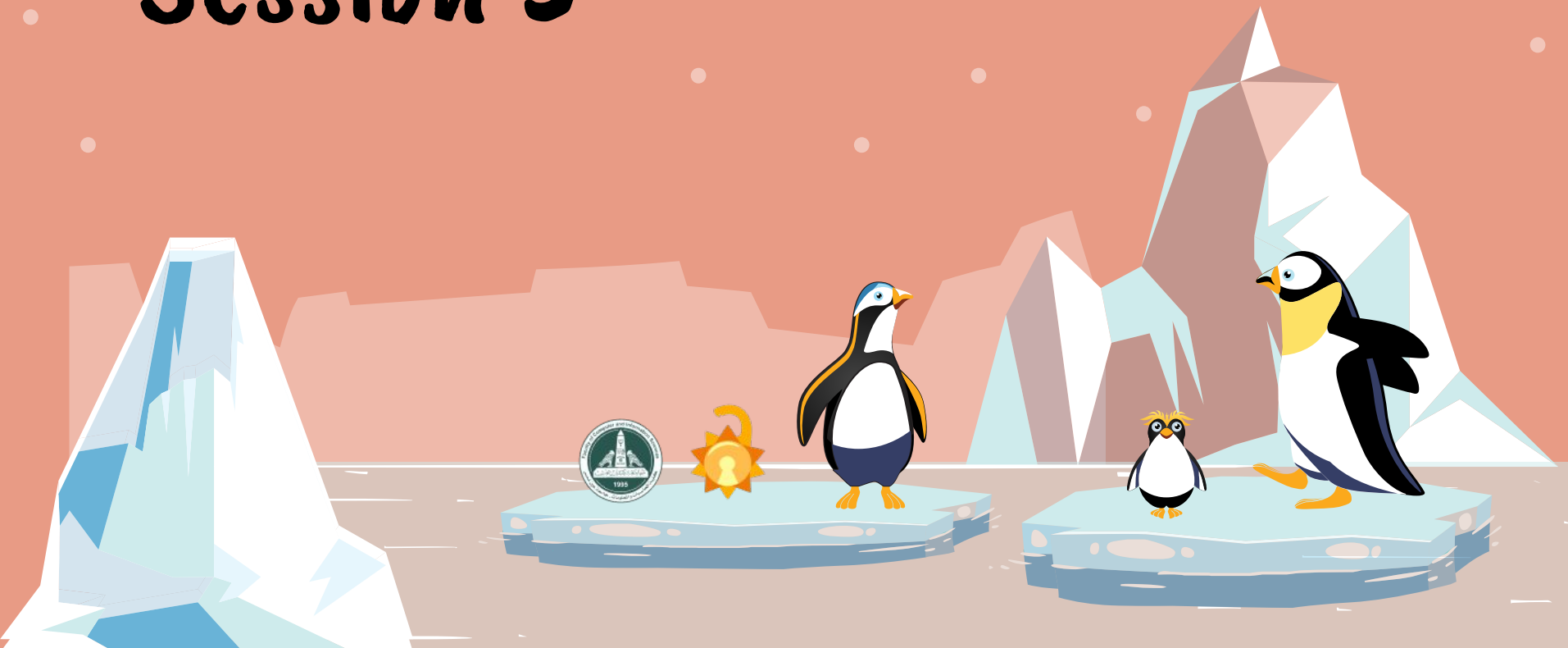




# Session 3





# Agenda

01

File Permissions and  
Ownership

02

Dealing with Users & Groups

03

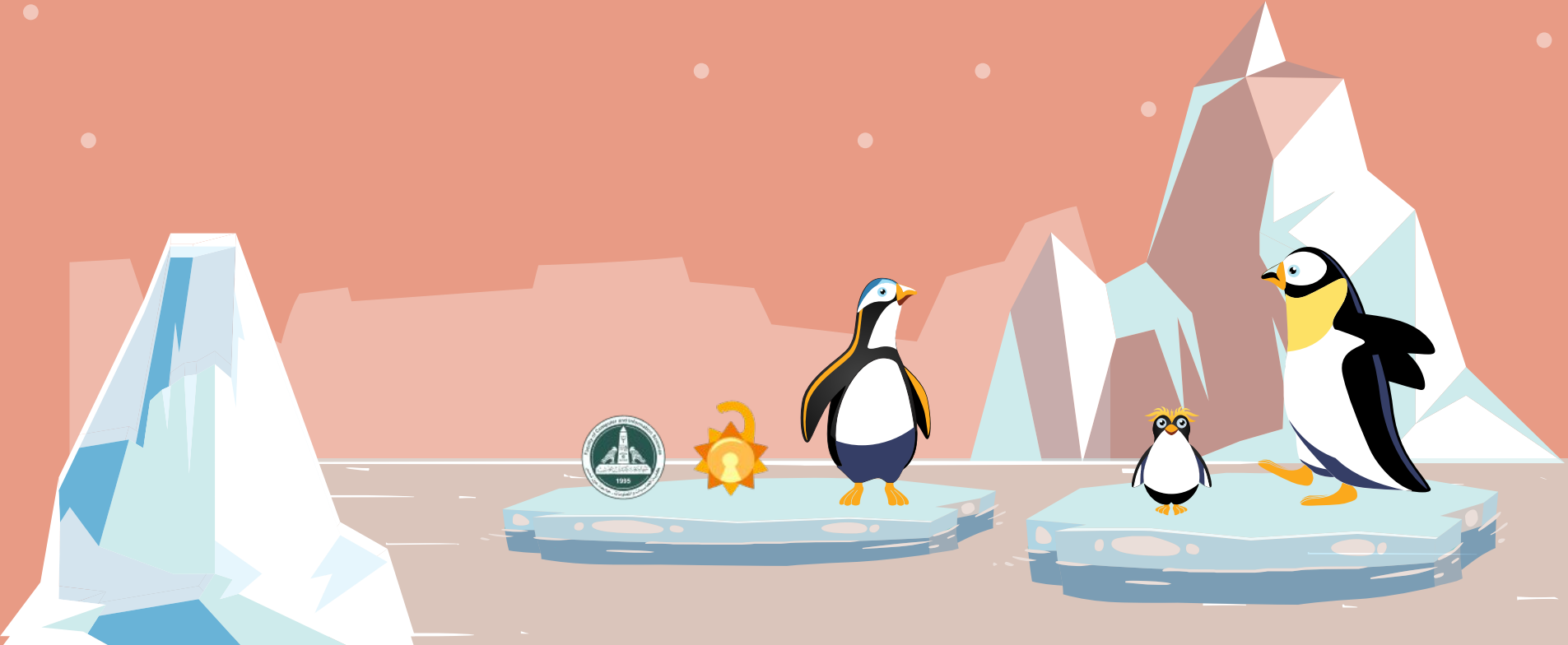
Processes

04

Package Managers



# Files Permissions & Ownership





# Linux File Ownership





# Three different types of owners

## 1. User:

A user is the owner of the file. By default, the person who created a file becomes its owner.

## 2. Group:

A group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file.

## 3. Others:

Any other user who has access to a file. This person has neither created the file, nor he belongs to a group who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.



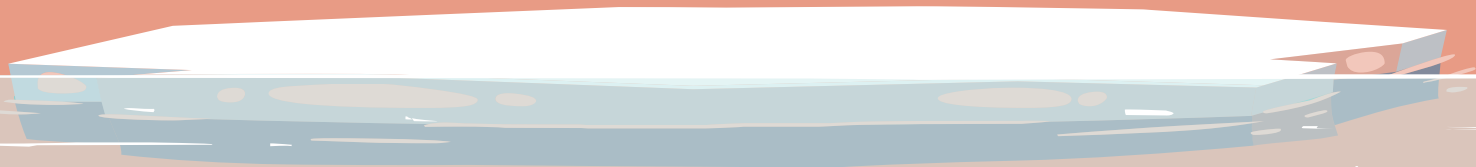
# Linux File Permissions







# Changing Permissions







# The chmod command

There are two way to use the chmod:

1. Absolute mode
2. Symbolic mode

# 1- Absolute mode

In this mode the permission for each kind of owner is symbolised by a number where:

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rxw

So the command "chmod 764 sample" means:

1. The owner has read write and execute permissions
2. The group to which this file belongs only has read and write permissions
3. Other users have only read permission



## 2- Symbolic mode

- The symbolic mode is more human readable and using it you can edit permission to a specific type of owner
- Syntax:  
“Chmod <owner type><operator><new permission><file name>”

User	Denotations
u	user/owner
g	group
o	other
a	all

Operator	Description
+	Adds a permission to a file or directory.
-	Removes the permission.
=	Sets the permission and <b>overrides</b> the permissions set earlier.



## 2- Symbolic mode (cont.)

- An example command is `chmod g-rwx sample.txt`
- The above command removed read write and execute permissions from the group owning the file `sample.txt`

# Hands on



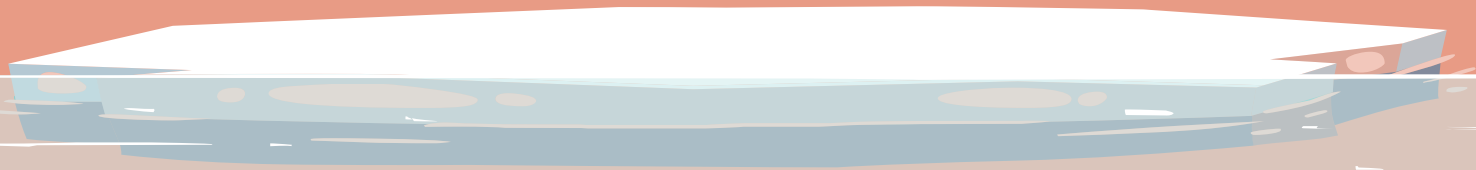


## Try the following

1. Create a text file called "test.txt" in a folder on the desktop called "testFolder"
2. Give the user who owns the file execute permission
3. Stop users who are in the group owning the file from being able to edit it
4. Stop users who are neither the owner nor in the owning group from being able to list the content of the "testFolder" directory



# Changing Ownership & Group





# Two commands

1. The “chown” command
2. The “chgrp” command





# 1- The “chown” command

- This command could be used to change both the owner and the group to which a file or directory belongs
- To change file or directory owner: “chown <new user> <file/directory name>”
- To change file or directory owner and group: “chown <new user>:<new group> <file name>”



## 2- The “chgrp” command

- This command can only be used to change the group to which a file or directory belongs
- To change file or directory group: “chgrp <new group> <file/directory name>”

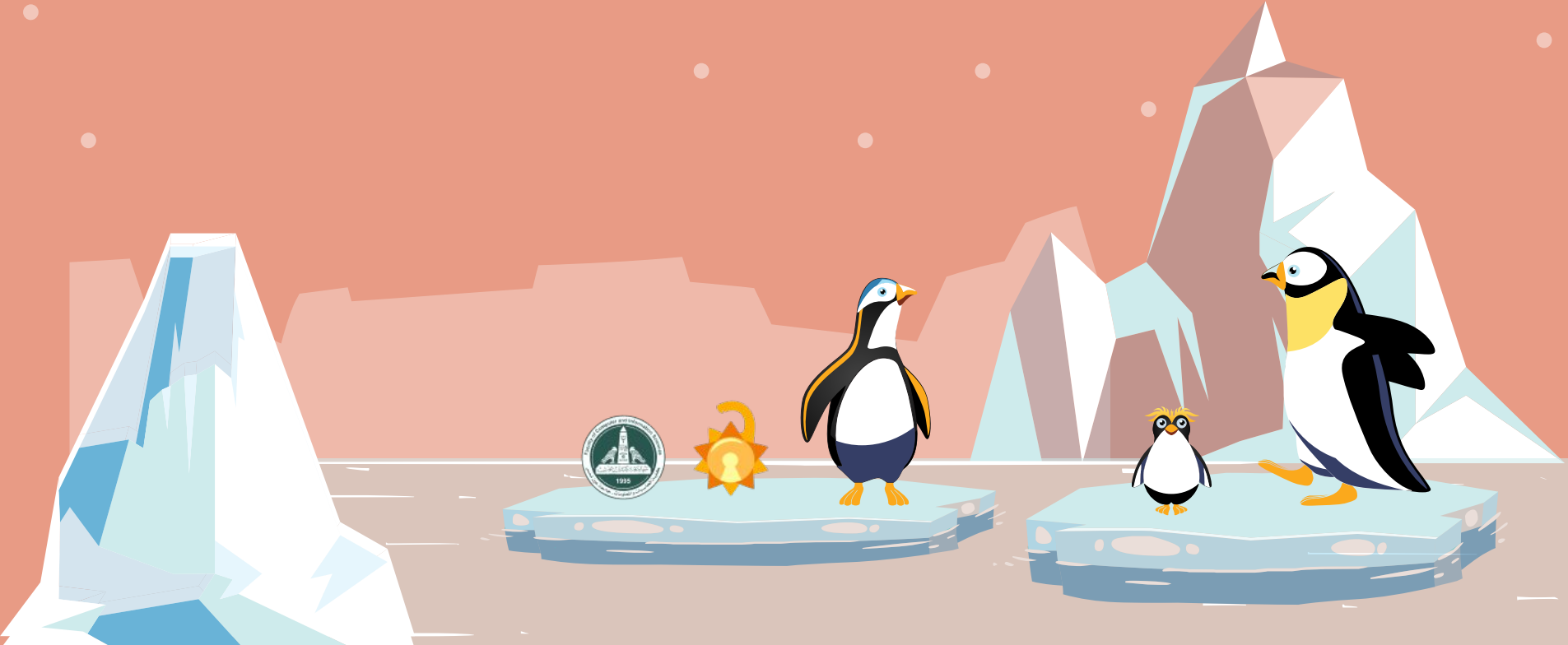


# The "groups" command

- Executing this command without any additional arguments will list all the groups that the current account belongs to
- If an account name is specified after the command (ex: "groups OSC), the command will list the groups to which the specified account belongs to



# Dealing with Users & Groups





# User Account Types





# Three different types of accounts

## 1. Superuser/root (UID=0):

This is the administrator of the system and has unlimited access to the resources of the system and unlimited ability to alter it

## 2. System user (UID=1-999):

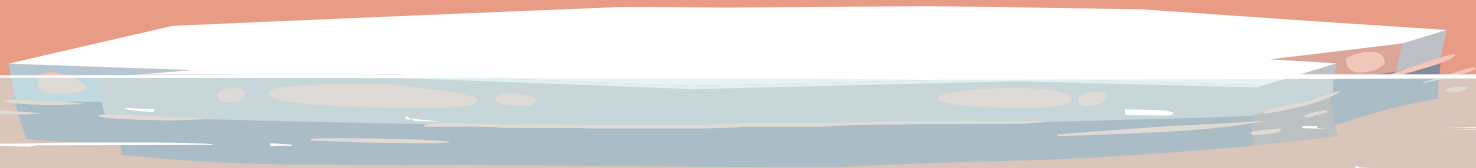
These are accounts created by programs supporting the operation of the system. They exist so that each process can secure and isolate its resources and files from other processes

## 3. Regular User (UID $\geq$ 1000):

These are the accounts used by the users of the system to finish their day to day work



# User groups





# Two types of user groups

## 1. Primary group:

This is the group that a new user creates on the system belongs to. It is named the same name of the user and is the default group that a files created by the user belongs to.

Only one primary group can exist for a given user account

## 2. Supplementary groups:

Any other group a user belongs to other than the primary group



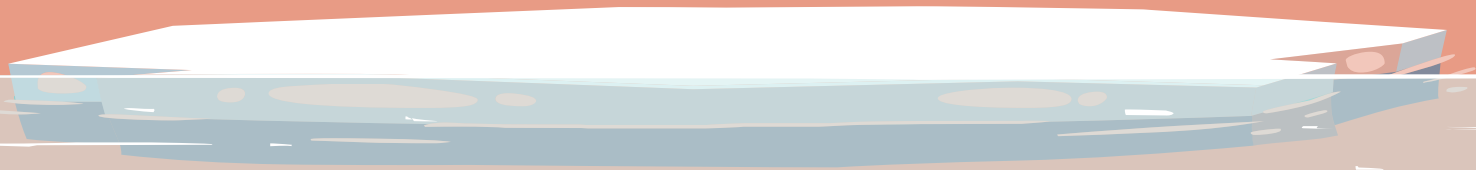


# The "id" command

- ID is a command used to get the UID and GID(s) of different users on the system
- When used without addition arguments it shows the UID, primary group GID and supplementary group(s) GID(s) of the logged in user.
- If a user account name is specified it shows the UID, primary group GID and supplementary group(s) GID(s) of the specified user.



***Know who is on your system***





# The “/etc/passwd” file

- This file stores information about the users on the system.
- Each line is giving information about a different user and is formatted as follows (the GID in this files is that of the primary group of the user)

```
root:x:0:0:root:/root:/bin/bash
```

Diagram illustrating the fields in the /etc/passwd file entry for the root user:

- Username
- User's password
- User ID
- Group ID
- Comment Field
- User's home directory
- User's shell

Note:

if the user shell is set to  
“/sbin/nologin” that mean interactive  
logins are not allowed for that user



# The “/etc/shadow” file

- This file stores information about user authentication. Each line is information about a different user and is formatted as follows

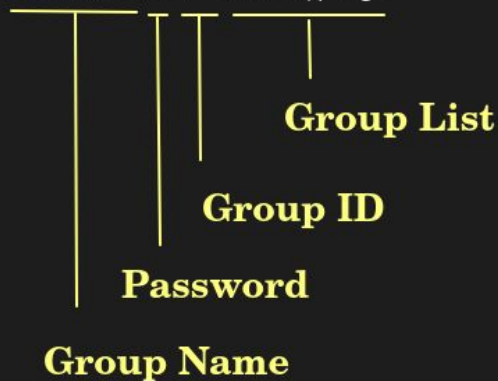
root:\$y\$j9T\$55g52AbKXrqjbNSctu9/v/\$on/M6pxcEHrAdqM650BlzpNJg4CCvS1sLsuDqfFD TN7:19195:0:99999:7:::						
Username	Encrypted password					Account expiration date
						Password inactivity period
						Password warning period
						Maximum password age
						Minimum password age
Date of last password changed						



# The “/etc/group” file

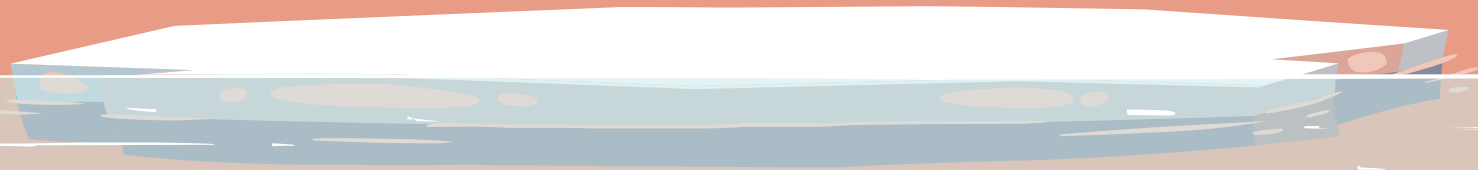
- This file stores information about user groups on the system. Each line is information about a different group and is formatted as follows

```
wireshark:x:974:salma,pingo
```





# Gaining Superuser Access (the “sudo” command)



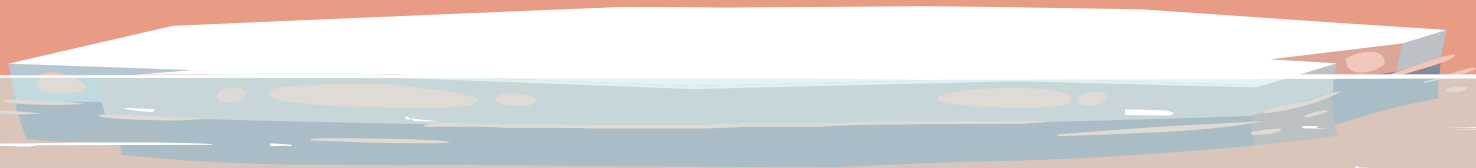


# The Sudo mechanism

- When we use the sudo command before a command that is reserved for the root user only, sudo searches the “sudoers” file (a file only accessed by the root user) and then checks if the root user gave you permission to run this command
- If it finds that you have the permission, you are then allowed to use this command with root privileges
- Else you are denied use of the command



# Managing Users







# Creating users

- Users are added using the “useradd” command
- syntax : “useradd [options] <username>”
- For unspecified options the command reads them from the “/etc/default/useradd” and the “/etc/login.defs” files



# Editing and setting passwords

- Passwords are edited and set using the “passwd” command
- syntax : “passwd <username>”
- The command will then prompt you to change or set the password for the account specified
- Of course only the root user or a user with sudo privileges can change and set password for other accounts



# Switching users

- We can switch between users using ``su <username>`` or ``su - <username>``
- The difference is that ``su - <username>`` is like logging into the user while ``su <username>`` is like opening a terminal from this user only.
- The password for the user we are switching to is needed unless you are the root user.



# Deleting users

- Users are deleted using the “userdel” command
- syntax : “userdel [options] <username>”
- If the “-r” option is specified the command deletes the home directory of the user; else, it only removes the data of the user from the “/etc/passwd” files while keeping the files in the home directory
- Note: leaving the home directory can lead to security issues regarding file access



# Managing groups





# Creating groups

- groups are created using the “groupadd” command
- syntax : “groupadd [options] <group name>”
- For unspecified option the command reads them from the “/etc/login.defs” file



# Adding users to groups

- Existing users accounts are added to groups using the “usermod” command
- syntax : “usermod [options] <group name> <username>”
- So to add the user OSC to the group testGroup we will write “usermod -a -G testGroup OSC”
- The “-G” option tells the command that we will add the user to a supplementary group. The “-a” option puts the command in append mode; otherwise, the command will remove the user from all groups unspecified in the command (will overwrite instead of add)



# Deleting groups

- Groups are deleted using the “groupdel” command
- syntax : “groupdel <group name>”
- You cannot delete the primary group of a user account



# Hands on





# Try the following

1. Create a user called "testUser"
2. Set the password for testUser (make it something you can remember)
3. Create a group called "testGroup"
4. Get the UID, primary group GID, and supplementary groups of the new user
5. Add "testUser" to "testGroup"
6. Finally delete "testGroup" and act nothing happened




# Processes





# Process Types





Foreground Processes (Non  
Automatic)



Background Processes  
(Automatic)





## Foreground Processes (Non Automatic)

- Initialised and controlled from a terminal session.
- There has to be a user connected to the system to start such processes.
- When a process is run in the foreground, no other process can be run through the terminal until this process is finished or killed.
- Examples: File manager, Firefox, VS code.



## Background Processes (Automatic)

- Not connected to any terminal as they don't expect any user input.
- Can't be seen by the user.
- To run a process in the background add `&` after the command.
- Examples: Network manager, Update manager



# Background Processes (cont.)

```
[I] momentk at navi in ~  
└─[i]─λ firefox &  
[I] momentk at navi in ~  
└─[i]─λ |
```

Process here is running in background.

To view background processes we use the `jobs` command

```
[I] momentk at navi in ~  
└─[i]─λ firefox &  
[I] momentk at navi in ~  
└─[i]─λ jobs  
Job      Group  CPU    State  Command  
1         93211  6%     running  firefox &
```

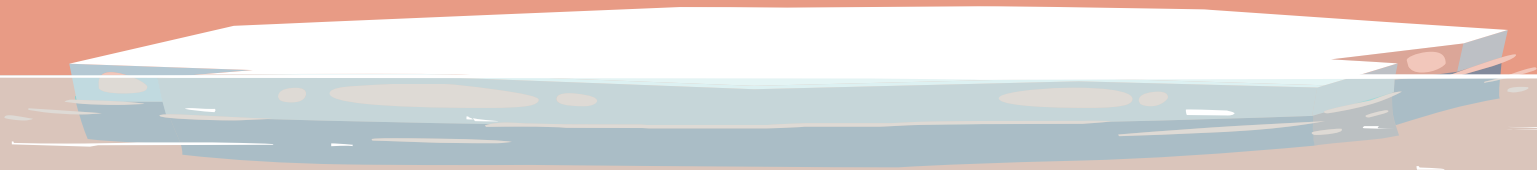
To bring a process from the background to the foreground use the `fg` command, `fg %job\_id`

```
[I] momentk at navi in ~  
└─[i]─λ fg %1  
Send job 1 (firefox &) to foreground
```





# Process Relationships





Parent Process



Child Process





# Process Relationships

## Parent

A process that created another process, directly or indirectly.

## Child

A process created by another process (its parent process).

- Every process has a parent process except the initial kernel process 'PID 0'.

**If a parent process is closed what will happen to its children?**



Orphan Process



Zombie Process





# Process Relationships

## Orphan

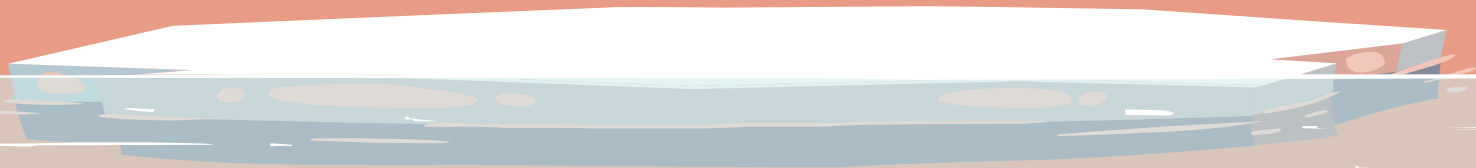
This is a process that has lost its parent process. This can happen if the parent process terminates before the child process does. When this occurs, the orphan process is assigned a new parent process, which is typically the init process (PID 0) on most systems.

## Zombie

This is a process that has completed execution (already dead) but still has an entry in the process table, as its parent process has not yet retrieved its exit status. Zombie processes are usually terminated once the parent process retrieves the exit status.



# ps Command





# ps command

- Used to display information about the currently running processes

```
~  
[~] ps  
  PID TTY          TIME CMD  
 713541 pts/2    00:00:00 bash  
 715490 pts/2    00:00:00 ps
```

- Provides other details like the process ID (PID), parent process ID (PPID), process state, user ID, CPU usage, memory usage and command.



# ps command (cont..)

- ``ps`` has many options here are some of them:
  - a Displays info about all processes, including those not associated with the terminal.
  - e Similar to ``-a`` but also displays kernel processes.
  - f Displays full list for the processes including user ID, PPID, start time and CPU usage.
  - l Displays long list for all processes.
  - u Displays list of processes belonging to a specific user.
  - x Displays list of processes not associated with the terminal.





# ps command (cont..)

You'll notice you're seeing a lot more fields now, no need to memorize them all. Here's a quick summary of the most important ones:

- **USER:** The effective user (the one whose access we are using)
- **PID:** Process ID
- **TTY:** Controlling terminal associated with the process
- **START:** Start time of the process
- **TIME:** Total CPU usage time
- **COMMAND:** Name of executable/command
- **STAT:** Process status code



# ps command (cont..)

Each process has a stat, a process stat can be one of the following:

- **Running process ( R ):**  
The process is currently running on the CPU.
- **Waiting Process ( D ):**  
The process is waiting for an event to occur, such as user input or completion of an I/O operation.
- **Stopped Process ( T ):**  
The process has been stopped, usually by a user or by a signal.
- **Zombie Process ( Z ):**  
The process has completed execution (already dead) but still has an entry in the process table, as its parent process has not yet retrieved its exit status.



# Signals

A signal is a notification to a process that something has happened.(command sent by the system to a process)

For example, the kernel sends a signal to a process when it terminates. A process can also send a signal to another process.

Each signal has a unique number and a name. Some signals are used for specific purposes, such as the `SIGKILL` signal which is used to forcefully terminate a process. Other signals are used for more general purposes, such as the `SIGTERM` signal which is used to request that a process terminate gracefully.



# Signals (cont..)

## **SIGINT (2)**

This signal is sent to a process when the user presses the interrupt key (usually CTRL+C) in the terminal. By default, this signal terminates the process.

## **SIGTERM (15)**

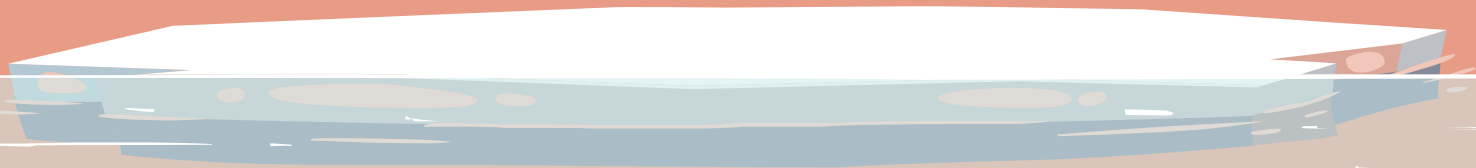
This signal is used to request that a process terminate gracefully. By default, this signal terminates the process.

## **SIGKILL (9)**

This signal is used to forcefully terminate a process. Unlike other signals, this signal cannot be ignored or caught by a signal handler.



# kill & killall Commands





# kill command

The `kill` command is used to send a signal to a process.

By default, the signal that is sent is the `SIGTERM` signal, which requests that the process terminate gracefully. However, you can specify a different signal using the `-s` option.

```
# Send the SIGTERM signal to a process with PID 1234
kill 1234

# Send the SIGKILL signal to a process with PID 1234
kill -s KILL 1234
```



# killall command

The killall command is used to send a signal to all processes with a specific name. Similar to `kill`, the signal that is sent is the SIGTERM signal, However, you can specify a different signal using the -s option.

```
# Send the SIGTERM signal to all processes with the name "firefox"
killall firefox
```

- You can see all the signal types using `kill -l` command

```
$ 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		

# Hands on







# Try the following

1. Open a new terminal process in the background.
2. In the new terminal open firefox in the background.
3. Run ps in the second terminal
4. Bring firefox into the foreground.
5. From the first terminal kill firefox processes.
6. From the second terminal kill all terminal processes.



# Package Managers





# Packages

- Linux OS is made up of 2 parts:
  - The linux kernel
  - Software packages that work with the kernel to give us a complete OS
- A package is usually referred to as an application, it can be a GUI, command line tool, etc...
- Everything other than the kernel is a package in Linux.



# Packages (cont..)

Contents of packages include:

- Binaries or the executable programs
- metadata files containing the version, dependencies, signatures and other relevant information
- documentation and manuals
- configuration files



# Packages dependencies

- Packages very rarely work by themselves, they are most often accompanied by dependencies to help them run. In Linux, these dependencies are often other packages or shared libraries. Shared libraries are libraries of code that other programs want to use and don't want to have to rewrite for themselves. If the dependencies aren't there the package will end up in a broken state and most of the time not even install.
- For example, if you need to run python-based apps, then you need to have the python present.
- Practically any software will not just have one dependency, it can have tens and sometimes even hundreds of dependency packages.



# Packages repositories

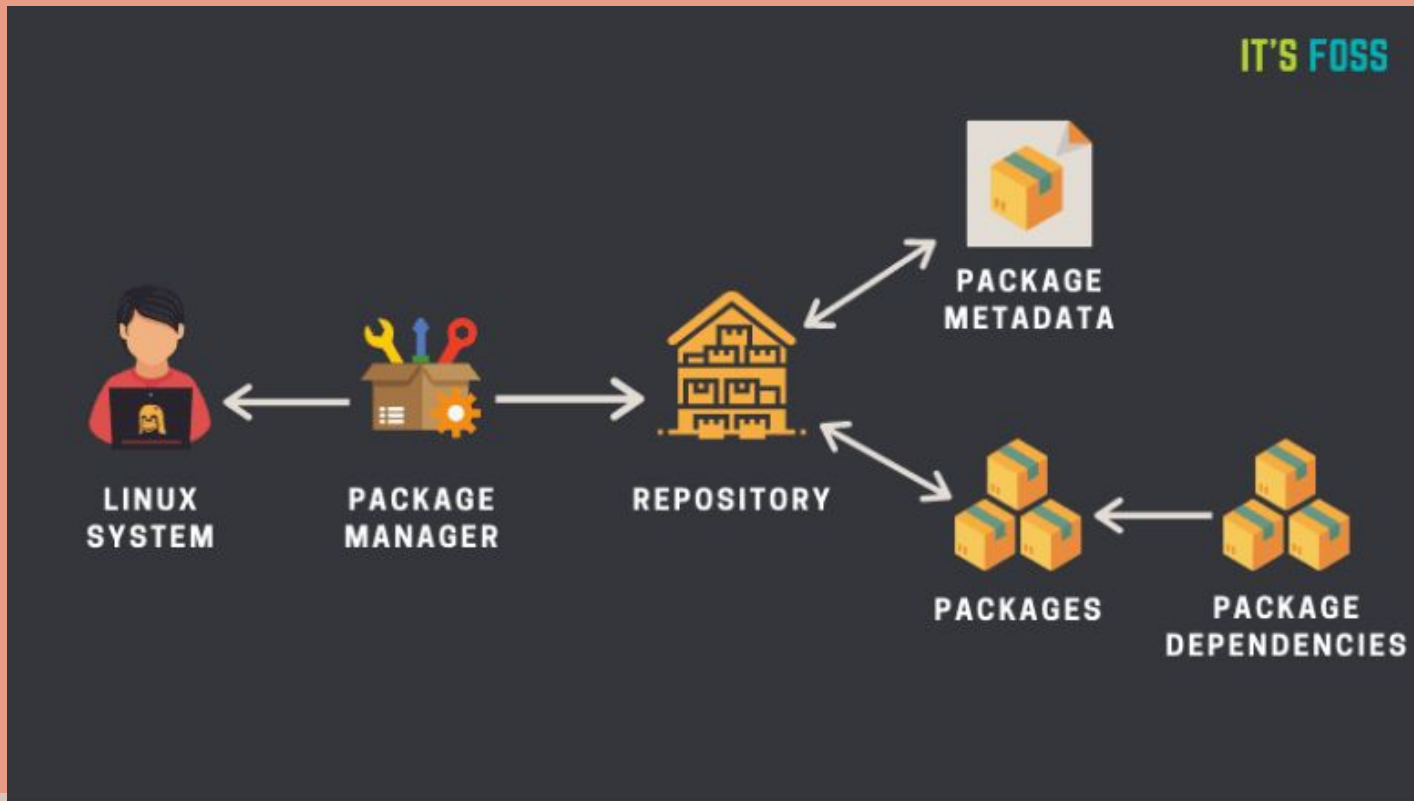
Repositories (or repos for short) are basically a place where verified packages are stored for easy retrieval by the package manager. They can be online like the YUM repository or they can be on a local folder or a DVD where you have a special collection of software that you need



# Package managers

A package manager is a tool that allows users to install, remove, upgrade, configure and manage software packages on an operating system. The package manager can be a graphical application like a software center or a command line tool like apt-get or pacman.

# Package managers







# Package managers

- Almost all linux distros have software repositories, which is a collection of packages.
- Repos also have metadata files which contain information about packages like the name, version, description, etc...
- Your system's package manager first interacts with the metadata. The package manager creates a local cache of metadata on your system, when you run the `update` command through any package manager, this local cache is updated.
- When you run the installation command of your package manager (for example `apt install [Package Name]`), the package manager refers to this cache. If it finds the package information in the cache, it uses the internet connection to connect to the appropriate repository and downloads the package first before installing on your system
- A package may have dependencies, package managers install these dependencies with the package and remove them if the package is removed.



# Package managers

- **Functions of package managers:**
  - **Automatically resolves dependencies.**
  - **Install, uninstall and update with ease.**
  - **Verifies the integrity of the package before installing it.**
  - **Verifies the architecture compatibility.**
  - **Keeps track of all the packages installed in the system so that the system administrator can easily get information about what packages are present, when was it installed, what version are they running in, etc.**



# Commands

- Most commands used with package managers are the same across different managers with the difference in the manager name, here are some examples for commands using `apt`:
  - Search for a package: `'sudo apt search package_name'`
  - Install a package: `'sudo apt install package_name'`
  - Remove a package: `'sudo apt remove package_name'`
  - Update package cache: `'sudo apt update'`
  - Upgrade all packages: `'sudo apt upgrade'`
  - Upgrade a specific package: `'sudo apt upgrade package_name'`
- It's always recommended to run `update` before `upgrade`.
- Also it's best practice to upgrade your system before installing a new package.



# Thanks!

