. The text entered in the above example will display on all local terminals and remote sessions currently logged into the system.

Note	*Pressing* **CTRL + D** *ends the message editor and sends the message.*

The next example displays a sample of the wall message output as seen by other users on the system.

```
$

Broadcast Message from root@e6400 (/dev/pts/0) at 11:56 ...

Anyone want some tacos?
```

Output of the wall message displayed on all terminals

In place of manually entering a message, a text file with a prewritten message can be used with the wall command, as shown in the next example.

```
# wall /home/nick/message.txt
```

Using a text file to send a message with the wall command

Common usage examples:	
wall	Send a message to all users
wall [FILE]	Send the message in the specified file to all users

➢ ulimit

Purpose: Display/set system resource limits.

Usage syntax: ulimit [OPTIONS] [LIMIT]

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 20
file size               (blocks, -f) unlimited
pending signals                 (-i) 16382
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Displaying defined resource limits using the ulimit command

The **ulimit command** displays and sets system resource limits. These limits control the maximum amount of system resources available to programs. It can be used to control the maximum amount of memory, CPU time, and file sizes available to each program launched by a user.

Tip	*Ulimit configuration is typically stored in* **/etc/limits.conf** *or* **/etc/security/limits.conf** *on most systems.*

Common usage examples:

ulimit -a	Display all defined resource limits
ulimit [OPTION] [LIMIT]	Set ulimit values