

System Administration (2)

IT401

Prepared by
Dr. Ahmed El-Douh

Lecture 4



System Administration 2

Lecture 4

IT401

Managing Text Files, Users, and Password Policies

Prepared by
Dr. Ahmed El-Douh

Objectives

Upon completion of this lecture, you will be able to:

- Learn how to create, view, and edit text files using command output or a text editor.
- Understand how to manage local users and groups on the system.
- Gain knowledge of local password policies, including creation, management, and deletion of users/groups.

Outlines

Part 1: Create, view, and edit text files

- Manage text files from command output or in a text editor.

Part 2: Manage local users and groups

- Create, manage, and delete local users and groups, as well as administer local password policies.

Chapter

04

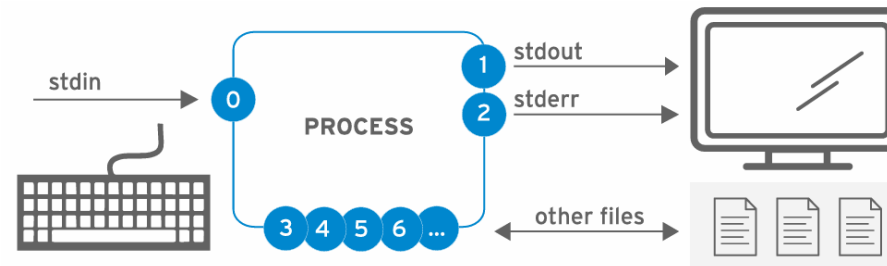
Create, view, and edit text files



Standard Input, Standard Output, and Standard Error

- A running program, or process, needs to read input from somewhere and write output either to the screen or to files. When a command is run from the shell prompt, it normally receives its input from the keyboard and sends its output to the terminal window.
- A process uses numbered channels called **file descriptors** to get input and send output.
- Every process will have at least **three file descriptors** to start with:
 1. **Standard Input (stdin)** - Channel 0: Reads input from the **keyboard**.
 2. **Standard Output (stdout)** - Channel 1: Sends normal output to the **terminal**.
 3. **Standard Error (stderr)** - Channel 2: Sends error messages to the **terminal**.

If a program opens separate connections to other files, it may use higher-numbered file descriptors (channel 3, 4, 5, etc.).



Process I/O channels (file descriptors)

Channels (File Descriptors)

NUMBER	CHANNEL NAME	DESCRIPTION	DEFAULT CONNECTION	USAGE
0	stdin	Standard input	Keyboard	read only
1	stdout	Standard output	Terminal	write only
2	stderr	Standard error	Terminal	write only
3+	<i>filename</i>	Other files	<i>none</i>	read and/or write

Redirecting Output to a File

1. Core Concept: I/O Redirection

- **Definition:** Changes the destination of a process's **output** and **errors**.
- The **process** reads from or writes to **files** instead of the **keyboard** or **terminal**.
- **Key Uses:**
 - Save **messages** and command **output** to a file.
 - **Discard** unwanted output or errors (prevent them from being displayed or saved).

2. Handling Standard Output (stdout)

- Redirecting **stdout** suppresses output from appearing on the **terminal**.
- **File Creation:** If the destination **file** doesn't exist, it is **created**.
- **Content Replacement:** If the file exists, redirection **overwrites** the file's contents (it does not append).

3. Discarding Output & Standard Error (stderr)

- To quietly discard a process's output, redirect it to the **special file: /dev/null**.
- **Crucial Point:** Redirecting *only* **stdout** does **NOT** suppress **stderr** (standard error) messages from displaying on the **terminal**.

Output Redirection Operators

USAGE	EXPLANATION	VISUAL AID
> file	redirect stdout to overwrite a file	
>> file	redirect stdout to append to a file	
2> file	redirect stderr to overwrite a file	
2>/dev/null	discard stderr error messages by redirecting to /dev/null	
> file 2>&1	redirect stdout and stderr to overwrite the same file	
&> file		
>> file 2>&1	redirect stdout and stderr to append to the same file	
&>> file		

Examples for Output Redirection

Simplify many routine administration tasks by using redirection. Use the previous table to assist while considering the following examples:

Save a time stamp in the `/tmp/saved-timestamp` file for later reference.

```
[user@host ~]$ date > /tmp/saved-timestamp
```

Copy the last 100 lines from the `/var/log/dmesg` file to the `/tmp/last-100-boot-messages` file.

```
[user@host ~]$ tail -n 100 /var/log/dmesg > /tmp/last-100-boot-messages
```

Concatenate all four `step` files into one in the `tmp` directory.

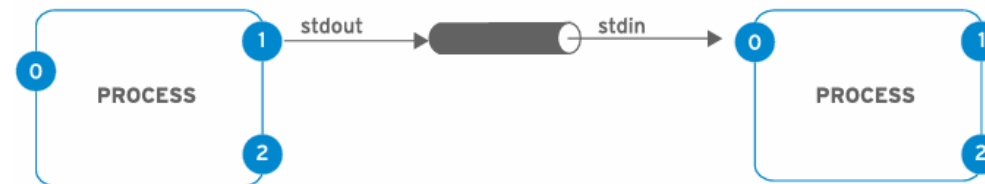
```
[user@host ~]$ cat step1.sh step2.log step3 step4 > /tmp/all-four-steps-in-one
```

List the home directory's hidden and regular file names and save the output to the `my-file-names` file.

```
[user@host ~]$ ls -a > my-file-names
```

Construct Pipelines

- **Core Concept: Data Flow → Terminal**
 - A **Pipeline** is a sequence of **two or more commands** separated by the **Vertical Bar (|)** character.
 - **Data Flow:** Data "flows" through the pipeline, where each command **manipulates or formats** the output before it reaches the **terminal**.
 - **How the Pipe (|) Works**
 - The **Pipe (|) character** connects the commands.
 - It takes the **Standard Output (stdout)** of the first command and connects it directly to the **Standard Input (stdin)** of the next command.
 - **Analogy:** Think of it as a factory conveyor belt where the output of one station becomes the raw material for the next.
- Pipelines are used to **manipulate and format** data efficiently using multiple processes **before** final output.



Process I/O piping

Pipeline Examples

```
[student@desktopX ~]$ls -l /usr/bin | less
```

The output of the **ls** command is piped to **wc -l**, which counts the number of lines received from **ls** and prints that to the terminal.

```
[student@desktopX ~]$ls | wc -l
```

In this pipeline, **head** will output the first 10 lines of output from **ls -t**, with the final result redirected to a file.

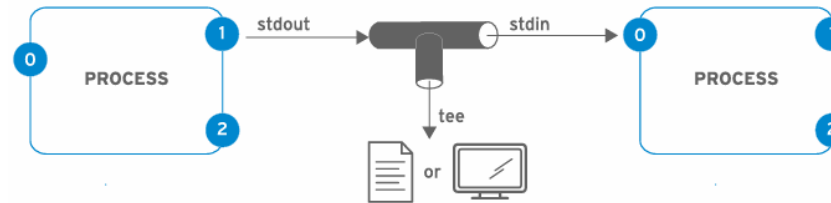
```
[student@desktopX ~]$ls -t | head -n 10 > /tmp/ten-last-changed-files
```

Pipelines, Redirection, and Tee

```
[student@desktopX ~]$ ls > /tmp/saved-output | less
```

- The Challenge: Output Interception

- When Output Redirection (>) is placed inside a Pipeline (|), the shell sends the output entirely to the file, intercepting the data flow and preventing it from reaching the next command for processing.



- The Solution: The **tee command** solves this by acting as a "T" joint: it copies the output to the file for saving while simultaneously directing it to the Standard Output, so the pipeline continues to process the data.

```
[student@desktopX ~]$ ls -l | tee /tmp/saved-output | less
```

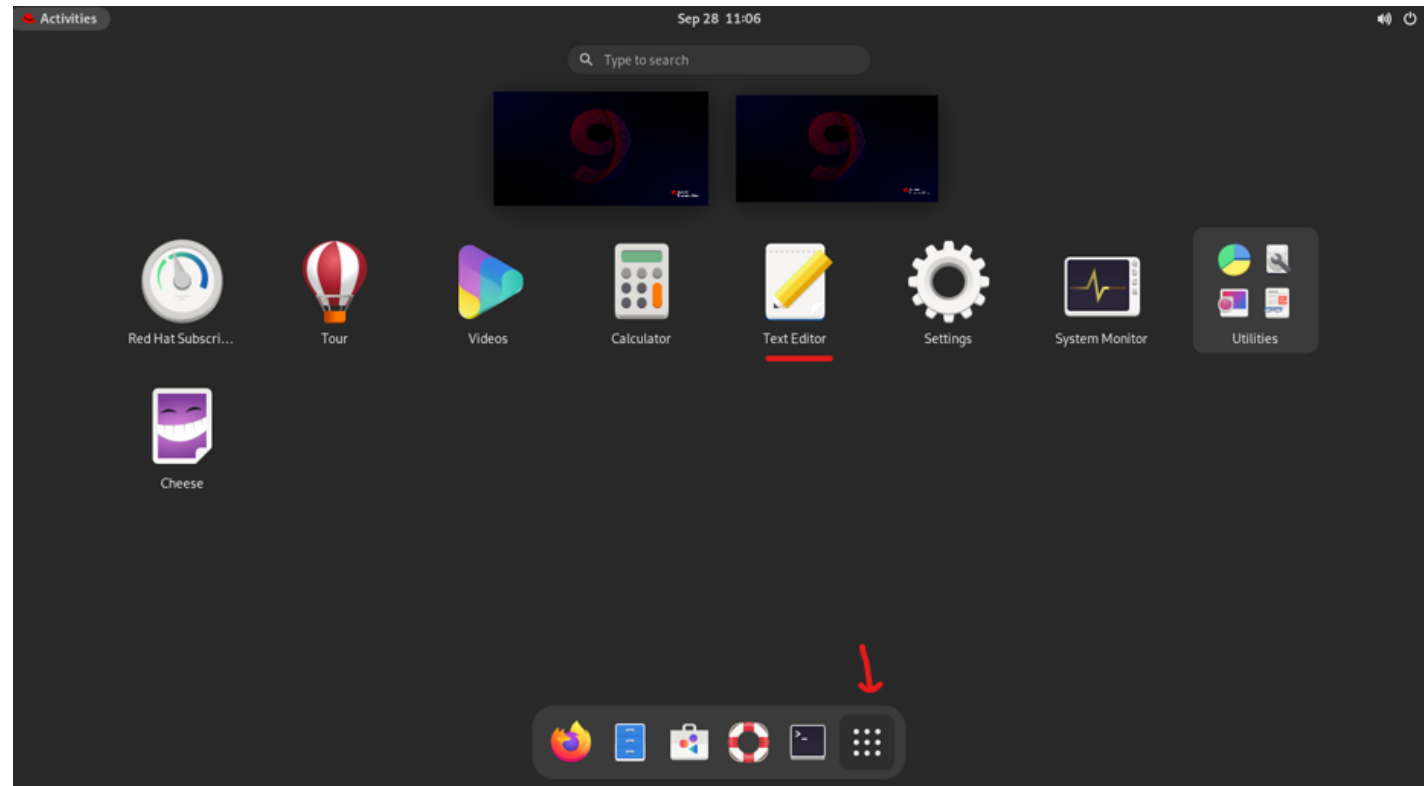
Chapter

04

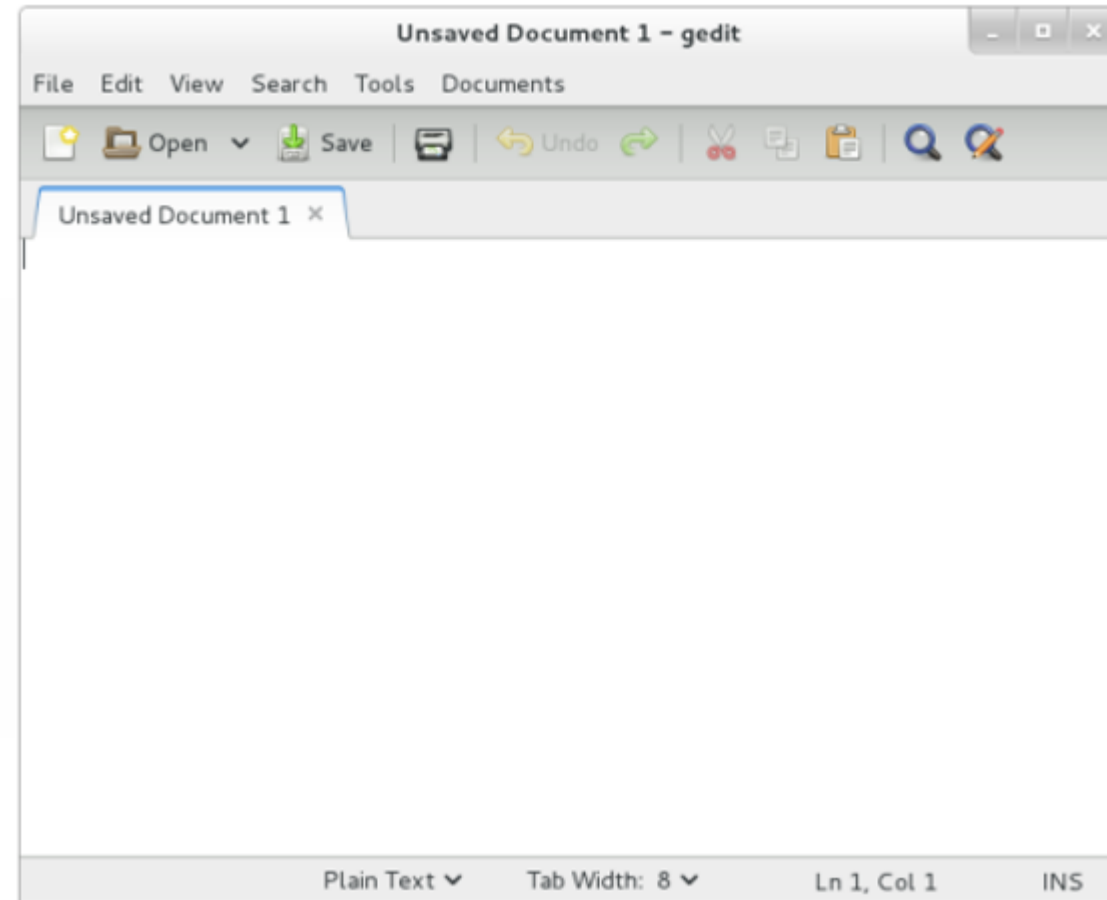
EDITING TEXT FILES WITH A GRAPHICAL EDITOR



gedit/TextEditor
Alt+F2



- ❑ **EDITING FILES WITH GEDIT** The gedit application is a full-featured text editor for the GNOME desktop environment. Launch gedit by selecting **Applications → Accessories → gedit from the GNOME menu**.
- ❑ Like other graphical applications, gedit can be started without navigating the menu. Press **Alt+F2** to open the Enter a Command dialog box. Type gedit and press Enter.



Chapter

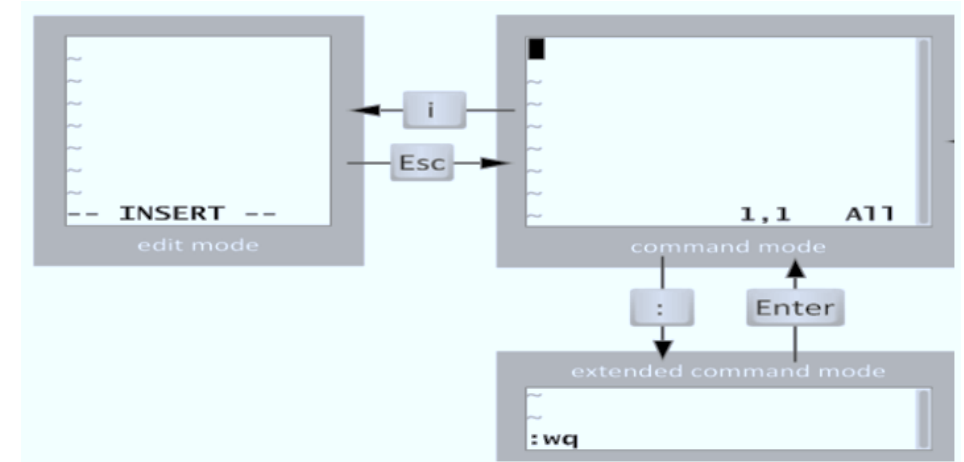
04

EDITING TEXT FILES FROM THE SHELL PROMPT

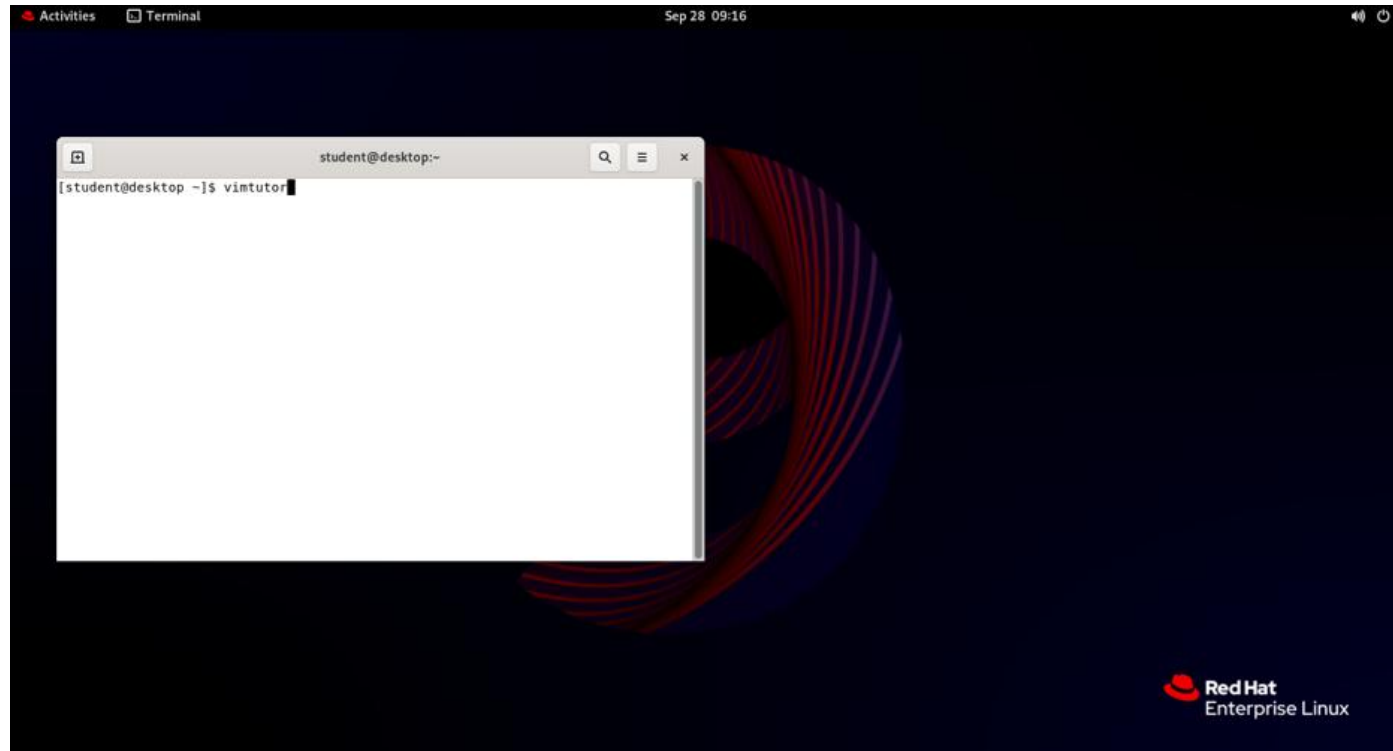


Edit Files with Vim

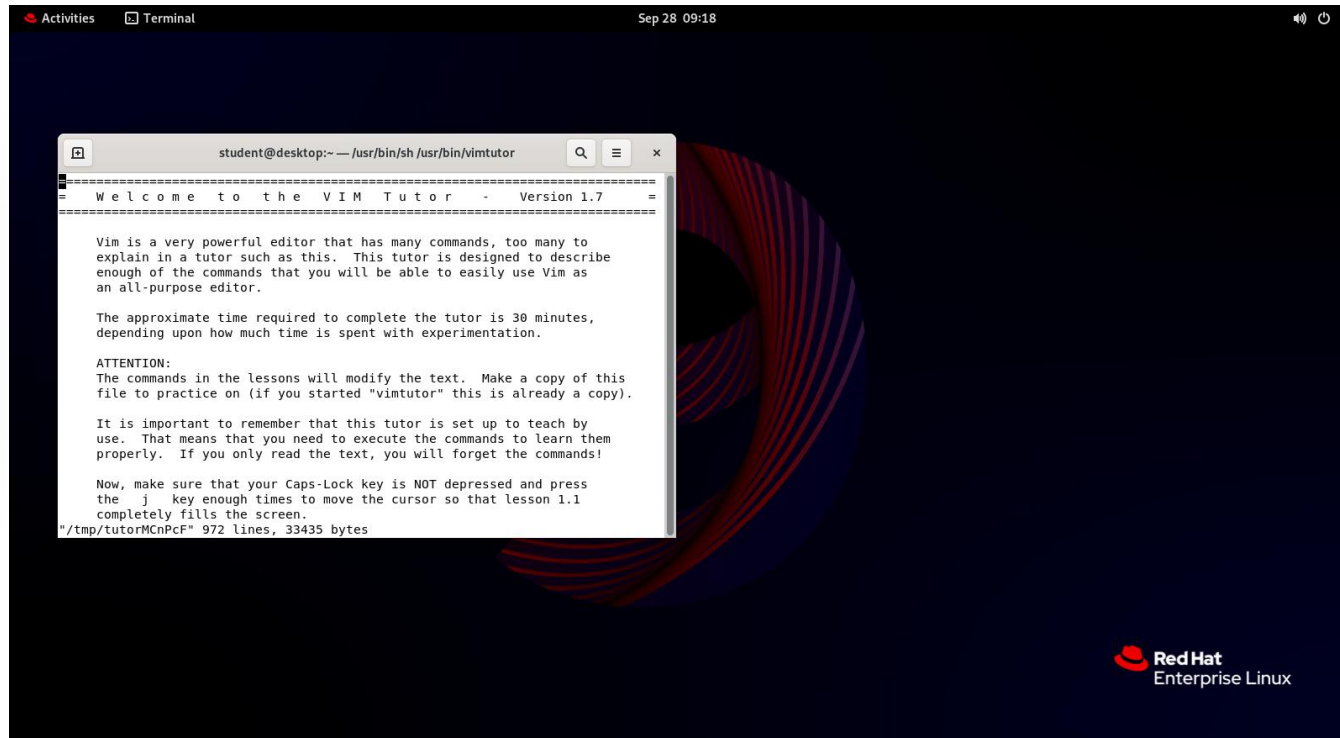
- **Linux File Principle:** Information is primarily stored in text-based files (including flat files and XML files), allowing them to be easily edited and shared without conversion.
- **Vim Definition:** Vim is an improved and efficient version of the standard vi text editor. **Key Features:** It is highly configurable and includes features like split screen editing, color formatting, and syntax highlighting.
- **Vim Modes** (See Diagram):
 - **Command Mode:** The default mode for navigation and issuing commands.
 - **Insert Mode (Edit Mode):** Entered by pressing `i`, used for typing and editing text. Exited by pressing `Esc`.
 - **Extended Command Mode:** Entered from Command Mode by pressing `:` (colon), used for saving, quitting, and file operations (e.g., `:wq` to write and quit).



Vimtutor



Vim → Terminal → :shell
Terminal → Vim → Exit , then Enter



The screenshot shows a terminal window titled "Terminal" with a dark background. A window titled "student@desktop:~ — /usr/bin/sh /usr/bin/vimtutor" is open, displaying the Vim tutor. The tutor text includes a welcome message, an introduction to Vim, an estimate of 30 minutes to complete, an attention warning about modifying text, and instructions to press the 'j' key. The bottom of the window shows the file path "/tmp/tutorMCnPcF" and its size "972 lines, 33435 bytes". The Red Hat Enterprise Linux logo is visible in the bottom right corner of the terminal window.

```
student@desktop:~ — /usr/bin/sh /usr/bin/vimtutor
=====
W e l c o m e   t o   t h e   V I M   T u t o r   -   V e r s i o n   1 . 7
=====

Vim is a very powerful editor that has many commands, too many to
explain in a tutor such as this. This tutor is designed to describe
enough of the commands that you will be able to easily use Vim as
an all-purpose editor.

The approximate time required to complete the tutor is 30 minutes,
depending upon how much time is spent with experimentation.

ATTENTION:
The commands in the lessons will modify the text. Make a copy of this
file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by
use. That means that you need to execute the commands to learn them
properly. If you only read the text, you will forget the commands!

Now, make sure that your Caps-Lock key is NOT depressed and press
the  j  key enough times to move the cursor so that lesson 1.1
completely fills the screen.

"/tmp/tutorMCnPcF" 972 lines, 33435 bytes
```

Insert & Replace insert mode	
i	insert text before the cursor
a	insert text after the cursor
I	insert text at the beginning of the current line
A	insert text at the end of the current line
o	insert text in a new line below the cursor
O	insert text in a new line above the cursor
r	replace a character at the cursor position
R	replace characters starting at the cursor position

Undo, Delete, and Copy edit commands	
u	undo last change
CTRL+r	redo last undo change
mA	set mark A to current position. Marks can be the letter [a-z] and [A-Z]
y'A	yank (copy) from current line to mark A
d'A	delete from current line to mark A
P	put the buffer content before the cursor
p	put the buffer content after the cursor
x	delete character at cursor position
dW	delete first word after cursor position
d\$ or D	delete from cursor position to end of line
dd	delete current line
J	join current line with following line

Search & Substitute	patterns
<code>/string</code>	search forward for string
<code>?string</code>	search backward for string
<code>n</code>	repeat the last search
<code>:\$s/str1/str2/gc</code>	search and substitute str1 by str2 from the current line to the end of file. A line number can be specified before the comma “,”. The “c” ask for confirmation before each substitution
<code>:%s/str1/str2/g</code>	replace all str1 by str2 in all the file without prompting for confirmation
<code>:%s/str1/str2/</code>	replace str1 by str2 on the first occurrence in each line of the file

Ex Commands	vi features
<code>:q</code>	quit current open file
<code>:q!</code>	force to quit without saving changes
<code>:w</code>	save file
<code>:w file</code>	save file as file
<code>:wq!</code>	force to save file then quit
<code>:x,yw file</code>	write from line x to line y into file
<code>:w » file</code>	append buffer to file

Chapter

04

Manage Local Users and Groups



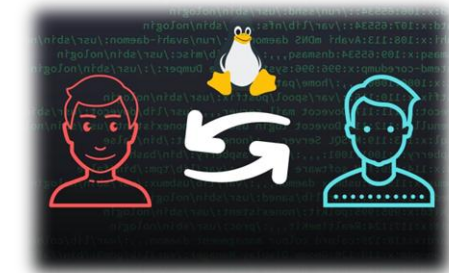
Linux Users and Groups

- ❑ Linux is a **multi-user** operating system that provides robust mechanisms for managing *user access, permissions, and security* through its user and group system.
- This fundamental aspect of Linux administration ensures that multiple users can ***safely*** and ***securely*** share system resources while maintaining appropriate levels of access control and data protection.

Linux Users and Groups

❑ Fundamental entities for identity and access management in Linux

- **Users**
- **Groups**



Linux Users and Groups

The concept of **users** and **groups** in Linux serves several critical purposes.

- First, it provides **identity management** by uniquely *identifying* each person or process that interacts with the system.
- Second, it enables **access control** by determining what **resources** each user can access and what operations they can perform.
- Third, it **facilitates resource allocation** by allowing administrators to assign system resources to specific users or groups of users.
- Finally, it supports **accountability** by maintaining logs and audit trails that track user activities on the system.

Linux Users and Groups

Users

A **user** in Linux is an account that can log into the system and perform operations. Each user has:

- **Username:** Human-readable identifier
- **User ID (UID):** Numeric identifier assigned by the system
- **Home directory:** Personal directory (usually `/home/username`)
- **Shell:** Command interpreter (like `/bin/bash`)
- **Password:** Authentication credential

Linux Users and Groups

Types of Users

- **Root User:** Superuser with UID **0**, has complete system access
- **System Users:** Used by system **services** and daemons (UIDs typically **1-999**)
- **Regular Users:** Normal user accounts for people (UIDs typically **1000+**)

Linux Users and Groups

System Users

- also known as **service accounts**, are created for system *processes*, *daemons*, and *services* that need to run with specific privileges. These accounts typically have UIDs below 1000 and are not intended for human login.
- Examples include the **www-data** user for web servers, the **mysql** user for database services, and the mail user for **mail services**. System accounts often have restricted shells or no shell at all, preventing direct login while still allowing processes to run under their identity.

Linux Users and Groups

WHAT IS A USER?

- **Process Identity:** Every **process** (running program) on the system runs as a **particular user**.
- **File Ownership:** Every **file is owned by a particular user**.
- **Access Control:** Access to files and directories are **restricted by user**. The user associated with a running process determines its accessible files and directories.

THE **id** COMMAND

- The **id command** is used to show information about the **current logged-in user**.
- **Other Users:** Basic information about another user can be requested by passing in the **username** as the first argument to the command.

```
[student@desktopX ~]$ id
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Linux Users and Groups

VIEWING PROCESS INFORMATION (**ps** COMMAND)

1. The **ps** Command (Process Status)

- The **ps** command is used to view process information.
- **Default:** Shows only processes in the **current shell**.

```
[student@desktop ~]$ ps
  PID TTY          TIME CMD
 2730 pts/0        00:00:00 bash
 2952 pts/0        00:00:00 ps
```

2. Key Options for Process Listing

Option	Function	Purpose
a	All	View all processes with a terminal (belonging to all logged-in users).
u	User Format	Includes detailed User information associated with the process.

```
[student@desktop ~]$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
student   1966  0.0  0.4 374120  7976 tty2    Ssl+  09:17   0:00 /usr/libexec
student   1979  0.1  1.1 521348 19752 tty2    Sl+   09:17   0:00 /usr/libexec
student   2730  0.0  0.3 224112  5664 pts/0    Ss    09:17   0:00 bash
student   2813  0.0  0.2 225500  3748 pts/0    R+    09:17   0:00 ps au
```

Linux Users and Groups

USER ACCOUNT INFORMATION (/etc/passwd)

1. Internal User Tracking

- **Users vs. IDs:** While commands display names (e.g., *student*), the OS internally tracks users by a **UID number**.
- **Database:** The mapping of names to numbers is stored in account information databases.
- **Location:** For local users, this information resides in the simple "flat file," **/etc/passwd**.

2. The **format** of **/etc/passwd** file is divided into **seven** colon-separated fields.

```
1 username: 2 password: 3 UID: 4 GID: 5 GECOS: 6 /home/dir: 7 shell
```

- 1 *username* is a mapping of a UID to a name for the benefit of human users.
- 2 *password* is where, historically, passwords were kept in an encrypted format. Today, they are stored in a separate file called **/etc/shadow**.
- 3 *UID* is a user ID, a number that identifies the user at the most fundamental level.
- 4 *GID* is the user's primary group ID number. Groups will be discussed in a moment.
- 5 *GECOS* field is arbitrary text, which usually includes the user's real name.
- 6 */home/dir* is the location of the user's personal data and configuration files.
- 7 *shell* is a program that runs as the user logs in. For a regular user, this is normally the program that provides the user's command line prompt.

Linux Users and Groups

Groups

Group accounts in Linux provide a powerful and flexible mechanism for managing permissions and access control for multiple users simultaneously.

Instead of assigning permissions to individual users, which can become cumbersome in environments with many users, **groups** allow administrators to define permissions once for a group and then assign users to that group. This simplifies administration, enhances security, and improves scalability.

Linux Users and Groups

Groups

A **group** is a collection of users that share certain permissions. Each group has:

- **Group name:** Human-readable identifier
- **Group ID (GID):** Numeric identifier
- **Member list:** Users belonging to the group

Types of Groups

- **Primary group:** Every user has one primary group (usually matches username)
- **Secondary groups:** Additional groups a user can belong to

Key Files

- **/etc/passwd:** Contains user account information
- **/etc/group:** Contains group information
- **/etc/shadow:** Contains encrypted passwords (restricted access)

Linux Users and Groups

- **Gaining Superuser Access:** means obtaining the **privileges** of the **root** user, with complete access to all commands and files. This **level of access** is necessary for administrative tasks but also carries significant **risks**.

- **The Root User: Power and Responsibility:**

1. The Risk of Unlimited Power

- The root user has unlimited power in the system.
- This power brings responsibility; the user can damage the system (e.g., remove files or user accounts).
- If the root account is compromised, the attacker gains administrative control of the entire system.

2. Best Practice for Administrators

- Recommendation: Always log in as a normal user (unprivileged account).
- Escalate privileges to root only when needed to perform administrative tasks (e.g., using the sudo command).

Linux Users and Groups: Managing Privileges (su vs. sudo)

❖ Switching Users with **su**

```
[student@desktop ~]$ sudo usremod -l student  
[sudo] password for student: █
```

- **Purpose:** Allows a user to **switch** to a different **user account** (e.g., become **root**).
- **Implication:** If no **username** is specified, the **root account** is **implied**.
- **Authentication:** Requires the **password** of the **target account** (usually the **root password**).
- **Drawback:** Sharing the **root password** is a major **security disadvantage**.

❖ Running Commands as Root with **sudo** (Recommended)

- **Goal:** Permits a user to execute a specific **command** with **root privileges** (or another user's privileges).
- **Configuration:** Access is controlled by settings in the **/etc/sudoers** file.
- **Authentication:** Requires the **user's own password** for validation. **It does not require the root password.**
- **Benefit:** Enables **administrators** to grant **fine-grained permissions** to specific users for specific tasks.
- **Security:** Enhances **system security** by avoiding the distribution of the **root password**.

Linux Users and Groups

Basic syntax: sudo command

- **sudo -u username command**

```
[student@desktop ~]$ sudo -u
sudo: option requires an argument -- 'u'
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
[command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-D directory] [-g group]
[-h host] [-p prompt] [-R directory] [-T timeout] [-u user]
[VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-D directory] [-g group]
[-h host] [-p prompt] [-R directory] [-T timeout] [-u user] file ...
```

- **Open root shell: `sudo -i` or `sudo -s` # without using root password**

```
[student@desktop ~]$ sudo -i
[sudo] password for student:
[root@desktop ~]# exit
logout
[student@desktop ~]$
```

- **Check available permissions: `sudo -l`**

```
[student@desktop ~]$ sudo -l
Matching Defaults entries for student on desktop:
!visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
env_reset, env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS",
env_keep+="MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE",
env_keep+="LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES",
env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE",
env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
secure_path="/sbin:/bin:/usr/sbin:/usr/bin
```

User student may run the following commands on desktop:
(ALL) ALL

```
[student@desktop ~]$ sudo -s
[root@desktop student]#
```

Linux Users and Groups

(Using Command-Line Tools for User Accounts)

I. Creating Users: useradd

Basic Syntax: **useradd username** —————→ **sudo useradd ali**

```
[student@desktop ~]$ useradd ali
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
[student@desktop ~]$ sudo useradd ali
[sudo] password for student:
```

- Creates the user with **default settings** (read from /etc/login.defs).
- Does **NOT** set a valid **password** by default.
- The user **cannot log in** until a password is set (using the passwd command).

```
[student@desktop ~]$ sudo passwd ali
[sudo] password for student:
Changing password for user ali.
```

II. Checking Options & Help

useradd - -help

- Displays **options** to override system defaults (e.g., set a specific home directory or shell).
- Most of these options can also be used with **usermod** to modify existing users

Linux Users and Groups

1. Inspecting Existing Groups

- To **list all groups** currently defined on the system, use the **getent group** command.
- *This command checks the system database to show you all existing group names.*

```
[student@desktop ~]$ getent group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
kmem:x:9:
wheel:x:10:eldouh,student
```

2. Add User to the Group: To add the user **ali** to an existing supplementary group (e.g., **Sys**): **sudo usermod -aG Sys ali**

```
[student@desktop ~]$ sudo usermod -aG sys ali
[sudo] password for student:
[student@desktop ~]$ id ali
uid=1002(ali) gid=1003(ali) groups=1003(ali),3(sys)
```

3. Verify Membership: Check that **ali** is now a member of the group: **id ali**

4. Deleting Users and Groups:

- **Delete the User and their Data:** To remove the user (e.g., **ali**) and delete their **home directory** along with all files: **sudo userdel -r ali** → The **-r (recursive)** flag ensures the user's home directory is **removed**.
- **Delete the Group:** To permanently remove a group (e.g., **Sys**): **sudo groupdel Sys**
 - **Note:** You cannot delete a group if it is currently set as the primary group for any user.



Thank You