OPERATING SYSTEMS
CCGC-5000

2.2

Agenda

Authentic information is available from the given resources in course outline and URL's mentioned from this slides, and this presentation is only supportive document to be read with the given resources and corrected accordingly if required..

- UNIX/Linux CLI
- Utilities - System Information
- UNIX/Linux text editors
- Grouping commands, pipes
- Regex, grep
- Printing, CUPS
- Shell scripting, variables, echo

Must read

- Chapters 1,2,3,7,22 of RHEL8, 2nd Edition book
- RedHat documentation
  https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/

UNIX/Linux CLI

- If you need to list the contents in a directory, we use **ls command**

```
[user1@hostname ~]$ ls
biotech     Downloads     Pictures      tech        technician   Videos
Desktop     Music         polytechnic   technic     technology
Documents   nontechnical  Public        technical   Templates
```

- To list the contents with more information we use **option -l** , we use **ls –l**
- In case you need to filter above output, we can use **argument** like file name or with wild cards **ls –l tech*** long lists file whose name begins with tech from directory where you run the command.
- Likewise **ls -l *tech** long lists files whose name ends with tech and **ls -l ?*tech*?** long lists files where tech is in the name of the file but not at start or/and at the end
- Command **ls -l *tech*** lists all files that contain the string tech in its name anywhere
- For wildcard search **?** can be used to search a single character, ?? two character search and so on. Some examples below

```
[user1@hostname ~]$ ls -l file*
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file1
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file2
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file22
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file2a
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file2lab
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file3
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file33
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file3clab1
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file4
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file4444
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file5555
```

```
[user1@hostname ~]$ ls -l file?
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file1
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file2
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file3
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:36 file4
```

```
[user1@hostname ~]$ ls -l file?lab
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file2lab
[user1@hostname ~]$ ls -l file??lab
ls: cannot access 'file??lab': No such file or directory
[user1@hostname ~]$ ls -l file??lab?
-rw-rw-r--. 1 user1 user1 0 Jan 19 00:38 file3clab1
```

```
[user1@hostname ~]$ ls -l
total 32
-rw-rw-r--. 1 user1 user1  8 Jan 19 00:26 biotech
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Desktop
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Documents
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Downloads
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Music
-rw-rw-r--. 1 user1 user1 13 Jan 19 00:27 nontechnical
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Pictures
-rw-rw-r--. 1 user1 user1 12 Jan 19 00:26 polytechnic
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Public
-rw-rw-r--. 1 user1 user1  5 Jan 19 00:25 tech
-rw-rw-r--. 1 user1 user1  8 Jan 19 00:25 technic
-rw-rw-r--. 1 user1 user1 10 Jan 19 00:26 technical
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:25 technician
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:26 technology
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Templates
drwxr-xr-x. 2 user1 user1  6 Jan 19 00:20 Videos
```

```
[user1@hostname ~]$ ls -l tech*
-rw-rw-r--. 1 user1 user1  5 Jan 19 00:25 tech
-rw-rw-r--. 1 user1 user1  8 Jan 19 00:25 technic
-rw-rw-r--. 1 user1 user1 10 Jan 19 00:26 technical
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:25 technician
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:26 technology
```

```
[user1@hostname ~]$ ls -l *tech
-rw-rw-r--. 1 user1 user1 8 Jan 19 00:26 biotech
-rw-rw-r--. 1 user1 user1 5 Jan 19 00:25 tech
```

```
[user1@hostname ~]$ ls -l ?*tech*?
-rw-rw-r--. 1 user1 user1 13 Jan 19 00:27 nontechnical
-rw-rw-r--. 1 user1 user1 12 Jan 19 00:26 polytechnic
```

```
[user1@hostname ~]$ ls -l *tech*
-rw-rw-r--. 1 user1 user1  8 Jan 19 00:26 biotech
-rw-rw-r--. 1 user1 user1 13 Jan 19 00:27 nontechnical
-rw-rw-r--. 1 user1 user1 12 Jan 19 00:26 polytechnic
-rw-rw-r--. 1 user1 user1  5 Jan 19 00:25 tech
-rw-rw-r--. 1 user1 user1  8 Jan 19 00:25 technic
-rw-rw-r--. 1 user1 user1 10 Jan 19 00:26 technical
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:25 technician
-rw-rw-r--. 1 user1 user1 11 Jan 19 00:26 technology
```

Use **ls --help** to know **ls** options

Utilities – System Information

HUMBER
Faculty of Applied Sciences & Technology

tty- prints the file name of the terminal connected to standard output

```
[user1@hostname ~]$ tty
/dev/pts/0
[user1@hostname ~]$
```

```
[user1@hostname ~]$ tty
/dev/pts/1
[user1@hostname ~]$
```

```
[user1@hostname ~]$ tty
/dev/pts/2
[user1@hostname ~]$
```

```
[user1@hostname ~]$ tty
/dev/pts/3
[user1@hostname ~]$
```

whoami - Print the user name associated with the current effective user ID.

```
[user1@hostname ~]$ whoami
user1
```

who - show who is logged on

Try **who -b**

(find purpose of option b)

```
[user1@hostname ~]$ who
user1    tty2         2021-01-19 00:20 (tty2)
user1    tty3         2021-01-20 22:29
user1    tty5         2021-01-20 22:29
user1    tty6         2021-01-20 22:29
user1    tty4         2021-01-20 22:30
```

last - shows last logged in users

```
[user1@hostname ~]$ last
user1    tty4                          Wed Jan 20 22:30   still logged in
user1    tty6                          Wed Jan 20 22:29   still logged in
user1    tty5                          Wed Jan 20 22:29   still logged in
user1    tty3                          Wed Jan 20 22:29   still logged in
user1    tty2         tty2             Tue Jan 19 00:20    gone - no logout
```

Try last command as below and learn its purpose

last reboot,

last tty0, last 0, last tty1, last 1

likewise try for other tty's and note tty# is same as tty #

w - show who is logged on and what they are doing

```
[user1@hostname ~]$ w
 22:32:22 up  4:44,  5 users,  load average: 0.05, 0.06, 0.02
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
user1    tty2     tty2             Tue00   46:19m  3:26   0.43s /usr/libexec/tr
user1    tty3     -                22:29    2:56   0.01s  0.01s -bash
user1    tty5     -                22:29    2:50   0.01s  0.01s -bash
user1    tty6     -                22:29    2:46   0.01s  0.01s -bash
user1    tty4     -                22:30    2:14   0.02s  0.02s -bash
```

lastb - shows listing of last logged in users

Listing System Information

- **lsblk** command lists the block devices *(and loop devices)*
- **lspci** command display information about the PCI buses in the system and the devices connected to it.
- **lscpu** command display information about the CPU architecture
- **lsusb** command display information about the USB buses in the system and the devices connected to it.
- **lsmod** command displays the status of modules in the Linux kernel.
- **lshw** command to extract detailed information on the hardware configuration of the machine
  - Recommended to run as super user
  - Individual class of hardware can be listed using **-class** option: **sudo lshw -class** *class_name*
  - To find the list of class you use lshw -short or lshw -businfo

Unix/Linux Creating Files

- Using **cat** command, regular file can be created and displayed
- To create a file using cat, we can use **cat >** *filename* and to save the file, **CTL+D**
- **>** is redirection operator.

- If cat command is used with single redirection operator with an existing file, it will be overwritten
- Double redirection operator **>>** should be used to **append** data to an existing file with cat command, we can use **cat >>** *filename*, and **CTL+D** to save

- Just to **display** the contents of the file: **cat** *filename*

```
[user1@hostname ~]$ cat > linuxcert
linux+
rhcsa
[user1@hostname ~]$ cat linuxcert
linux+
rhcsa
[user1@hostname ~]$ cat >> linuxcert
rhce
[user1@hostname ~]$ cat linuxcert
linux+
rhcsa
rhce
[user1@hostname ~]$ cat > linuxcert
Oracle Linux Certification
[user1@hostname ~]$ cat linuxcert
Oracle Linux Certification
```

- Also, command **touch** which update the access and modification times of each FILE to the current time it can also be used to create an empty regular file with command syntax **touch** *filename*
- Multiple files can be created using **touch**. For example to create file1, file2 and file3, we can use **touch** *file1 file2 file3*. Likewise many files can be created.

Unix/Linux - Creating Files

- Commands cat and touch may not be used to edit files, the text editors are helpful in creating or editing the files
- There are various text editors in linux, vi, vim, nano, pico, etc.,
- **vi** is a powerful text editor used in unix/linux
- **vim** is vi improved version of text editor
- **nano** is a simple text editor
- **pico** is similar to nano
- These editors helps to create news files and modify existing files.
- There is also GUI text editor **gedit** available in ubuntu which is quite similar to notepad in windows.

Unix/Linux vi/vim editor

- To create or edit a file **vi** *filename* or **vim** *filename* can be used.
- **vi** is installed by default, and for **vim** it has to be installed.
- **vi** and **vim** editor operates in two modes and **ESC** key toggles between the two modes
  - Command mode
    - When **vi**/**vim** is invoked, it is in command mode
    - Use **ESC** key to enter command mode
    - When in command mode, enter the command required in saving or editing the file.
    - Refer man vi or man vim for all the options available.
    - Command to enter text mode : i, I, a, A, o, O, R and c
  - Text mode
    - In this mode is were the typing of text is done
- Using vim/vi the text will displayed in color depending on the codes and quotes.

vi/vim commands

- Commands to enter text mode

i	Insert before cursor
I	Insert at the beginning of current line
a	Append text after cursor position
A	Append at the end of the current line
o	Open a new line before the cursor position
O	Open a new line after the cursor position

- Text deletion commands

x	Deletes character under cursor
dw	Deletes from cursor to beginning of next word
dd	Deletes line containing cursor

Usual steps in creating and saving file using vi or vim

1. **vi *filename*** or **vim *filename***
2. Type **i**
3. Type the text in the file
4. To save you can press **ESC** and type **:wq!** or press **ESC** and type **:x**
5. If need to save in between press **ESC** then type **:w!** and then press **ESC** to type text again and continue

vi/vim commands

Text alternation commands	
r	Replace character under cursor with next char typed
cw	change word (beginning at cursor) to new text
J	join next line down to line with cursor
u	undo last command

Text moving commands	
yy	yank a copy of a line, place it in a buffer
p	put after the cursor the last item yanked or deleted
P	put before the cursor the last item yanked or deleted

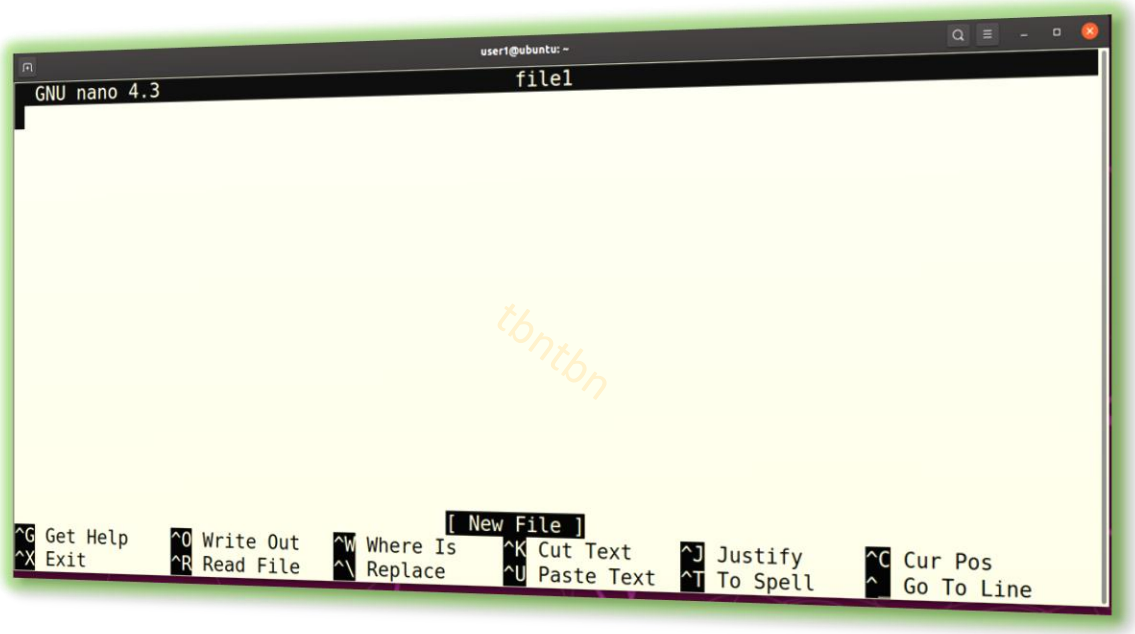
Saving text and quitting	
:w	write the current text into the permanent file
:q	quit, if no changes since last write
:q!	quit, discarding all changes since last write
:wq	save and quit
:x	save and quit

- To search a word in vi or vim,
- Type / and then the word which need to be searched in the file and press enter
- It will highlight the first available word and then if need to search the next word, press **n**
- Likewise until the last word available in the file, n can be used, when the searching word not available it will display as

  Pattern not found (press RETURN)

nano

- nano one of the text editor used to edit or create a file with **nano** *filename*
- Now text can by entered
- When nano editor is opened, at the bottom various options that can be used are displayed
- To save the file we use Ctl+x, then confirm to save by y and press enter to confirm the file name location.
- To search a word CTL+w can be used.
- Refer to manual pages and the help options available



Creating a file in Linux - basics

- Most common text editor in CLI are **vi**, **vim** *(vim is vi improved)* and **nano**

- Steps to create a file using **vim**
  1. Type cd, *(to be in your home directory)*
  2. To create file: **vim** *filename*
  3. Press **i** key or **Esc + i**
  4. Now type as typing in notepad
  5. To save use keys **Esc + :x**
     *(to exit **without saving** use keys **Esc + :q!**)*
  6. To list the file: **ls -l** *filename*
  7. To view file content: **cat** *filename*
- To edit the file start from step 2.
- **vi** can also follow the same steps for **vim**

- Steps to create a file using **nano**
  1. Type cd, *(to be in your home directory)*
  2. To create file - **nano** *filename*
  3. Now type as typing in notepad
  4. To save **CTL + x**
  5. Enter **Y** *to accept Save modified to buffer*
     *(to exit **without saving** Enter **N**)*
  6. Press **Enter** key *to confirm the file name to write*
  7. To list the file: **ls -l** *filename*
  8. To view file content: **cat** *filename*
- To edit the file start from step 2.

- The command **cat** which prints the file to screen, can be used to create file as below, but once the line is completed, you cannot edit the completed lines above
  1. To create file **cat >>** ***filename***
  2. Type the text
  3. To save **CTL + D**

Command **cat >** *filename* also can be used to create file and save using **CTL + D**., but by default for an existing file it will overwrite the existing file content with new text.

gedit

- gedit is text editor in GNOME desktop environment
- To use gedit, type **gedit** *filename*
  - start typing at the cursor
  - the file can be saved by clicking on Save
  - then to close click on ×

- gedit when run in the terminal, no further commands in the terminal can be run.
- Alternatively edit can be run in the background with the command **gedit** *filename* **&**
- It is important that you should have write permission to the directory where the file is created

Commands - grouping, pipe redirection

- Executing multiple UNIX/Linux commands in single line is termed as grouping commands and the commands are separated by ; (semicolon)
- Example: **pwd;who;whoami;ls -l file***

```
[user1@hostname ~]$ pwd;whoami;w;ls -l linuxcert
/home/user1
user1
 23:12:49 up  5:24,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
user1    tty2     tty2             Tue00   46:59m  3:40   0.50s /usr/libexec/tr
-rw-rw-r--. 1 user1 user1 27 Jan 20 23:07 linuxcert
```

- Output of a command can be redirected as input to another command using **|** *(called as pipe)*
- Example: **man mkdir |less**
- *In the above command* **man mkdir | less**,
  - *the output of manual pages of mkdir command, due to* **|** *(pipe) it is redirected as input to* **less** *command.*
  - *less command is a filter for paging through text one screenful at a time*
  - *Press spacebar to move pagewise and up and down arrow should help to move the text up and down accordingly.*
  - *To quit less, type* ***q***

Regular expression search - grep

- **g**lobally search a **r**egular **e**xpression and **p**rint
- **grep** was created by Ken Thompson as a standalone application adapted from the regular expression parser he had written for **ed** (which he also created, ed is line oriented text editor)
- **grep** is a command-line utility for searching plain-text data sets for lines matching a regular expression.
- In addition, the variant programs **egrep**, **fgrep** and **rgrep** are the same as grep -E, grep -F, and grep -r, respectively. These variants are deprecated, but are provided for backward compatibility. The shell recognizes a limited form of regular expressions when you use filename substitution.
- There are three different forms
  - **Anchors** : search based on location of the pattern,
    - using ^, $ in pattern search
  - **Character sets** : pattern search based on a single character
    - using [ ] in pattern search
  - **Modifiers** : specifies the number of times the previous character set is repeated

regular expressions

*	match zero or more instances of the single character immediately preceding it
.	match any single character except <newline>
+	matches the preceding element one or more times
[abc]	match any of the characters enclosed
[a-d]	match any character in the enclosed range
[^expr]	match any character not in the following expression
abc$	the regular expression must end at the end of the line
^abc	the regular expression must start at the beginning of the line
\	treat the next character literally. This is usually used to escape the meaning of special characters such as "*"
\<abc\>	will match the enclosed regular expression as long as it is a separate word.
{x,y}	Matches the preceding element at least *m* and not more than *n* times, *for example a{2,4}, searches for aa, aaa, aaaa in the file or text*

A period in a regular expression matches any single character, no matter what it is. So the regular expression **r.** specifies a pattern that matches an r followed by any single character.
The regular expression **.x.** matches an x that is surrounded by any two characters, not necessarily the same.

grep

- The general format of this command is

  grep *pattern filename* or

  cat *filename* | grep *pattern*

  <u>Examples</u>

  To find a pattern "Ken" in a file named intro, we can use **grep 'Ken' intro**

  or **cat intro | grep 'Ken'**

- Every line of each file that contains pattern is displayed at the terminal.
- If more than one file is specified to grep, each line is also immediately preceded by the name of the file, thus enabling you to identify the particular file that the pattern was found in.
- single quotes are required for the pattern when using metacharacters.

grep command

HUMBER
Faculty of Applied Sciences & Technology

- grep when used with options

EXAMPLE using mycourse.txt file with content as below

```
I am studying in Humber's north campus
Humber College, North Campus is in Toronto
Humber offers various programs from each school
My school at Humber is FAST
FAST refers to Faculty of Applied Sciences and Technology
HUMBER
FAST offers my program at Humber
My program is at Humber's North campus.
Other Humber campuses are Lakeshore and Carrier Drive
Humber
HUMBER.
humber
```

-n displays line number of the pattern found

```
[user1@hostname ~]$ grep -n 'Humber' mycourse.txt
1:I am studying in Humber's north campus
2:Humber College, North Campus is in Toronto
3:Humber offers various programs from each school
4:My school at Humber is FAST
7:FAST offers my program at Humber
8:My program is at Humber's North campus.
9:Other Humber campuses are Lakeshore and Carrier Drive
10:Humber
```

-v it displays the search pattern other than the regular expression

```
[user1@hostname ~]$ grep -nv 'Humber' mycourse.txt
5:FAST refers to Faculty of Applied Sciences and Technology
6:HUMBER
10:HUMBER.
11:humber
```

```
[user1@hostname ~]$ grep -nv 'Campus' mycourse.txt
1:I am studying in Humber's north campus
3:Humber offers various programs from each school
4:My school at Humber is FAST
5:FAST refers to Faculty of Applied Sciences and Technology
6:HUMBER
7:FAST offers my program at Humber
8:My program is at Humber's North campus.
9:Other Humber campuses are Lakeshore and Carrier Drive
10:Humber
11:HUMBER.
12:humber
```

*(ref **grep --help** for more options)*

grep & Regex - regular expressions

^*pattern* - the regular expression must start at the beginning of the line

```
[user1@hostname ~]$ grep -n '^Humber' mycourse.txt
2:Humber College, North Campus is in Toronto
3:Humber offers various programs from each school
10:Humber
```

*pattern***$** - the regular expression must end at the end of the line

```
[user1@hostname ~]$ grep -n 'Humber$' mycourse.txt
7:FAST offers my program at Humber
10:Humber
```

^*pattern***$** - the regular expression is only the word in between ^ and $

```
[user1@hostname ~]$ grep -n '^Humber$' mycourse.txt
10:Humber
```

grep & Regex - regular expressions

HUMBER
Faculty of Applied Sciences & Technology

* - match zero or more instances of the single character immediately preceding it

```
[user1@hostname ~]$ grep -n 'l*' mycourse.txt
1:I am studying in Humber's north campus
2:Humber College, North Campus is in Toronto
3:Humber offers various programs from each school
4:My school at Humber is FAST
5:FAST refers to Faculty of Applied Sciences and Technology
6:HUMBER
7:FAST offers my program at Humber
8:My program is at Humber's North campus.
9:Other Humber campuses are Lakeshore and Carrier Drive
10:Humber
11:HUMBER.
12:humber
```

[kxn] - match any of the characters k, x, n enclosed within square brackets

```
[user1@hostname ~]$ grep -n '[kxn]' mycourse.txt
1:I am studying in Humber's north campus
2:Humber College, North Campus is in Toronto
5:FAST refers to Faculty of Applied Sciences and Technology
9:Other Humber campuses are Lakeshore and Carrier Drive
```

For learning purpose, grep with search pattern and regular expression can be used with the dictionary in linux at **/usr/share/dict/words**

grep command has also variants **egrep**, **fgrep**, **rgrep** are the same as **grep -E**, **grep -F**, **grep -R** respectively, which has been deprecated, but could still be used.
Refer man grep for more options and information on grep command.

Pattern search

- In a pattern, \< represents the beginning of a word, and \>represents the end of a word

```
[user1@hostname ~]$ grep -n '\<camp' mycourse.txt
1:I am studying in Humber's north campus
8:My program is at Humber's North campus.
9:Other Humber campuses are Lakeshore and Carrier Drive
```

```
[user1@hostname ~]$ grep -n 'shore\>' mycourse.txt
9:Other Humber campuses are Lakeshore and Carrier Drive
```

- Note that without quotes grep does not give any result

```
[user1@hostname ~]$ grep -n \<camp mycourse.txt
[user1@hostname ~]$ grep -n shore\> mycourse.txt
[user1@hostname ~]$
```

- To match words <u>exactly</u> with 4 characters in /usr/share/dict/words

  grep -E -w '^[A-Za-z0-9]{4,4}' /usr/share/dict/words

- To find word length is <u>at least</u> 4

  grep -E '^[A-Za-z0-9]{4,}' /usr/share/dict/words

- To find word length is <u>maximum</u> 4

  grep -E -w '^[A-Za-z0-9]{,4}' /usr/share/dict/words

- *find the purpose of using -w option*

Linux Printing & CUPS

- Unix printing commands

  lpadmin - configure cups (Common Unix Printing System) printer and classes

  lpstat - print cups status information

  lpstat –p –d, displays the printers (-p option) and default printer (-d option)

- To print a file,

  lp *filename* or **lpr** *filename*

  lp –d *printer filename* or **lpr –P** *printer filename*

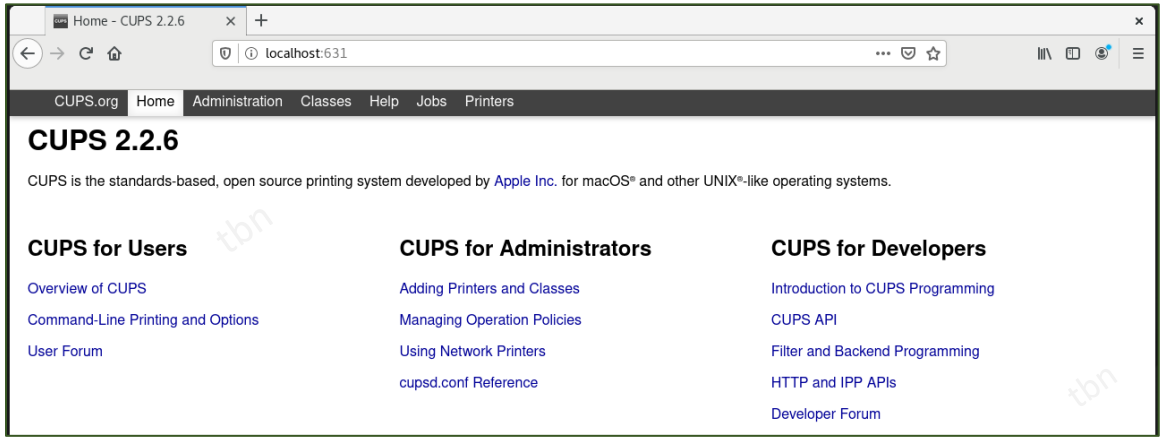
- To show printer queue status

  lpq

- To set default printer

  lpoptions –d *printer*

- Common Unix Printing System - CUPS can be accessed using web browser at URL http://localhost:631
- CUPS provides GUI interface to manage and administer Linux printing



http://www.cups.org/documentation.php/options.html

Shell scripting

- Shell scripts are created using text editor and saved with or without extension **sh**
- Using a file editor, you can add commands and then execute it based on the shell you are executing.
- Start each line with command
- To comment a line, # is used at the start of the line.
- The script is called as bash script when it is being run in bash shell.
- The first line normally called **shebang line** is mandatory only if the operating system executes the script in the same way binary executables.
- This has no effect on normal scripts
- Majority of shell scripts use a *shebang line* (#!) at the beginning to control the type of shell used to run the script that tells the Linux kernel that specific shell is to be used to interpret the contents of the file

- Script with file name script1a, using **echo** command,

```
[user1@hostname ~]$ cat script1
#!/bin/bash
echo "Sample bash script"
echo ==============================
echo "Hello Welcome to the program"
echo "Started learning Linux"
echo ==============================
echo "End of script"
```

- Following is execution of the script using command **bash** script1a, refer the output line by line in comparison with the bash script above.

```
[user1@hostname ~]$ bash script1
Sample bash script
==============================
Hello Welcome to the program
Started learning Linux
==============================
End of script
```

Variables

- Like virtually all programming languages, the shell allows you to store values into variables.
- A shell variable begins with an alphabetic or underscore (_) character and is followed by zero or more alphanumeric or underscore characters.
- To store a value inside a shell variable, you write the name of the variable, followed immediately by the equals sign =, followed immediately by the value you want to store in the variable:

  variable=value Examples: num1=5, str1="hello"
- echo command followed by $variablename can be used to display the variable
- Example for num1=5, to display the value in the variable num1 the command is **echo $num1**

```
[user1@hostname ~]$ str1="Hello"
[user1@hostname ~]$ echo $str1
Hello
```

```
[user1@hostname ~]$ num1=5
[user1@hostname ~]$ echo $num1
5
```

- These variables are local to environment they have been assigned unless they are exported.
- Until these variables are exported it can be termed as local variables.
- In UNIX/Linux variables can be of three types
  - **Environmental variables**
  - **Built-in or shell variables**
  - **user variables**

```
[user1@hostname ~]$ echo $x

[user1@hostname ~]$ x=57
[user1@hostname ~]$ echo $x
57
[user1@hostname ~]$ cat vartest1
#!/bin/bash
echo $x
[user1@hostname ~]$ bash vartest1

[user1@hostname ~]$ export x
[user1@hostname ~]$ bash vartest1
57
[user1@hostname ~]$ echo $x
57
```

Variables

- Symbolic names that represent values stored in memory
- Three types of variables are used in Linux while writing scripts
- **Environment Variables** :
  - A number of in-memory variables are assigned and loaded by default when you log in
  - Part of system environment, you can use them in your shell program, can define new and modify some of them within a shell program.
  - Commands **env** or **printenv** displays the environmental variables.

    Ex: $PATH, $USER, $LOGNAME, $HOME, $LOGNAME
- **Shell Variables**
  - also called built-in Variables, it starts with $,
  - examples are $$, $0,$1, $2, $#, $*, $?

    echo "My script's PID is $$" - *this displays the process ID of current process*
- **User Variables** :
  - Variables defined within a shell script, may be also be called as local variables

    string1=apples

    num1=345

Output of **env** or **printenv** command
(not complete output)

```
DISPLAY=:0
HOSTNAME=hostname.local.net
COLORTERM=truecolor
USERNAME=user1
XDG_VTNR=2
SSH_AUTH_SOCK=/run/user/1001/keyring/ssh
XDG_SESSION_ID=4
USER=user1
DESKTOP_SESSION=gnome
WAYLAND_DISPLAY=wayland-0
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/c9aa2197_513b_4240_920e_033d8c9
51864
PWD=/home/user1
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
HOME=/home/user1
XDG_SESSION_TYPE=wayland
XDG_DATA_DIRS=/home/user1/.local/share/flatpak/exports/share/:/var/lib/flatpak/e
xports/share/:/usr/local/share/:/usr/share/
XDG_SESSION_DESKTOP=gnome
GJS_DEBUG_OUTPUT=stderr
MAIL=/var/spool/mail/user1
VTE_VERSION=5204
TERM=xterm-256color
SHELL=/bin/bash
QT_IM_MODULE=ibus
XMODIFIERS=@im=ibus
XDG_CURRENT_DESKTOP=GNOME
GNOME_TERMINAL_SERVICE=:1.128
XDG_SEAT=seat0
SHLVL=2
GDMSESSION=gnome
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=user1
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1001/bus
XDG_RUNTIME_DIR=/run/user/1001
```

Shell Variable	Usage
$#	Number of arguments (positional parameters)
$?	Exit value of the last command executed
$$	Process number of the current process
$n	Argument on the command line, where n is from 1 through 9,
$0	The name of the current shell or program (the first token)
$*	All arguments on the command line ("$1 $2 … $9")

echo

- Command **echo** displays the string to standard output, by default the terminal
- Refer **echo --help** for options available
- When using quotes (single, double, no-quote) and $
- To display the full string when no quotes or double quote used, **escape character** - **** *(backslash)* need to be used.
- To display output of command using echo
  - echo $(**command**)
  - echo `**command**`

```
[user1@hostname ~]$ echo My laptop costs $2,354
My laptop costs ,354
```

```
[user1@hostname ~]$ echo "My laptop costs $2,354"
My laptop costs ,354
```

```
[user1@hostname ~]$ echo 'My laptop costs $2,354'
My laptop costs $2,354
```

```
[user1@hostname ~]$ echo My laptop costs \$2,354
My laptop costs $2,354
[user1@hostname ~]$ echo "My laptop costs \$2,354"
My laptop costs $2,354
```

```
[user1@hostname ~]$ echo $(whoami)
user1
```

```
[user1@hostname ~]$ echo `whoami`
user1
```

backquote - ` *(it is the key in US Keyboard where ~ (tilde) is or on the left to key 1)*