010123131

Software Development Practice

Handout #7

<rawat.s@eng.kmutnb.ac.th>

Last Update: 2024-07-22

Basic Linux Commands

Linux Shells

Shell Scripting

Bash Shell Programming

Terminal, Shell, Shell Script

- **Terminal**
  - A **terminal** is a peripheral (**hardware**) that interfaces with a human, it is composed of I/O such as a screen and a keyboard.
  - A **terminal** is a window (**software**) that holds a **shell** (or a **command line interpreter or CLI**).
- **Shell**
  - A **shell**, also known as **terminal**, **console**, **command prompt** and many others, is a computer program intended to interpret commands.
  - The main purpose of a **shell** is to allow the user to interact with the OS.
  - A **Linux terminal** is a **text-based interface** used to control a Linux computer.

Terminal, Shell, Shell Script

- A **shell script** is a text file that contains a sequence of commands for interacting with an OS such Unix and Linux.
- A **shell script** is a computer program designed to be run by a **shell** or a **command-line interpreter**.
- A **shell script** is a text-based file containing one or more commands that the user would type on the command line for specific tasks.

Shell Scripting

- **Shell programming**, usually referred to as **shell scripting**, allows for task automation for ease of use, reliability, and reproducibility.
- **Shell scripting** provides an easy way to carry out tedious commands, large or complicated sequence of commands, and routine tasks.
  - perform daily tasks efficiently and schedule them for automatic execution.
  - set certain scripts to execute on startup such as showing a particular message on launching a new session or setting certain environment variables.

Unix Shells

- Every modern OS has one or more **shells** as part of the system.
- In Unix, there are **different shells** such as the **Bourne shell** (sh), the **C shell** (csh), the **Korn shell** (ksh) and the **Bash shell**.
  - The **Bourne Shell** (sh) was originally developed by Stephen Bourne while working at Bell Labs.
  - The **Bourne Again Shell** (bash) was written as a **free and open source** replacement for the Bourne Shell.
  - **Bash** is succeeded by **Bourne shell** (sh) and has been adopted as **the default shell on most Linux distributions as well as macOS**.

Bash Shell

- In a **Linux Bash shell**, the first character is often a **dollar sign** (`$`) to indicate the **shell prompt** waiting for a command from the user.
  - If the user is **root**, the dollar sign will be replaced by the pound key (`#`).
- Check which shell is used by executing the following command:

```
$ echo $SHELL
```

- If a user has logged-in in a terminal, the **Bash script file** `~/.profile` in the user's home directory is executed automatically by a **Bash shell**.
- The **tilde symbol** (`~`) represents and expands to **the home directory of the current user**, which is the same as **the environment variable** `$HOME`.

The ~/.bashrc File

- The `~/.bashrc` file is a Bash script that is loaded whenever a user opens a new terminal session.
- **Environment variables** in this file are executed whenever a new session is started.
- You can add one or mode Bash commands:

  Example: `PATH="$HOME/.local/bin/:$PATH"`

- You can reload the .bashrc file with the following command

  ```
  $ source ~/.bashrc
  ```

- Examples of system-wide files (for all users):
  - `/etc/profile`
  - `/etc/bash.bashrc`
  - `/etc/environment`

Shell Prompt

- Bash keeps a list of directories in which it should look for commands in an environment variable called `PATH`.
- The default shell prompt, it is composed by

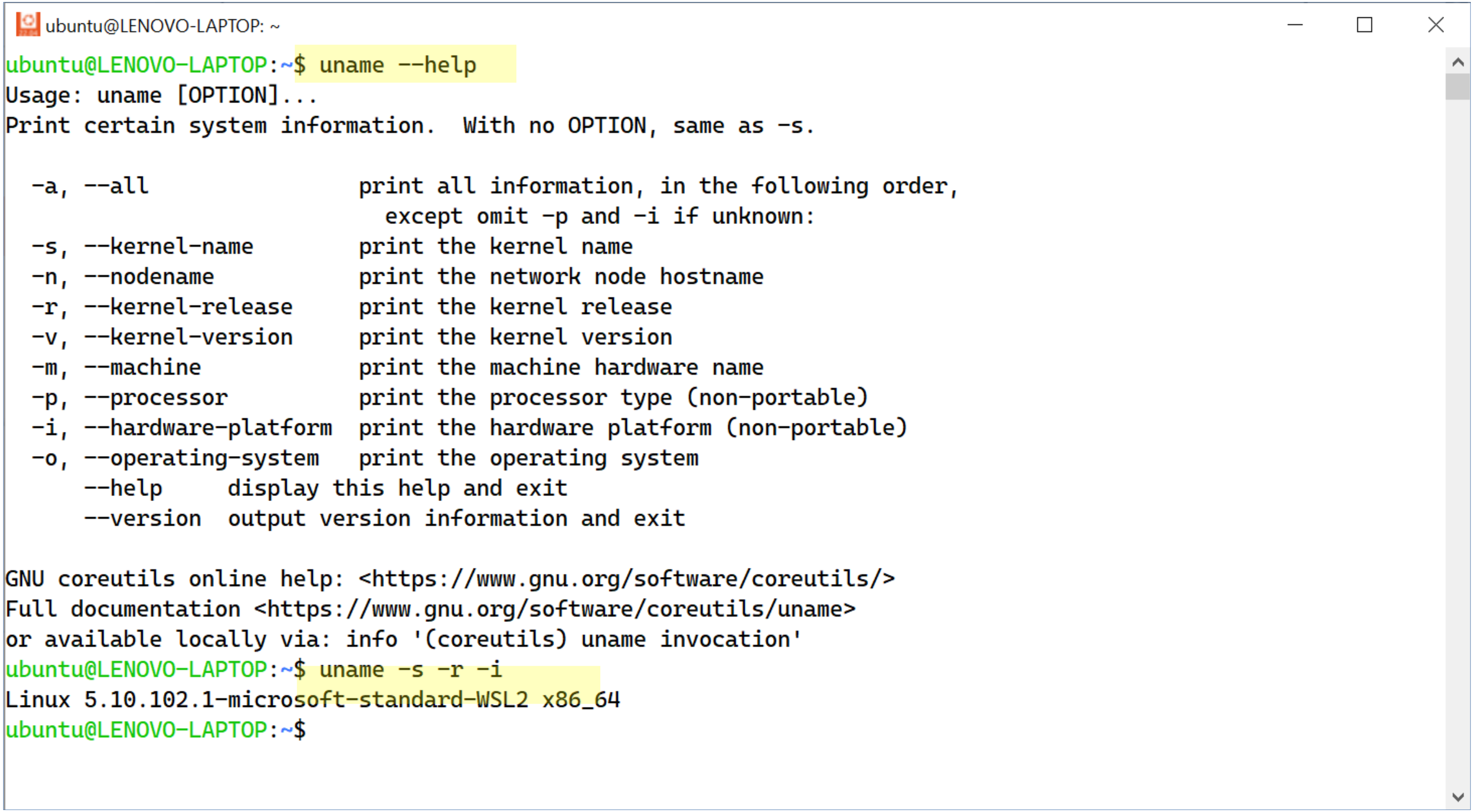
```
username@hostname:location$
```

`username`: the username of the current user who has logged in

`hostname`: the name of the system

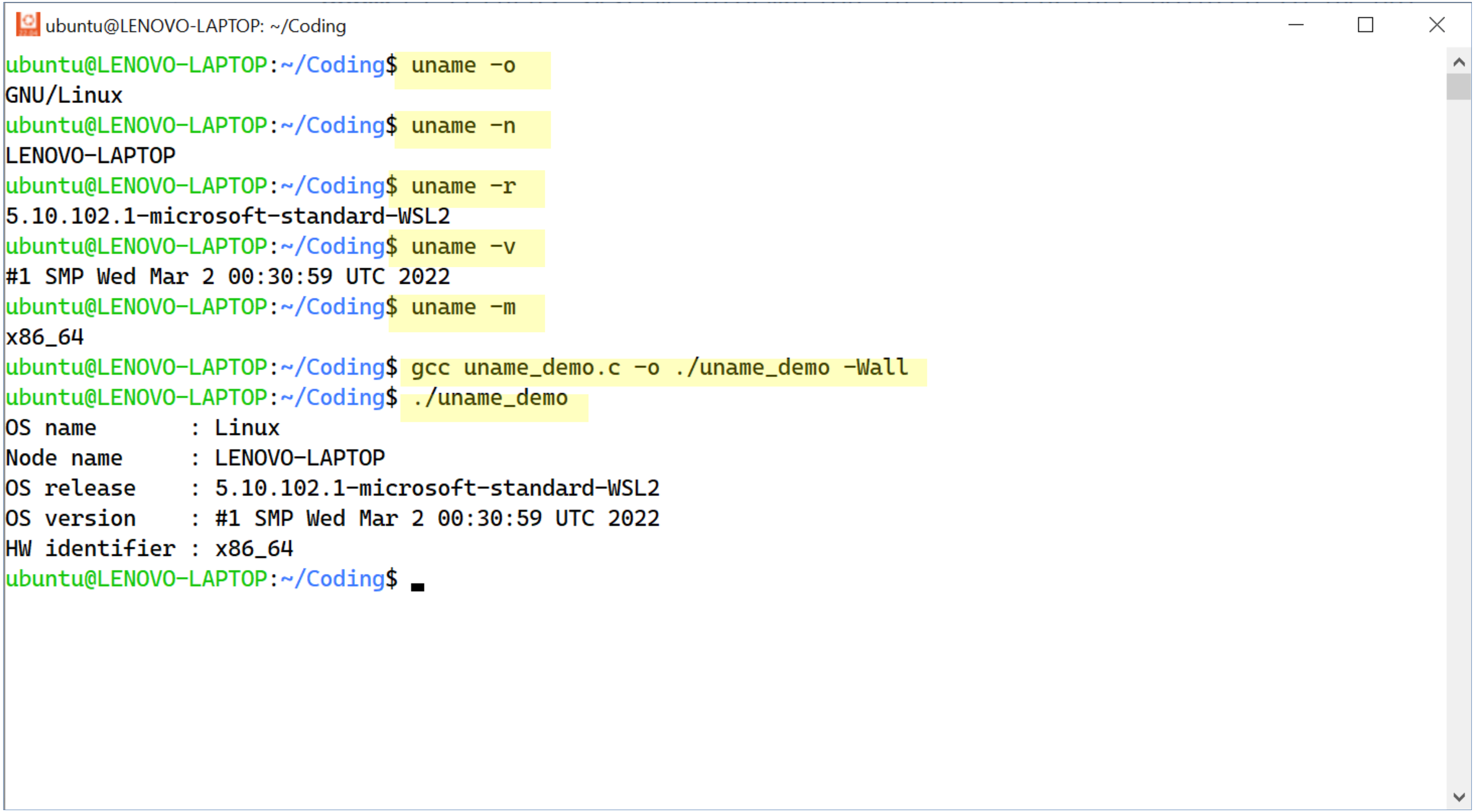
`location`: the current working directory

`$`: the end of prompt.

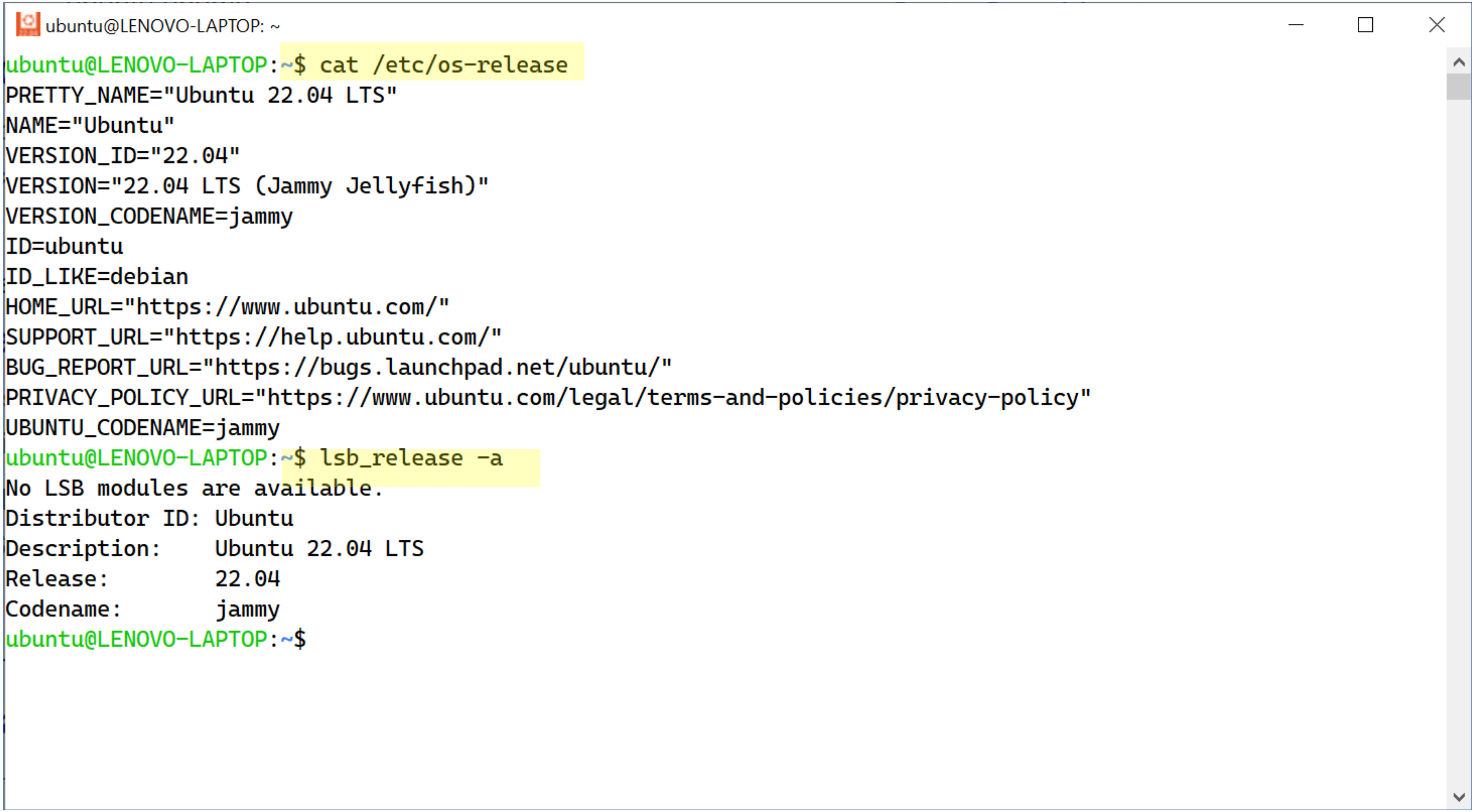


```
ubuntu@LENOVO-LAPTOP:~$ uname --help
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system   print the operating system
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/uname>
or available locally via: info '(coreutils) uname invocation'
ubuntu@LENOVO-LAPTOP:~$ uname -s -r -i
Linux 5.10.102.1-microsoft-standard-WSL2 x86_64
ubuntu@LENOVO-LAPTOP:~$
```

https://manpages.ubuntu.com/manpages/bionic/man2/uname.2.html



```
ubuntu@LENOVO-LAPTOP:~/Coding$ uname -o
GNU/Linux
ubuntu@LENOVO-LAPTOP:~/Coding$ uname -n
LENOVO-LAPTOP
ubuntu@LENOVO-LAPTOP:~/Coding$ uname -r
5.10.102.1-microsoft-standard-WSL2
ubuntu@LENOVO-LAPTOP:~/Coding$ uname -v
#1 SMP Wed Mar 2 00:30:59 UTC 2022
ubuntu@LENOVO-LAPTOP:~/Coding$ uname -m
x86_64
ubuntu@LENOVO-LAPTOP:~/Coding$ gcc uname_demo.c -o ./uname_demo -Wall
ubuntu@LENOVO-LAPTOP:~/Coding$ ./uname_demo
OS name       : Linux
Node name     : LENOVO-LAPTOP
OS release    : 5.10.102.1-microsoft-standard-WSL2
OS version    : #1 SMP Wed Mar 2 00:30:59 UTC 2022
HW identifier : x86_64
ubuntu@LENOVO-LAPTOP:~/Coding$ 
```



```
ubuntu@LENOVO-LAPTOP:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
ubuntu@LENOVO-LAPTOP:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04 LTS
Release:        22.04
Codename:       jammy
ubuntu@LENOVO-LAPTOP:~$
```

https://manpages.ubuntu.com/manpages/bionic/man1/lsb_release.1.html

Superuser Account

- `root` is the user name or account that by default has access to all commands and files on a Linux or other Unix-like operating system.
- The `root` user (also referred to as a superuser or root) has all the rights that are necessary to perform administrative tasks or access some files, execute privileged commands, and much more.
- The home directory of the root is `/root`.

Privileged vs. Root User

- The su command allows you to switch the user to someone else by providing its username. This command requires the root password.
- The sudo command allows a user belonging to the sudo group to run a command as root. This requires the user password.

```
# Make a new shell login as root.
# Note: Use the exit command to exit the shell after login.
# Method 1) This requires the root’s password.
$ su -
# Method 2) This requires the user’s password.
$ sudo su -
# or
$ sudo -i
```

Linux Built-in Commands

`man`	display the user manual or man pages of a Linux command.
`ls`	list files or directories in Linux file system.
`type`	find out whether it is built-in or external binary file.
`mkdir`	create or make new a directory.
`cd`	change the current working directory.
`pwd`	get the current working directory.
`grep`	search text and strings in a given file or standard input stream.
`cat`	create single or multiple files, view content of a file, concatenate files and redirect output in terminal or files.
`which`	locate the executable files or location of a program from file system.

Linux Built-in Commands

`locate` find files in the Linux file system using the specific file name.

`echo` display line of text/string that are passed as an argument.

`rm` delete files or directories.

`touch` create new files by giving file names as the input, or change and modify timestamps of a file.

`stat` give information about the file and file system (such as the size of the file, access permissions and the user ID and group ID,...).

`file` determine the type of a file and its data.

`readelf` display information about one or more ELF format object files.

`alias` create a custom shortcut used to represent a command.

`unalias` remove an alias specified as an argument.

The `rm` command

`rm -i` Ask before deleting each file.

`rm -r` Delete recursively a directory and all its contents.
Normally, `rm` will not delete directories, while `rmdir`
will only delete empty directories.

`rm -f` Force delete files without asking.

Demo: Execution of Commands

```
# create a new directory with a subdirectory.
$ mkdir -p ~/test/subdir/
# create an empty file.
$ touch ~/test/subdir/file-1.txt
# create a text file with a single-line text.
$ echo "Hello world!" >> ~/test/subdir/file-2.txt
# list all files and directories under ~/test/
$ ls -lr ~/test/*
# find all files with a .txt file extension under ~/test
$ find ~/test -name *.txt -type f
# remove the directory '~/test/' recursively.
$ rm -fr ~/test
```

Bash Version

To get the bash version number:

```
$ echo "${BASH_VERSION}"
$ bash --version | grep -i "version"
```

```
$ bash --version | head -n 1

GNU bash, version 5.1.16(1)-release (x86_64-pc-linux-gnu)
```

Environment & Shell Variables

Examples of **shells variables** for Linux:

`$HOSTNAME`

`$HOSTTYPE`

`$HOME`

`$LANG`

`$TERM`

`$SHELL`

`$DISPLAY`

`$PATH`

- A **shell variable** is a variable that is available only to the current shell. In contrast, an **environment variable** is available system wide and can be used by other applications on the system.
- The `echo` command can be used to display values of **shell variables and environment variables** in Linux.

Environment & Shell Variables

To list all the **environment variables** in Linux:

```
$ env
$ printenv
$ declare -xp
```

Set and Unset Variables

```
# set a session variable to a string value
$ MESSAGE="Hello World!"
# or set an environment variable
$ export MESSAGE="Hello World!"
# print the value of the variable
$ echo $MESSAGE
# unset the variable
$ unset MESSAGE
# search the variable
$ set | grep MESSAGE
$ printenv | grep MESSAGE
```

Question

What are the outputs of the following commands?

```
$ type ls
$ alias ll
$ file ~/.profile
$ which bash
$ type bash
```

```
$ whatis `which bash`
$ readelf -h `which bash`
$ echo $PATH
$ echo $PATH | tr ':' '\n'
$ echo $PATH | tr ':' '\n' | sort
```

Nano Editor

- nano is a lightweight terminal editor.
- It has been installed by default.
    - If not, run the following command to install the nano program.

```
$ sudo apt install nano -y
```

- To use the nano editor, run the following command

```
$ nano <text file>
```

- To make a bash script file executable and then run the script:

```
$ chmod +x <file.sh>
$ ./<file.sh>
```

*The caret or hat (^) preceding the command letter means you should hit **CTRL** first, followed by the key of your choice, say [X], to quit.*

Bash Script: Example 1

```bash
#!/usr/bin/env bash

echo "Run script: $0" # show the bash script name
echo "The number of arguments: $#"
if [ $# -eq 0 ]; then # no argument is passed.
   exit 1 # exit the script with 1.
else
   for arg in "$@"  # for each of arguments
   do
     if [ ! $arg == "" ] ; then # not empty.
       echo "$arg"
     else
       echo "This argument is an empty string."
     fi
  done
fi
```

```
$ bash ./ex-1.sh a b c d e

Run script: ./ex-1.sh
The number of arguments: 5
a
b
c
d
e
```

Bash Script’ Arguments

$0 the name and fullpath of the script executed in the terminal.

$1,$2,... the positional arguments passed to the script.

$# the number of positional arguments passed to the script.

$@ the positional arguments list.

$? the variable that can be used to determine whether
a command or script has executed successfully.
0=ok, 1=error

Bash Script: Example 2

```bash
#!/usr/bin/env bash

echo "Run script: $0"

echo "The number of arguments: $#"
if [ $# -eq 0 ]; then
    exit 1
else
    num_args=$#
    for ((i=1; i<=${num_args}; i++)); do
        echo "arg ${i}:"  \'$"${!i}"\'
    done
fi
```

```
$ bash ./ex-2.sh 1 2 3 a "hello"

Run script: ./ex-2.sh
The number of arguments: 5
arg 1: '1'
arg 2: '2'
arg 3: '3'
arg 4: 'a'
arg 5: 'hello'
```

Bash Script: Example 3

```
#!/usr/bin/env bash

if [ $# -ne 1 ] ; then
   exit 1 # only one argument is expected.
fi
case $1 in
   0)
     echo "The argument is 0 (zero)."
     ;;
   [1-9]|10)
     echo "The argument is between 1 and 10."
     ;;
   *)
     echo "others"
     ;;
esac
```

Bash Script: Example 4

```
#!/usr/bin/env bash

x=1
if [ $x -eq $x ]  ; then echo "equal"; fi
if test $x -eq $x ; then echo "equal"; fi
if (($x == $x))   ; then echo "equal"; fi
test $x -eq $x && echo "equal"

! test $x -ne $x  && echo "equal"
[[ ! $x -ne $x ]] && echo "equal"
```

Bash Script: Example 5

```
#!/usr/bin/env bash

x="1 "
if [[ $x -eq 1 ]]     ; then echo "x is equal to 1."; fi
if [[ "$x" -eq "1" ]] ; then echo "x is equal to 1."; fi

echo "$x-1"
echo "$((x-1))"
# Note: Anything inside $((...)) is considered to be
# an arithmetic operation.
```

```
$ bash ./ex-5.sh
x is equal to 1.
x is equal to 1.
1 -1
0
```

Brackets and Parentheses

- **Double Square Brackets** or `[[ ]]` for bash conditional expressions (e.g. string conditionals, pattern matching and file tests)
- **Double Parenthes**es or `(( ))` for arithmetic expressions and conditionals
- **Single Square Brackets** or `[ ]` similar to the POSIX test command. It is an alternative command for the test built-in command.

Bash Script: Example 6

```
#!/usr/bin/env bash

count=0              # set the count variable to 0
count=$((count+1)) # increment the count variable by 1

# while loop
while [ "$count" -le 5 ] ; do # if less than or equal to 5
   echo "The value of \$count is $count."
   let "count += 1"
done
```

```
$ bash ./ex-6.sh
The value of $count is 1.
The value of $count is 2.
The value of $count is 3.
The value of $count is 4.
The value of $count is 5.
```

Bash Script: Example 7

```
#!/usr/bin/env bash

for i in {1..10}
do
   echo "The value of \$i is $i."
   if [ $i -eq 5 ]
   then
      break
   fi
done
```

```
$ bash ./ex-7.sh
The value of $i is 1.
The value of $i is 2.
The value of $i is 3.
The value of $i is 4.
The value of $i is 5.
```

Bash Script: Example 8

```
#!/usr/bin/env bash

# check whether the wget command is available.
# if not, install the wget package.
if command -v wget &>/dev/null; then
  echo "The wget package is already installed."
else
  echo "Installing the wget package..."
  sudo apt update && sudo apt install -y wget
fi
```

```
#!/usr/bin/env bash

result=`which wget`
if [ $? -eq 0 ]; then
  echo "The package is already installed."
else
  echo "The package is not installed"
fi
```

```
#!/usr/bin/env bash

result=$(which wget)
if [ ! -z $result ]; then
  echo "The package is already installed."
else
  echo "The package is not installed"
fi
```

Bash Script: Example 9

```bash
#!/usr/bin/env bash

# create a function that can be used to check
# whether a command does exist.
command_exists () {
  command -v "$@" > /dev/null 2>&1
}
# get the code name of Ubuntu
if [ -z $(command_exists lsb_release) ] ; then
  codename=$(lsb_release --codename | cut -f2)
  echo "The Ubuntu code name is $codename."
else
  echo "Cannot determine the code name of Ubuntu..."
fi
```

Bash Script: Example 10

```
#!/usr/bin/env bash

# note: $RANDOM returns a random integer between 0..32767.
# create a random integer number between -10..+10.
let "x = $RANDOM % 21 - 10"
if [ "$x" -gt 0 ] ; then
  echo "$x is positive."
elif [ "$x" -eq 0 ] ; then
  echo "$x is zero."
elif [ "$x" -lt 0 ] ; then
  echo "$x is negative."
fi
# conditional executions
[[ $x -eq 0 ]] && echo "$x is zero."
[[ $x -ne 0 ]] && echo "$x is nonzero."
```

Bash Script: Example 11

```bash
#!/usr/bin/env bash

unset x
# note: x is unset and it will be expanded to an empty string.
[[ -v x ]] ; echo "The result is $?."
if [[ ! $x ]] ; then echo "x is an empty string or not set." ; fi

x="" # x is set as an empty string.
[[ -v x ]] ; echo "The result is $?."
if [[ ! $x ]] ; then echo "x is an empty string or not set." ; fi

x="hello"
[[ -v x ]] ; echo "The result is $?."
if [[ ! $x ]] ; then echo "x is an empty string or not set." ; fi
```

Bash Script: Example 12

```bash
#!/usr/bin/env bash

FILENAME=tmp-$(date +"%a-%d-%b-%Y-%k-%M-%S-%Z").txt
# create an empty file using the specified filename.
touch $FILENAME
# check if a file exists.
if [ -e "$FILENAME" ] ; then
    echo "$FILENAME exists."
else
    echo "$FILENAME does not exist."
fi
# remove the file
rm -f $FILENAME
```

Bash Script: Example 13

```
#!/usr/bin/env bash

# calculate 2 to the power of i, i=0...10
for i in {0..10}; do
   echo "2^i = $((1 << i))"
done
```

Bash Script: Example 14

```
#!/usr/bin/env bash

# generate a hex string of random data of 32 bytes
n=32
RAND=$(hexdump -n ${n} -v -e '/1 "%02X"' /dev/urandom)
echo $RAND
```

```
$ for i in {1..5} ; do bash ./ex-14.sh; done
A91DF21678E8A307802E3C3E0563DCB31A7E6C3CE573B25A1C51FB0C8721ED1F
9A26672A5BC1AC8B1BC6B694EAFB0BF41DAEC5747A1CC22E7CE9E69834BA1BB4
DED111A3614C7CE86EEE58EE8C3C42F49A9E55AB3DF98087FA24E2D6B7D75D90
0209AD21450F491DD7E5FFA4852902E881141CFDB0B6C90D7356D20A39840014
B7FE2B09A1D0981795E14EF268372D5D3DC0C1F0D3211DED28CF8E65B54E4B2D
```

Bash Script: Example 15

```
#!/usr/bin/env bash

answers="yes,no,ok,Yes,NO"

# split the string into an array (use ',' as the delimiter)
answers=($(echo $answers | tr ',' "\n"))
for ans in ${answers[@]} ; do
  case "$ans" in
     "yes")  echo "Yes" ;;
      "no")  echo "No"  ;;
         *)  echo "Invalid choice" ;;
  esac
done
```

Bash Script: Example 16

```
#!/usr/bin/env bash

DIRNAME="/etc/apt/"; FILENAME="sources.list"
FULL_NAME="${DIRNAME}${FILENAME}"

get_num_lines() { wc -l "${FULL_NAME}" | cut -d ' ' -f1 ; }

if [ -d "${DIRNAME}" ]; then # if the directory exists.
   # check whether the file specified by its full name exists.
   if [ ! -f "${FULL_NAME}" ]; then
      echo "${FULL_NAME} doesn't exist."
   else
      num_lines="$(get_num_lines)"  # execute the command
      echo "'${FULL_NAME}' has ${num_lines} lines."
   fi
else
   echo "${DIRNAME} doesn't exist."
fi
```

Sample Output

```
$ bash ./ex-16.sh

Host: ubuntu-desktop-vm
Date: Tue Aug  8 08:14:09 +07 2023
DateTime: 2023-08-08_08:14:09
>> ping 8.8.8.8
>> 3 packets transmitted, 3 received, 0% packet loss, time 2005ms
>> rtt min/avg/max/mdev = 67.496/82.004/89.728/10.266 ms
>> ping 9.9.9.9
>> 3 packets transmitted, 3 received, 0% packet loss, time 2004ms
>> rtt min/avg/max/mdev = 27.680/68.063/104.707/31.557 ms
>> ping 4.4.4.4
>> 3 packets transmitted, 0 received, 100% packet loss, time 2053ms
done...
```

Bash Script: Example 17

```bash
#!/usr/bin/env bash

echo "Host: $(hostname)" # show the $HOSTNAME environment
echo "Date: $(date)" # show the $DATE environment
echo "DateTime: $(date +%Y-%m-%d_%H:%M:%S)"

dns_servers=("8.8.8.8" "9.9.9.9" "4.4.4.4")
n="${#dns_servers[@]}"
for ((i=0; i < $n; i++)); do
    remote="${dns_servers[$i]}"
    echo ">> ping $remote"
    result=`ping "$remote" -c 3 | tail -n 2`
    readarray lines < <(echo -n "$result")
    for line in "${lines[@]}"; do
        printf ">> %s" "$line"
    done
    printf "\r\n"
done
echo "done..."
```

Bash Script: Example 18

```
#!/usr/bin/env bash

INSTALL_PKGS=""

if [ ! -x /usr/bin/curl ]; then
    INSTALL_PKGS="${INSTALL_PKGS} curl"
fi

if [ ! -x /usr/bin/wget ]; then
    INSTALL_PKGS="${INSTALL_PKGS} wget"
fi

if [ "X${INSTALL_PKGS}" != "X" ]; then
    echo "Installing packages: ${INSTALL_PKGS}..."
    sudo apt-get update
    sudo apt-get install -y ${INSTALL_PKGS} > /dev/null 2>&1
else
    echo "No packages to be installed.."
fi
```

Bash Script: Example 19

```bash
#!/usr/bin/env bash

TAR_FILENAME="gedit-40.0.tar.xz"
SHA256SUM_FILENAME="gedit-40.0.sha256sum"
if [ ! -f "${SHA256SUM_FILENAME}" ] ; then
   echo "The SHA256SUM file doesn't exist..."
   exit 1
fi

check_sha256sum() {
   SHA256SUM_OUTPUT=$(sha256sum "${TAR_FILENAME}" | cut -d' ' -f1)
   if [ "$CKSUM" = "$SHA256SUM_OUTPUT" ] ; then
      echo "0"
   else
      echo "1"
   fi
}
readarray lines < <(cat "${SHA256SUM_FILENAME}")
# Code continues on the next page...
```

Bash Script: Example 19 (cont’d)

```
num_lines=${#lines[@]}
for ((i=0; i < ${num_lines}; i++)); do
   args=(${lines[$i]})
   if [ ${#args[@]} -eq 2 ]; then
      CKSUM=${args[0]}; FILENAME=${args[1]}
      if [ "$FILENAME" = "$TAR_FILENAME" ]; then
          printf "File name: %s\n" "$FILENAME"
          printf "SHA256SUM: %s\n" "$CKSUM"
          retval=$(check_sha256sum)
          if [ $retval -eq  0 ] ; then
              echo "Checksum OK"
          else
              echo "Checksum FAILED"
          fi
          break
      fi
   fi
done
```

Using wget and sha256sum

```
# install wget and sha256sum
$ sudo apt install wget -y
$ sudo apt install hashalot -y
```

```
# download Gedit source code file and checksum file
$ mkdir -p $HOME/gedit-src && cd $HOME/gedit-src/
$ URL="https://download.gnome.org/sources/gedit/40"
# download the source code file (.tar.xz)
$ wget -c "${URL}/gedit-40.0.tar.xz"
# download the SHA256 checksum file
$ wget -c "${URL}/gedit-40.0.sha256sum"
# compute the SHA256 checksum for source code file
$ sha256sum ./gedit-40.0.tar.xz
0e8aac632b8879a57346aaf35c66f7df40c3fd5ea37a78e04ea218e41e3984e9  gedit-40.0.tar.xz
```