What is FOSS?

Day 1

- Free/Open Source Software (FOSS) provides many freedoms, including the ability to:
  - View the source code used to compile programs.
  - Make modifications.
  - Distribute these modifications.
- Most FOSS is covered under a public license. The most common public license is the GNU General Public License(GPL).

FOSS Licenses

- An open-source license is a type of license for computer software and other products that allows the source code, blueprint or design to be used, modified and/or shared under defined terms and conditions.

- Examples: GPL, LGPL, Apache, Mozilla Public License and BSD.

Linux History

- Multics was developed in the 1963-1969 period through the collaboration of the Massachusetts Institute of Technology (MIT), General Electric,and Bell Labs.
- Unix first version created in Bell Labs in 1969.
- Unix flavors:
  - IBM->AIX
  - Hewlett-Packard->HP/UX
  - Sun-> Solaris
  - Silicon Graphics->IRIX
- Operate in a same manner.
- Offer the same standard utilities and commands.

Linux History

- In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX-like operating system.
  - GNU General Public License (GPL).
  - Free Software Foundation (FSF).
- In 1991, Linus Torvalds created Linux kernel.
- In 1992, Linux and GNU developers worked to integrate GNU components with Linux to make a fully functional and free operating system.

Linux distributions

From slides

Why Linux?

● Linux is open source.
● Linux is released under the GNU General Public License (GPL).
● Linux is secure and virus free.
● Linux is perfect for programmers.
● Linux has a better community support.

Why Linux?

● Linux is growing in the home users sector and the dominant of the professional and servers sector.
● Internet service providers (ISPs), e-commerce sites, and other commercial applications all use Linux today and continue to increase their commitment to Linux.

Red Hat

● Red Hat was founded on March 26, 1993.
● Red Hat Linux first appeared in 1994.

Red Hat Enterprise   fedora

Red Hat

● More than 90% of Fortune Global 500 companies use Red Hat products and solutions*.
● The most demanding applications run better on Red Hat Enterprise Linux.
● RHEL scales well, and is more reliable.
● RHEL is secure.
● Red Hat partnership with hardware vendors.
● Red Hat training and support.Linux

Red Hat

● Companies using red hat slide 12

Types of installation

● Automated installation.
● Graphical installation.
● Remote installation.

Linux Components

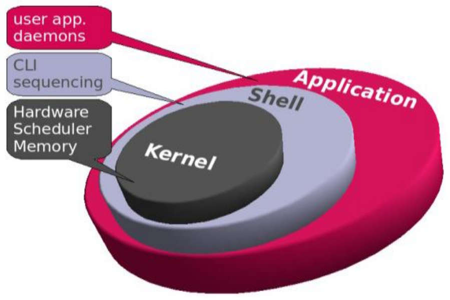
● Kernel
- Is the core of the operating system.
- Contains components like device drivers.
- It loads into RAM when the machine boots and stays resident in RAM until the machine powers off.

Linux Components

● Shell
- Provides an interface by which the user can communicate with the kernel.
- “bash” is the most commonly used shell on Linux.
- The shell parses commands entered by the user and translates them into logical segments to be executed by the kernel or other

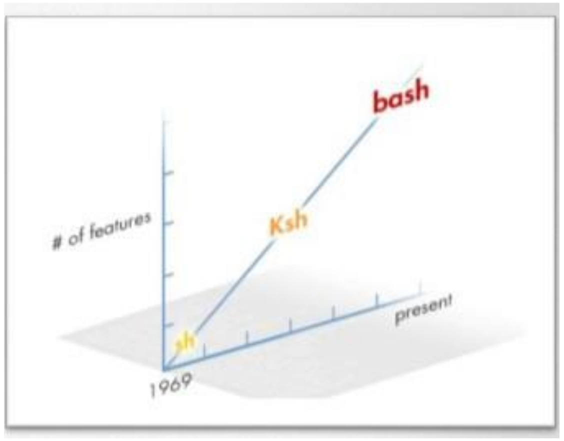
Linux Components

- Gives the shell a place to accept typed commands and to display their results.



Command-Line Shells

● There are lot of shells as :
Bourn Shell (sh),
Korn Shell (ksh),
C Shell (csh) and
Bourn Again Shell (bash).
● They have different features that will be discussed later.



Running Commands

● Commands have the following syntax: command [options] [arguments]
● Each item is separated by a space.
● Options modify the command's behavior.
● Arguments are files name or other information needed by the command.
● Separate commands with semicolon (;).

Examples

print system information

● **uname**
Linux
● **uname -n**
host1
● **uname -a**
Linux host1 ....
● **cal**
To get the calender
● **cal 9 2020**
cal [month] [year]

● **date**
print or set the system date and time
● **whoami**
root
print effective userid

Directories

● Think of
- File system as a building, Directory is a room, File is a desk

● The current working directory is the room you are.
● To find out where you are at any time.
● **pwd**
/home/guest
print name of current/working directory

File System

Picture slide 25, 26 (fatma said that “study it from internet”
● Pathnames
- Absolute pathname
- Relative pathname

Changing Directories

● To move from directory to directory on the system.
cd /home/user1/work
cd ..
cd ~
cd -

Listing Directory Contents

ls [option] [argument]

● **ls**
dir1 dir2 file1
● **ls /home/user1/dir1**
f1 f2
● **Pwd**
/home/user1
● **ls dir1**
f1 f2

Listing Directory Contents

ls [option] [argument]

● **ls -a dir1**
. .f1 f1
.. .f2 f2
● **ls -l dir1**
total 2
-rw-r--r-- 1 fatma fatma 20 2 May 21 16:11 f1
-rw-r--r-- 1 fatma fatma 20 0 May 21 16:11 f2
● **ls -F**
dir1/ dir2/ file1
dir3/ file2* file3@

Listing Directory Contents

ls [option] [argument]

● **ls -ld dir1**
drwxr-xr-x 2 fatma fatma 51237 May 29 16:06 dir1
-:file d:directory c:? b:? S:? P:? l:?

Listing Directory Contents

- **ls -R**

```
. :
dir1 dir2 file1
dir3 file2 file3
./dir1:
f1 f2
./dir2:
./dir3:
```

File Naming

- File names may be up to 255 characters.
- There are no extensions in Linux.
- Avoid special characters as >< ? * # '.
- File names are case sensitive.

Viewing File Content

- **cat fname**
- **more fname**

  Scrolling keys for the more command

  Spacebar: moves forward on screen.
  Return: scroll one line at a time.
  b: move back one screen.
  /string: search forward for pattern.
  n: find the next occurrence.
  q: quit and return to the shell prompt.
- **head -n fname**
- **tail -n fname**

File Globing

- When typing commands, it is often necessary to issue the same command on more than one file at a time.
- The use of wildcards, or "metacharacters", allows one pattern to expand to multiple filenames.

Metacharacters

- **Asterisk(*): represents 0 or more character, except leading(.)**

Example:

```
ls f*
file.1 file.2 file.3 file4
file1 file2 file3 fruit
ls *3
file.3 file3
dir3:
Moon planets space sun
```

Metacharacters

- Question mark(?) character represents any single character except the leading (.)

Example:

```
ls file?
file4 file1 file2
ls z?     Z?:No such file or directory
```

Metacharacters

- Square bracket([]): represent a range of characters for a single character position.

Example

```
ls [a-f]*
ls [pf]*
```

Metacharacters

```
ls -a
. .. .profile abm bam bat battle project
ls -l b*
-rw-r----- 1 sgs 16 Feb 12 11:04 bam
-rw-r----- 1 sgs 12 Feb 12 11:05 bat
-rw-r----- 1 sgs 19 Feb 12 11:06 battle
ls *
abm bam bat battle project
ls .*
. .. .profile
ls *m
abm bam
```

Metacharacters

```
ls ???
abm bam bat
ls ?a?
bam bat
ls ?a*
bam bat battle
ls *a*
abm bam bat battle
```

Metacharacters

```
ls [ab]*
abm bam bat battle
ls -l [ab]m
ls: "[ab]m: No such file or directory
ls [a-zA-Z]*
abm bam bat battle project
```

File & Dir. Manipulation

- **Coping Files and Directories**

  Cp [options] source(s) target

Option	Description
-i	Prevents you from accidentally overwriting existing files or directories
-r	Copy a directory including the contents of all subdirectories

File & Dir. Manipulation

- **Coping Files and Directories**

  cp [options] sfile1 sfile2 sfile3 targetDest_directory

Option	Description
-i	Prevents you from accidentally overwriting existing files or directories
-r	Copy a directory including the contents of all subdirectories

File & Dir. Manipulation

● **Moving and Renaming Files and Directories**

mv [options] source(s) target

Option	Description
-i	Prevents you from accidentally overwriting existing files or directories

File & Dir. Manipulation

● To create files

touch file(s)_name

● To create directories

mkdir [-p] dir(s)_name

File & Dir. Manipulation

● To remove files

rm [-i] file(s)_name

● To remove directories

rmdir dir(s)_name

rm [-r] dir(s)_name

Linux Documentation

Manual page consists of:

● **Name**

The name of the command and a one-line description.

● **Synopsis**

The syntax of the command.

● **Description**

Explanation how the command works and what it does.

● **Files**

The file used by the command.

● **Bugs**

Known bugs and errors.

● **See also**

Other commands related to this one.

Linux Documentation

● Shows the commands that have manual pages that contains any of the given keywords.

man -k keyword

man -s keyword

● Shows the commands one line description.

whatis command

Linux Documentation

--help Option

● Another way to get help about a command.

● help is built in the command itself (if supported).

Interrupting Execution

● To interrupt a command that's taking too long to execute, use [Ctrl]-c.

● Occasionally, you might enter a command without an argument that expects input to come from the keyboard. In this case, use [Ctrl]-d to terminate the command.

Day 1 Contents

● Free/Open Source Software and Licenses.

● Linux History.

● Installation.

● Linux Components.

● Basic Commands.

● File and Directory Basics.

● Linux Documentation.

Day 2

Day 2 Contents

- User and group administration.
- Permissions.
- Switching to other accounts.
- Shutting down the system.

Listing Directory Contents

- **ls -l dir1**

```
-rwxr-xr-x 2 root root 20 512 May 21 16:06 file1
drwxr-xr-x 2 fatma fatma 20 512 May 21 16:06 dir2
```

Permission Owner Group

User Accounts

- Root user (Adminstrator).
- Normal user.
- Service user.

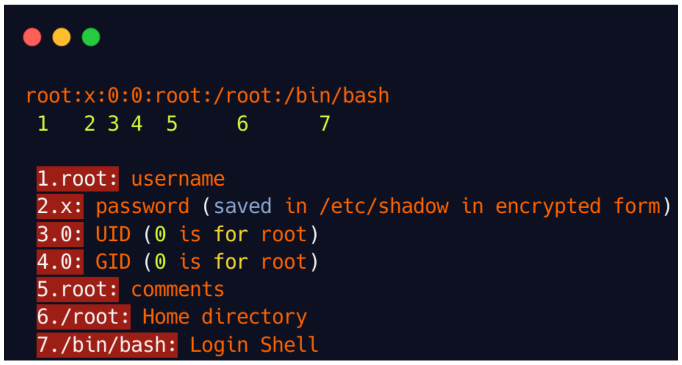
Users and Groups

- The **/etc/passwd** file

login-name:x:uid:gid:comment:home-directory:login-shell

Included fields are:

- Login name.
- Encrypted password.
- User Id (uid).
- Group Id (gid).
- Comment about the user.
- Home Directory.
- Login shell.



```
root:x:0:0:root:/root:/bin/bash
 1   2 3 4  5     6      7

 1.root: username
 2.x: password (saved in /etc/shadow in encrypted form)
 3.0: UID (0 is for root)
 4.0: GID (0 is for root)
 5.root: comments
 6./root: Home directory
 7./bin/bash: Login Shell
```

Users and Groups

- The **/etc/shadow** file

username:encrypted passwd:last changed: min:max warn:inactive:expire:future-use

Included fields are:

- Login name.
- Encrypted password.
- Days since Jan 1, 1970 that password was last changed.
- Days before password may not be changed.
- Days after which password must be changed.
- Days before password is to expire that user is warned.
- Days after password expires that account is disabled.
- Days since Jan 1, 1970 that account is disabled.

Users and Groups

- The **/etc/group** file.

groupname:x:gid:comma-separated list of group members

- The /etc/gshadow file???

Adding New User

useradd [options] username

- passwd username
  - The **useradd** command populates user home directories from the **/etc/skel** directory.
- To view and modify default setting:
  useradd -D
- Adding multiple user accounts:
  newusers filename

Creating New Group

groupadd [options] groupname

- Linux users can be a member of two different kinds of groups.
  Primary group Secondary group
- Every user must be a member of **only one "private" primary group.**
- This primary group has the same name as the user's username.
- Every user can be a member of **one or more secondary groups.**
- Use the **-r** option to the groupadd command avoids using a GID within the range typically assigned to users and their private groups.

Adding New User

Example

useradd -u 1003 -g 1003 -c “comment” -md /home/user1 -s /bin/bash user1

passwd user1

id ???

groups ???

Modifying User Accounts

- To change a user's account information, you can:
  - Edit the **/etc/passwd** or **/etc/shadow** files manually.
  - Use the **usermod** or **chage** commands.

Modifying User Accounts

- The **usermod** command can be used to set all properties of users as stored in **/etc/passwd** and **/etc/shadow**, plus some additional tasks, such as **managing group membership**.
- There is just one task it does not do well: setting **passwords**.

Users and Groups

- **usermod [options] username**

Options

- To change the login name use **-l** <login name>
- To lock the password use **-L**
- To unlock the password use **-U**
- To add new secondary group use **-aG**

Modifying an Existing Group

groupmod [options] groupname

The **groupmod** command can be used to change the **name** or **group ID** of
the group, but it does not allow you to add group members.

Options

- To changes the groupname use **-n**
- To changes group ID use **-g**

Group membership

- **groupmems -g group1 -l**
- The groupmems command can be used to see whic users are a member of group1 .

Deleting A User Accounts

- To delete a user account, you can:
- Manually remove the user from:
- /etc/passwd file.
- /etc/shadow files.
- /etc/group file.
- remove the user's home directory (/home/username).
- and mail spool file (/var/spool/mail/username).

Deleting A User Accounts

- To delete a user account, you can:

  userdel -r username

Options

- -r: It will remove user's home directory and the user's mailspool. Files located in other file systems will have to be searched for and deleted.

Deleting A Certain Group

- To delete a certain group, you can:

  groupdel groupname
- To List all file which are owned by groups not defin in /etc/group file

  find / -nogroup

Password Aging Policies

- The chage command sets up password aging.

  chage [options] username

Options

- To change the min number of days between password changes use **-m**
- To change the max number of days between password changes use **-M**
- To change the expiration date for the account use **-E date**
- To change the number of days to start warning before a password change will be required use **-W**
- To show password expiry information use **-l**

Password Aging Policies

- The passwd command updates authentication tokens.

  passwd [options] username

Options

- To change the min number of days between password changes use **-n**
- To change the max number of days between password changes use **-x**
- To change the expiration date for the account use **-i**
- To change the number of days to start warning before a password change will be required use **-w**
- To lock the password use **-l**
- To unlock the password use **-u**

Switching Accounts

- **su [-] [username]**
- **su [-] [username] -c command**

The whoami Command

- After switching into several users, it is a sever issue to know your current (effective) user
- **whoami**

```
root
```

The id Command

- **Displays**
  - Effective user id.
  - Effective user name.
  - Effective group id.
  - Effective group name.
- **Example**

id user1

```
uid=101(user1) gid=100(user1)groups=101(user1)
```

The who Command

- **Who is on the system.**
- **Displays**
  - User Login name .
  - Login device(tty).
  - Login date and time.
- **Example** who

The w Command

- The **w** command display a summary of the current activity on the system, including what each user is doing.
- W [user]
- **Example**

```
w
```

Using sudo Command

Ownership and Permissions

- Every file and directory has both **user** and **group** ownership. A newly-created file will be owned by:
  - The user who creates it.
  - That user's primary group.

Ownership and Permissions

- File ownership can be changed using **chown** command.
- Example:

```
chown user1 file1
chown user1:group1 file1
chown :group1 file1
chown -R user2 dir1
```

Security Scheme

- Each file has an owner and assigned to a group.
- Linux allows users to set permissions on files and directories to protect them.
- Permissions are assigned to:

➢ File owner.

➢ Members of the group the file assigned to.

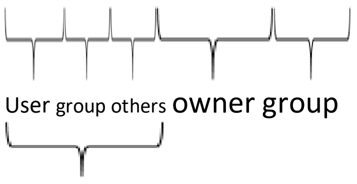
➢ All other users.

- The most specific permissions apply.
- Permissions can only be changed by the owner and root.

Listing Directory Contents

- **ls -l dir1**

```
-rwxr-xr-x 2 root root 20 512 May 21 16:06 file1
drwxrwxrwx 2 fatma fatma 20 512 May 21 16:06 dir2
```



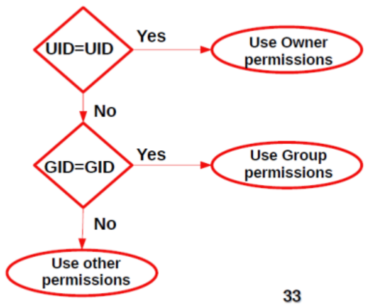
User group others owner group

permission

Permission Notations

Permission	Access for a File	Access for a Directory
Read	You can display file contents and copy the file.	You can list the directory contents with the ls command.
Write	You can modify the file contents.	If you also have execute access, you can add and delete files in the directory.
Execute	You can execute the file if it is an executable. You can execute a shell script if you also have read and execute permissions.	You can use the cd command to access the directory. If you also have read access, you can run the ls -l command on the directory to list contents.

Determining Permissions



33

Changing Permissions

- chmod permission filename
- Permissions are specified in either
  Symbolic mode

Who	Operator	Permissions
u: Owner permissions	+ Add permissions	r: read
g: Group permissions	- Remove permissions	w: write
o: Other permissions	= Assign permissions absolutely	x: execute
a: all permissions		

To change the file permissions for an existing file or directory.

```
chmod u=symbolic_value,g+symbolic_value,o-symbolic_value filename
```

Changing Permissions

- **chmod permission filename**
- **Example:**

Change the permissions of oldpasswd file to give owner read and write permissions and for group read ,write and execute and execute onlv for the others.

Changing Permissions

- **chmod permission filename**
- **Example:**

Change the permissions of oldpasswd file to give owner read and execute permissions and add read permission to group and remove execute permission for the others.

Changing Permissions

- chmod permission filename
- Permissions are specified in either
  Octal mode

- 4 read
- 2 write
- 1 execute

To change the file permissions for an existing file or directory.

```
chmod  octal_value  filename
```

Examples

- **ls -l file1**

```
-rw-r--r-- 1 user1 staff 1319 Mar 22 14:51 file1
```

- **chmod o-r file1**
- **ls -l file1**

```
-rw-r----- 1 user1 staff 1319 Mar 22 14:52 file1
```

- **chmod g-r file1**
- **ls -l file1**

```
-rw------- 1 user1 staff 1319 Mar 22 14:53 file1
```

Examples

- **chmod u+x,go+r file1**
- **ls -l file1**

```
-rwxr--r-- 1 user1 staff 1319 Mar 22 14:54 file1
```

- **chmod a=rw file1**
- **ls -l file1**

```
-rw-rw-rw- 1 user1 staff 1319 Mar 22 14:55 file1
```

- **chmod 555 file1**
- **ls -l file1**

```
-r-xr-xr-x 1 user1 staff 1319 Mar 22 14:56 file1
```

Examples

- **chmod 775 file1**
- **ls -l file1**

```
-rwxrwxr-x 1 user1 staff 1319 Mar 22 14:54 file1
```

- **chmod 755 file1**
- **ls -l file1**

```
-rwxr-xr-x 1 user1 staff 1319 Mar 22 14:55 file1
```

Default Permissions

- The umask command sets the default permissions
for files and
directories.

Example:

```
umask 002
umask
002
```

System Shutdown

- It only requires reboot or shutdown when you
need to
- Add or remove hardware
- Upgrade to a new version of Ubuntu
- Or upgrade your kernel
- shutdown –k now
    - # doesn't really shutdown only send the warning messages and disable logins.
- shutdown -h time # Halt after shutdown
- poweroff
- init 0

Day 3

Day 3 Contents

- Vi text editor.
- Initialization Files.
- Environment Variables.

The Vi Text Editor

- Vi editor (visual editor) is the default editor for Unix and Linux operating system.
- Vi is used to manage file content.
- Vi is an interactive editor that you can use to create and modify test files.
- Usually the only editor available in emergency mode.
- It is used when the desktop environment window system is not available.

The Vi Text Editor

- vi in Linux is usually vim (vi improved):
  - Syntax highlighting.
  - Arrow keys, Del, BS work in insert mode.
  - Mouse support.
- An advantages of this editor is that we can manipulate text without using a mouse. We can only need the keyboard.

Vi Operations

VI has three basic modes:

- Command mode:
  - – Default mode.
  - – Perform commands to delete, copy, ...
- Edit (insert) mode:
  - – Enter text into the file.
- Last line mode:
  - – Advanced editing commands.
  - – To access it, enter a colon (:) while in the command mode.

Vi Operations

- The syntax of vi command:

vi
vi filename
vi options filename

- To recover a file

vi -r filename

- Viewing files in Read-only mode:

view filename
– Perform the :q command exit.

Vi Operations

- **Inserting and appending text:**
  - **i** Inserts text before the cursor.
  - **o** Opens a new blank line below the cursor.
  - **a** Appends text after the cursor.
  - **A** append text at the end of the line.
  - **I** insert text at the beginning of the line.
  - **O** opens a new line above the cursor.
- After editing Press **esc** to enter command mode

Manipulating Files Within Vi

- **Inserting and appending text:**
  - **h**, **left arrow**, or **backspace**: left one character.
  - **j** or **down arrow**: down one line.
  - **k** or **up arrow**: up one line.
  - **l**, **right arrow** or **space**: right one character

Manipulating Files Within Vi

- **Moving the cursor within the vi (cont.):**
  - **w** forward one word.
  - **b** back one word.
  - **e** to the end of the current word.
  - **0** to the beginning of the line.
  - **Enter**: down to the beginning of the next line.
  - **G** Goes to the last line of the file.
  - **nG** Goes to Line n.
  - **:n** Goes to Line n.
  - **Control-F** Pages forward one screen.
  - **Control-B** Pages back one screen.
  - **Control-L** refresh the screen.

Manipulating Files Within Vi

- **Substitute and delete text:**
  - **s** Substitutes a string for a character at the cursor.
  - **x** Deletes a character at the cursor.
  - **dw** Deletes a word or part of the word to the right of the cursor.
  - **dd** Deletes the line containing the cursor.
  - **D** Deletes the line from the cursor to the right end of the line.
  - **n,nd** Deletes Lines n through n.

Manipulating Files Within Vi

- **Search and replace:**
  - **/string** Searches forward for the string.
  - **?string** Searches backward for the string.
  - **n** Searches for the next occurrence of the string.
  - **N** Searches for the previous occurrence of the string.
  - **%s/old/new/g** Searches for the old string and replaces it with the new string globally.

Manipulating Files Within Vi

- **Copy and paste:**
  - **yy** Yank a copy of a line.
  - **p** Put yanked text under the line containing the cursor.
  - **P** Put yanked text before the line containing the cursor.
  - **n,n co n** Copy Lines n though n and puts them after Line n.
  - **n,n m n** Move Lines n through n to Line n.

Manipulating Files Within Vi

- **Save and quit:**
  - **:w** save the file.
  - **:w** new_file save as new file.
  - **:wq, :x, ZZ** save and quit.
  - **:q!** quit without saving.

Manipulating Files Within Vi

- **Customizing vi session:**
  - **:set nu, :set nonu** show and hide line numbers.
  - **:set ic, :set noic** ignore or be case sensitive.
  - **:set showmode, :set noshowmode** display or turn off mode.

Editing Files With Gedit

- The gedit text editor is a graphical tool for editing text files.
- The gedit window is launched by selecting: Search menu→ gedit

Environment Variables

- **$HOME**
  - Complete path of the user home directory. Example:
    – mkdir $HOME/file1
- **$PWD**
  - The user current working directory.
- **$SHELL**
  - Path name of the login shell.
- **$USER**
  - Currently logged in user.
- **$HOSTNAME**
  - Name of the computer.
- **$PATH**
  - A colon-separated list of directories used by the shell to look for executable program names.

    Example:

    – echo $PATH

/home/fatma/.local/bin:/home/fatma/bin:/usr/local/bin: /usr/local/sbin:/usr/bin:/usr/sbin

Viewing Variable Contents

- The shell assumes whatever follows the dollar sign ($) in the command line is a variable and substitutes its value.
  - **echo $HOME**

    /home/user1
- To display the current Environment variables with values use the **env** or **printenv** command.

Creating a User Environment

- When a user logs in, an environment is created.
- The environment consists of some variables that determine how the user is working.
- One such variable is **$PATH**, which defines a list of directories that should be searched when a user types a command.

Creating a User Environment

- To construct the user environment, a few file play a role:
  - Global initialization file: **/etc/profile and /etc/bashrc**
  - Initialization file: **~/.profile**
  - Startup files: **~/.bashrc**
- When logging in, the files are read in this order, and variables and other settings that are defined in these files are applied.
- If a variable or setting occurs in more than one file, the last one wins.

Creating a User Environment

- **/etc/profile:** Used for default settings for all users when starting a login shell.
- **/etc/bashrc:** Used to define defaults for all users when starting a subshell.
- **~/.profile:** Specific setting for one user applied when starting a login shell.
- **~/.bashrc:**Specific setting for one user applied when starting a subshell.

Command Alias

- The purpose of the linux shell is to provide an environment in which commands can be executed.
- The shell takes care of interpreting the command that a user has entered correctly.
- To do this, the shell makes a distinction between three kinds of commands.

- Aliases.

- Internal commands.

- External commands.

Command Alias

● **Alias** is a command that a user can define as needed.
● **alias newcommand = ‘oldcommand’** alias ll=’ls -l’
● **Alias** are executed before anything else.
● An **internal command** is a command that is a part of the shell itself and, as such, doesn’t have to be loaded from disk separately.
● An **external command** is a command that exists as an executable file on the disk of the computer.

Command Alias

● To find out whether a command is a Bash internal or an executable file on disk, you can use the **type** command.
● To find out which exact command the shell will be using, you can use the **which** command.
● Type **alias** at the terminal to see all set aliases.
● To remove aliases, you can use **unalias** command.

Command History

● Bash stores a history of commands you have entered so that you can recall them later.
● The history is stored in the user's home directory and is called **.bash_history** by default.
● You can recall commands by pressing the up arrow key.
!!: Repeats the last command.
!string: Repeats the last command that started with string.
!n: Repeats a command by its number in history output.
!-n: Repeats a command entered n commands back.

Day 4

Day 4 Contents

- Processes, priorities and signals Concepts .
- Redirection.
- Pipe Line.
- Word Count.

Processes

- Every program you run creates a process.
  Example:
  - **Shell**
  - **Command**
  - **An application**

Processes

- For everything that happens on a Linux server, a process is started.
- System starts processes called daemons which are processes that
run in the background and provide services.
- Every processes has a PID.
- When a process creates another, the first is the **parent** of the new process. The new process is called the **child process**.

Processes

- **Types of process:**
  - **Shell jobs.**
  - **Daemons.**
  - **Kernel threads.**

Viewing Processes

- **Ps (process status) command**

  ps [options]

Output	**Options**
– **PID.**	– **-e: all system processes.**
– **TTY -> terminal identifier.**	– **-f: full information.**
– **Execution time.**	– **-u uid: display processes of that user.**
– **Command name.**	– **a: all processes attached to a terminal.**
	– **x:all other processes.**

Shell jobs

- Shell jobs are commands started from the command line. They associated with the shell that was current when the process was started.
- When a user types a command, a shell job is started.
- By default, any executed command is started as **foreground** job.
- If you know that a job will take a long time to complete, you can start it in with an & behind it.
- This immediately starts the job in the **background** to make room for other tasks to be started from the command line.

Shell jobs

- **A signal is a message sent to a process to perform a certain action.**
- **To send signals to processe or process group , you can use kill command, killall command or pkill command.**

• **Kill -[signal] PID**	**kill 12047**
• **Pkill -[signal] process_name**	**kill -9 mail**
• **Killall process_name**	**killall vim**

Shell jobs

- **Signals are identified by a signal number and a signal name, and has an associated action.**

  – **SIGTERM → 15** – **SIGKILL → 9**
- **If no signal is specified, the TERM signal is sent.**
- **For complete overview of all the available signals, you can use man 7 signal.**

Shell jobs

Examples

```
• sleep 3600&
  [1] 3302
• jobs
  [1]+  Running              sleep 3600 &
• fg 1
  sleep 3600
• ctrl+z  [1]+  Stopped             sleep 3600
• jobs
  [1]+  Stopped              sleep 3600
• bg %1
  [1]+ sleep 3600 &
```

10

Shell jobs

Examples

```
• jobs
  [1]+  Running              sleep 3600 &
• kill -SIGSTOP %1
  [1]+  Stopped              sleep 3600
• kill %1
  [1]+  Terminated           sleep 3600
• jobs
```

Processes

- Every process has a **parent process**, and as long as it lives, the parent process is responsible for the **child processes** it has created.
- **In older versions of Linux**, killing a parent process would kill all of
its child processes.
- **In RHEL 8**, if you kill a parent process, all of its child processes become children of the systemd process.

Processes Priority

- When Linux processes are started, they are started with a specific priority.
- By default, all regular proceeses are equal and are started with the same priority, which is the priority number20.
- Every process which is ready to run has a scheduling priority.
- The Linux process divides CPU time into time slices, in which each process will get a turn to run, higher priority processes first.
- User can affect the priority by setting the niceness value for a process.

Adjusting Priority

- Niceness values range from -20 to +19, which indicates how much of a bonus or penalty to assign to the priority of the process.
- To change the default priority that was assigned to the process when it was started.
- Use **nice** if you want to start a process with an adjusted priority.
- **nice** [-n adjustment] command

nice -n 5 dd if=/dev/zero of=/dev/null &

Adjusting Priority

- Use **renice** to change the priority for a currently active process, or you can use the **r** command from the **top utility** to change the priority of a currently running process.
- **renice** priority [[-p] pid ...] [[-g] group ...] [[-u] user ...]

ps | aux

renice -n 10 -p 1234

- The default niceness of a process is set to 0 (which results in the priority value of 20.

Adjusting Priority

- By applying a negative niceness, you increase the priority.
- Use a positive nicenessto decrease the priority.
- Do not set process priority to -20, it risks blocking other processes from getting served.
- The regular users can only decrease the priority of a running process.
- You must be root to give processes increased priority.

Viewing Processes

- Use **top** to display Linux processes.
- The top program provides a dynamic real-time view of a running system.
- It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel.

From slide 18 to 22 questions in pdf

Searching For A Process

- To search for a process, you can use pgrep command.
- pgrep option(s) pattern.
- Options
  - **-x:** exact match.
  - **-u uid:** processes for a specific user.
  - **-l:** display the name with pid.

Standard Input And Output

- **Standard input:**
  - Refers to the data source from which data is input to a command.
- Typically the keyboard.
  - Standard output:
  - Refer to data destination to which data from the command is written.
  - Typically the screen.
- Standard error:
  - Refer to the output destination for the errors and messages generated by the command.
  - Typically the screen also.

Standard Input And Output

- In I/O redirection, files can be used to replace the default **standard input**, **standard output** and **standard error**.
- You can also redirect to device files.
- If you want to discard a command's output, you can redirect to **/dev/null**.

Redirection

- **Standard input:**
  - command < fname
- **Standard output:**

Redirection

- **Examples:**

```
ls -R / > file.txt 2> /dev/null
ls –l /etc >> findresult
find /etc –name passwd > findresult
find / -name passwd 2> errs > results
mail < file2.txt
sort < file2.txt > sortedf1.txt
```

Pipe Line

● A pipe (|) is used to send the output of one command as the input to another.

● Command 1 | Command 2.

● Examples:

```
ls -lR / | more
ps -ef | more
history | more
```

The tee Command

● The tee command reads from the standard input and writes to the
standard output and a file.

● Examples:

```
ls -lR / | tee fname | more
```

String Processing

● Use the **wc** and the **diff** commands to gather word file statistics and compare two files.

● Search strings for patterns using the **grep** command.

● Move and delete data using **cut** and **paste** commands.

● Organize data using the **sort**, and **paste** command.

The wc command

● The wc command displays the number of characters, words, and
lines in a specified file.

● The syntax for the wc command is:

wc [option] [filename]

● The wc command is often used when differentiating between two
versions of a file.

The wc command

- **Word-count command options**

Option	Meanings
-c	Count the number of characters only.
-l	Count the number of lines only.
-w	Counts the number of words only

- **Example:**

```
wc story.txt
39 237 1901 story.txt
```

The diff command

● The diff command is also used to compare the contents of two files for differences. If you upgrade a utility and want to see how the new configuration files differ from the old, use the diff command.

● **diff** /etc/named.conf.rpm.new /etc/named.conf

will give the output as:

```
20c20
<
---- file "root.hints";
> file "named.ca"
```

The grep command

- **Displays the lines of its input that match a pattern given as an argument.**
- **The syntax for the grep command is:**

grep [options] regular-expression filename(s)

Option	Description
-i	Not case sensitive.
-v	Only shows lines that do not contain the regular expression.
-r	Searches files in the current directory and all subdirectories.

The tr command

● The tr command can be used to translate characters from standard
input and write to standard output.

● The syntax for the tr command is:

tr [option] string1 string2

● **Example**

● echo "Hello, world." | tr 'a-z' 'A-Z'

```
HELLO, WORLD
```

The cut command

● cut command cuts fields or columns of text from standard input or the named file
and displays the result to standard.

● The syntax for the cut command is:

cut option[s] [filename]

● **Options**

• **-f** specifies field or column.

• **-d** specifies field delimiter (default is TAB).

• **-c** specifies characters and cuts by characters.

● **Example**

```
cut -f3 -d: /etc/passwd
```

The sort command

● The sort command sorts text data after accepting it from either a file or the output of another command.

● The sorted text is sent to the standard output, with the original file remaining unchanged in the process.

● The syntax for the sort command is:

sort option[s] [filename]

● Example

```
sort –t : -k1 /etc/passwd
sort –t : –k3 /etc/passwd
sort –t : -n –k3 –o passwd_sorted /etc/passwd
```

Day 5

Day 5 Contents

- Inode table.
- Archiving.
- Compression.
- Yum.
- Search.

Listing Directory Contents

- **ls -l dir1**

```
-rwxr-xr-x 2 root root 20 512 May 21 16:06 file1
drwxrwxrwx 2 fatma fatma 20 512 May 21 16:06 dir2
```

User group others owner group

permission

Inode

- Linux stores administrative data about files in inodes.
- Linux see all files as numbers called “inodes”, or index nodes.
- Within each filesystem is an inode table, in which all of the used inodes are mapped to particular files.

Inode

- **The information stored in this table for each entry includes the following:**

1.The type of file.
2.The file's permissions.
3.The number of links.
4.The file owner's user ID.
5.The group owner's GID.
6.When the file was last changed.
7.When the file was last accessed.
8.Where the file is on the media.

Inode

- But It does not contain the file name or file content.
- Names are stored in the directory.
- Each file name knows which inode it has to address to access further file information.
- An inode does not know which name it has; It just know how many names are associated with the inode, These names are referred to as hard links.

Inode

- To view inode number of a file:

```
ls -i /etc/passwd
1971109 /etc/passwd
```

- To view inode number of a directory:

```
ls -id /etc
1966081 /etc
```

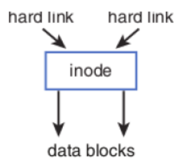
Hard Link

- When you create a file, you give it a name. Basically, this name is a hard link.
- On a Linux file system, multiple hard links can be created to a file. This can be useful, because it enables you to access the file from multiple different locations.
- If the first hard link that ever existed for a file is removed, that does not impact the other hard links that still exist.
- **Some restrictions apply to hard links, though:**
  - Hard links must exist all on the same device (partition, logical volume, etc).
  - You cannot create hard links to directories.
  - When the last name (hard link) to a file is removed, access to the file’s data is also removed.

Hard Link

- **To create hard link:**
  - **ln source-file targetfile or directory**
  - **ln /home/fatma/myfile hardlinkfile**
  - **ls -i /home/fatma/myfile hardlinkfile**

```
11272876 myfile  11272876 hardlinkfile
```



- **To be able to create hard links, you must be the owner of the item that you want to link to .**

File Manipulation

- The **cp** command:
- Allocates a new inode number for the copy, placing a new entry in
the inode table.
- Creates a directory entry, referencing the file name to the inode
number within that directory.

File Manipulation

- Example:
  - ls -i f1

    1196100 f1
  - cp f1 f2
  - ls -i f1 f2

    1196100 f1
    1196463 f2

File Manipulation

- The mv command:
- If the destination is on the same file system as the source:
- mv creates a new directory entry with the new file name.
- Example:
  - ls -i f1

    1196100 f1
  - mv f1 f2
  - ls -i f2

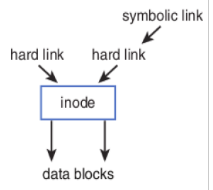
    1196100 f2

Symbolic Links

- New entry is made to the inode table for the link The content of this entry is the path to the original file.
- This allows you to use symbolic links across partition boundaries.
- The advantage of symbolic links is that they can link to files on other devices, as well as on directories.
- But when the original file is removed, the symbolic link becomes invalid and does not work any longer.

Symbolic Links

- **To create hard link:**
  - **ln -s source-file targetfile or directory**
  - **ln -s testfile softlinkfile**



  - **ls -li testfile softlinkfile**

```
1127996 -rw-rw-r-- 1 user user 12 Mar 12 03:50 testfile
1127999 lrwxrwxrwx 1 user user 8 Mar 12 09:50 softlinkfile-> testfile
```

Archiving

- To safeguard your files and directories, you can create a copy, or archive, of the files and directories on a removable medium, such as a cartridge tape. You can use the archived copies to retrieve lost deleted, or damaged files.

Archiving

- The Tape Archiver (tar) utility is used to archive files. It designed to stream files to a backup tape.
- To put files on the directory, you need at least read permissions to the file and execute permissions on the directory the file resides in.
- **To create an archive:**
  - tar -cvf archivename.tar file1 file2 file3

    c: create a new tar file.

    v: verbose mode.

    f: specify the archive file.

Managing Archives with tar

- To add a file to an existing archive or to update an archive:
  - tar -cvf /root/homes.tar /home
  - tar -rvf /root/homes.tar /etc/hosts

    r: Appends files to an archive.
  - tar -uvf /root/homes.tar /home

    u: updates an archive, only newer files will be written to the archive.

Managing Archives with tar

- To see the contents of the tar archive:
  - tar -tvf /root/homes.tar

    t: List table of content.

- To extract the contents of an archive:
  - tar -xvf /root/homes.tar

    x: Extracts files from the tar command.
  - tar -xvf /root/homes.tar -C /tmp

    C: To specify the target directory you want to extract the file to.

Compression

- Many files contain a lot of redundancy. Compression programs allow you to make files take less disk space by taking out that redundancy.
- If there is no redundancy, you won't gain much by using compression.

Compression

- After creating the archive, it had to be compressed with a separate compression utility, such as gzip or bzip2.
- you can include the **-z**(gzip) or **-j**(bzip2) **option** while creating the archive with tar.This will immediately compress the archive while it is created.
  - gzip homes.tar
  - bzip2 homes.tar
- To decompress:
  - gunzip homes.tar
  - bunzip2 homes.tar

Managing Software

- The default utility used to manage software packages on RHEL is yum (**Yellowdog Updater**, **Modified**).
- Yum is designed to work with **repositories** which are online depots of available softwarepackages.
- In RHEL 8, Yum has been replaced with the dnf utility. Because software in RHEL is based on Fedora software.
- It was expected that, the yum command would be replaced with the dnf command. But Red Hat decided that, with RHEL 8, a new version of yum has been introduced, which is based on the dnf command.
- You'll notice that in many cases, when requesting information about yum, you're redirected to dnf resources.
- So in fact you are using dnf, but RedHat has decide to rename it to yum.

Yum

- **Basic command**
- yum search somefile (look for the package)
- yum list somefile (get installed and available versions)
- yum list installed (same as rpm -qa)
- yum list available (what's available in repository)
- yum grouplist "some search string" (look for like packages to search string)
- yum install somefile (install the package and any dependencies)
- yum localinstall /path/to/somefile (yum install off local media)

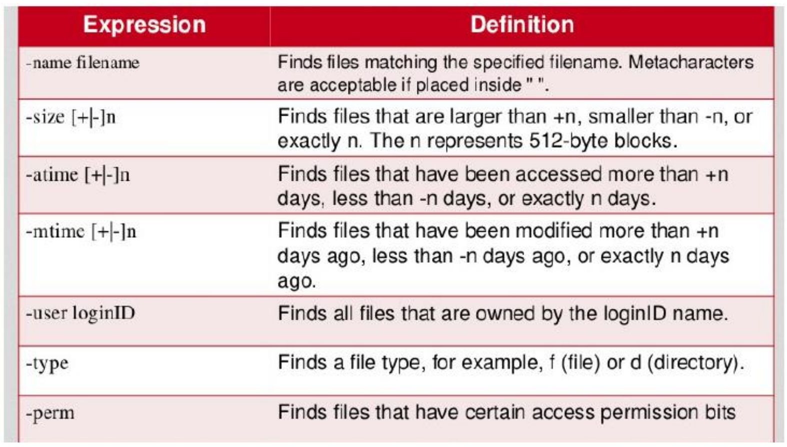
Yum

- **Basic command**
- yum remove somefile (uninstall the package)
- yum upgrade somefile (upgrade the package removing prior versions)
- yum update somefile (update the package keeping prior version)
- yum provides somefile (what packages are associated with a file)
- yum repolist all (list defined repositories)
- yum clean all (clean yum download directories)

Search

- The find command searches the live filesystem.
- You are limited by your own permissions.

Find



Expression	Definition
-name filename	Finds files matching the specified filename. Metacharacters are acceptable if placed inside " ".
-size [+|-]n	Finds files that are larger than +n, smaller than -n, or exactly n. The n represents 512-byte blocks.
-atime [+|-]n	Finds files that have been accessed more than +n days, less than -n days, or exactly n days.
-mtime [+|-]n	Finds files that have been modified more than +n days ago, less than -n days ago, or exactly n days ago.
-user loginID	Finds all files that are owned by the loginID name.
-type	Finds a file type, for example, f (file) or d (directory).
-perm	Finds files that have certain access permission bits