Introduction to Linux OS

CMPS 405

Linux Operating System

- **Linux** operating systems was started in 1991 by **Linus Torvalds**.
- Was inspired by Unix developed in the 1970s by AT&T Laboratories.
- Uses the same principles and basic ideas of Unix, but Linux itself **doesn’t contain Unix code**.
- Linux **is freely available** and is not backed by an individual company but by an international community of programmers.

Linux Distributions

- A Linux ***distribution*** is a bundle that consists of a Linux ***kernel*** and a selection of applications/Packages that are maintained by a company or user community.
- Distributions often include distribution-specific tools for software installation and system administration.
- Just like a distribution, a **package** is a bundle of software and a corresponding configuration and documentation that makes it easier for the user to install, update and use the software.
- Packages' Maintainers use a specific **package format** to specify how the software is installed on the operating system and how it is configured by default.

Linux Distributions

- **Debian** distribution family use the package manager **dpkg** to manage the software.
- **deb** package format is used for packages to be installed on Debian.
- The *Debian* ***GNU/Linux*** distribution is the biggest distribution of the Debian distribution family. A **very reliable** operating system.
- **Ubuntu** is another Debian-based distribution created by **Mark Shuttleworth** and his team in 2004. Is an easy to use Linux desktop.

Linux Distributions

- **Red Hat Enterprise Linux**, often abbreviated as **RHEL**. It is provided to companies as a reliable enterprise solution that is supported by Red Hat and comes with software that aims to ease the use of Linux in professional **server environments**. Some of its components require fee-based **subscriptions** or **licenses**.
- **CentOS** project uses the freely available source code of Red Hat Enterprise Linux and compiles it to a distribution which is available completely free of charge.
- **Fedora** is sometimes considered a test-bed for new technologies which later might be included in RHEL.
- All Red Hat based distributions use the package format **rpm**

Embedded Systems

- **Embedded systems** are a combination of computer hardware and software designed to have a specific function within a larger system.
- Usually they are **part of other devices** and help to control these devices.
- A variety of operating systems **based on the Linux kernel** was developed in order to be used in embedded systems.
- A significant part of smart devices have a Linux kernel based operating system running on it.
- Therefore, with embedded systems comes **embedded software**. The purpose of this software is to access the hardware and make it usable.
- Two of the most popular embedded software projects are **Android**, that is mainly used on mobile phones across a variety of vendors and **Raspbian**, which is used mainly on Raspberry Pi.

Linux on The Cloud

- The term ***cloud computing*** refers to a standardized way of consuming computing resources, either by buying them from a public cloud provider or by running a private cloud.
- As of 2017 reports, **Linux runs 90% of the public cloud workload**. Every cloud provider, from *Amazon Web Services* (**AWS**) to *Google Cloud Platform* (**GCP**), offers different forms of Linux. Even Microsoft, a company whose former CEO compared Linux to cancer, offers **Linux-based virtual machines** in their ***Azure*** cloud today.
- Linux is usually offered as part of *Infrastructure as a Service* (**IaaS**) offering. IaaS instances are virtual machines which are provisioned within minutes in the cloud.
- When starting an **IaaS instance**, an **image** is chosen which contains the data that is deployed to the new instance.
- **Cloud providers** offer various images containing ready to run installations of both popular Linux distributions as well as own versions of Linux.
- The **cloud user** chooses an image containing their preferred distribution and can access a cloud instance running this distribution shortly after.
- Most cloud providers add **tools** to their images to adjust the installation to a specific cloud instance. These tools can, for example, extend the file systems of the image to fit the actual hard disk of the virtual machine.

Software Packages

- Every Linux distribution offers a **pre-installed set of default applications**.
- In addition, a distribution has a **package repository** with a vast collection of applications available to install through its **package manager**.
- Several different package management systems exist for various distributions.
  - For instance, Debian, Ubuntu and Linux Mint use the **dpkg**, **apt-get** and apt tools to install software packages, generally referred as **DEB packages**.
  - Distributions such as Red Hat, Fedora and CentOS use the **rpm**, **yum** and **dnf** commands instead, which in turn install **RPM packages**.
  - The commands **dpkg** and **rpm** operate on individual package files.
  - In practice, almost all package management tasks are performed by the commands **apt-get** or **apt** on systems that use **DEB packages** or by **yum** or **dnf** on systems that use **RPM packages**.

Installing & Removing Software Packages

- Search for a DEB package
  $ apt-cache search *packageName*
- Install a DEB package with system's administer privileges
  $ sudo apt-get install *packageName*
- *Removing* an DEB package with system's administer privileges
  $ sudo apt-get remove *packageName*
- Search for an RPM package
  $ yum search *packageName*
- Install an RPM package with system's administer privileges
  $ sudo yum install *packageName*
- *Removing* an RPM package with system's administer privileges
  $ sudo yum remove *packageName*

Desktop Environments

- Two major desktop environments in the Linux world: **Gnome** and **KDE**.
- **Gnome** tries to follow the **KISS** ("**keep it simple stupid**") principle, with very streamlined and clean applications.
- **KDE** has larger selection of applications and giving the user the **opportunity to change every configuration setting in the environment**.
- **Gnome** applications are based on the **GTK *toolkit*** (written in the **C** language)
- **KDE** applications make use of the **Qt *library*** (written in **C++**).
- Applications written with the same graphical toolkit share a similar look and feel.
- Having the same shared graphical library for many frequently used applications may save some memory space at the same time that it will speed up loading time once the library has been loaded for the first time.

Command Line

- **Administrator** will always need to know how to work with the command line, or ***shell*** as it is called.
- The **shell** is a program that enables text based communication between the operating system and the user.
- There are several different shells on Linux, these are just a few:
  - Bourne-again shell (**Bash**)
  - C shell (csh or tcsh, the enhanced **csh**)
  - Korn shell (**ksh**)
  - Z shell (**zsh**)
- On Linux the most common one is the **Bash shell**.
- When using an interactive shell,
  - the user inputs **commands** at a so-called **prompt**.
  - For each Linux distribution, the default prompt may look a little different.

Command Line Structure

- Most commands at the command line follow the same basic structure:

  command [option(s)/parameter(s)...] [argument(s)...]

  $ ls -l /home

- The command is **ls**, the option/switch is **-l** is identical to **--format=long**, the data is **/home**.
- The only mandatory part of this structure is the command itself. In general, all other elements are optional, but a program may require certain options, parameters or arguments to be specified.
- Multiple options can be combined as well and for the short form, the letters can usually be typed together.

  $ ls -al

  $ ls -a -l

  $ ls --all --format=long

Command Behavior Types

- The shell supports two types of commands:
- **Internal**
  - These commands are **part of the shell itself** and are not separate programs. There are around **30** such commands. Their main purpose is executing tasks inside the shell (e.g. **cd**, **set**, **export**).
- **External**
  - These commands **reside in individual files**. These files are usually **binary programs or scripts**.
  - When a command which is not a shell builtin is run, the shell uses the **PATH** variable to search for an executable file with same name as the command.
    - The command **env** prints all environment variables to standard output.
  - In addition to programs which are installed with the distribution's package manager, users can create their own external commands as well.
- The command **type** shows what type a specific command is:

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

Learning About Commands

- **Built-in Help:** Most commands display a short overview of available commands when they are run with the **--help** parameter.
- **Man Pages:** Most commands provide a manual page or "**man**" page. This documentation is usually installed with the software and can be accessed with the **man** command.
  - **Each man page** is divided in **maximum of 11 sections**, though many of these sections are optional.
  - Man pages are organized in **eight categories**, numbered from **1 to 8**:
    - **User command, System calls, Functions of the C library, Drivers and device files, Configuration files and file formats, Games, Miscellaneous, System administrator commands, Kernel functions (not standard)**
- **Info Pages:** Another tool that will help you while working with the Linux system are the info pages. The **info** pages are usually more detailed than the man pages and are formatted in hypertext, similar to web pages on the Internet.
- The **/usr/share/doc/** directory stores most documentation of the commands that the system is using. This directory contains a directory for most packages installed on the system.

Variables

- In most Linux shells, there are two types of variables:
- **Local variables**
  - These variables are **available to the current shell process only**. They are **not inherited** by sub processes.
- **Environment variables**
  - These variables are **available both in a specific shell session and in sub processes spawned from that shell session**.
  - The majority of the environment variables are in capital letters (e.g. PATH, DATE, USER , HOME , PWD , SHELL). A set of default environment variables provide, for example, information about the user's home directory or terminal type.
- These types of variables are also known as **variable scope**.

Manipulating Variables

- You can **set up a local variable** by using the **=** (equal) operator without spacing before or after.

```
$ greeting=hello
```

- To **access the value of the variable** you will need to use **$** (dollar sign) in front of the variable's name.

```
$ echo $greeting
hello
$ echo $greeting world
hello world
$ bash -c 'echo $greeting world'
world
```

- In order to **remove a variable**, you will need to use the command **unset**:

```
$ unset greeting
```

- To **make a variable available to subprocesses**, turn it from a local into an environment variable using the command **export**

```
$ export greeting=hey
$ echo $greeting world
$ bash -c 'echo $greeting world'
```

Quoting

```
$ TWOWORDS="two words"
$ touch $TWOWORDS
$ ls -l
$ touch "$TWOWORDS"
$ ls -l
$ touch '$TWOWORDS'
$ ls –l
$ echo "I am $USER"
$ touch new file
$ ls –l
$ touch "new file"
$ ls -l
$ echo I am $USER
$ echo 'I am $USER'
$ echo $USER
$ echo \$USER
```

- In Bash, there are three types of quotes:
  - **Double quotes**
  - **Single quotes**
  - **Escape characters**
- **Double quotes** tell the shell to take the text in between the quote marks ("...") as **regular characters**.
  - All special characters lose their meaning, except the $ (dollar sign), \ (backslash) and ` (backquote). This means that variables, command substitution and arithmetic functions can still be used.
  - A space character, on the other hand, loses its meaning as an argument separator.
- **Single quotes** don't have the exceptions of the double quotes. They **revoke any special meaning from each character**.
  - Returns **the exact string without substituting the variable**.
- We can use **escape** characters to remove special meanings of characters from Bash.

>>

I/O Redirection and Command Line Pipes

- The Linux command line redirects the information through specific **standard channels:**
  - (stdin or channel 0), (stdout or channel 1), (stderr or channel 2).
  - I/O redirection enables the user to redirect information from or to a command by using a text file.

  $ echo "*Any Text*" > *text*

  $ echo "another text here…!" >> *text*

  $ *command* 2> *errorfile*

  $ *command* 2>> *errorfile*
- **Command Line Pipes:** The first command's output automatically becomes the second command's input by using the | (vertical bar) operator:

  $ ls -l | head | wc -w

Files and Directories

$ **tree**
$ **pwd**
$ **cd ~**
$ **ls**
$ **su**
$ **mkdir**
$ **find**
$ **touch**
$ **nano**
$ **cat**
$ vi
$ **mv**
$ **rm**
$ **rmdir**
$ **cp**
$ **sort**
$ **chmod**
$ **more**
$ **wc**
$ **locate**
$ **grep**
$ **echo** “any text to write to the given file” **>** file1

File and Directory Permissions

- The first 10 chars in the output of the command **ls -l** from left to right:
  - The first char if **d** it means directory and if **–** it means regular file
  - The next three chars are user's (**u**) permission in the order read (**r**) write (**w**) and execute (**x**).
    - The first char: **r** (read is permitted) or **–** (read is not permitted).
    - The second char: **w** (write is permitted) or **–** (write is not permitted).
    - The third char: **x** (execute is permitted) or **–** (execute is not permitted).
  - The next three chars are group's (**g**) permission in the same as before order read (**r**) write (**w**) and execute (**x**) with the same discussion.
  - The next three chars are other's (**o**) permission in the same as before order read (**r**) write (**w**) and execute (**x**) with the same discussion.
  - Hence a permission is set with the letters **r**, **w**, **x** or not set (denied) with **–**
  - Consider the permission let the bit **1** represent set and **0** represent not set (denied) permission then: **rwxr-x--x == 111101001 == 75**

Changing file & Directory permissions

- The command **chmod** is used to change the permissions of files and directories.
  - Use man chmod for more details.
  - Options used with command
    - Any combination of u, g, o meaning user, group, others Or a meaning all
    - + meaning grant permission Or – meaning deny permission
    - Any combination of r, w, x meaning read, write, execute
    - Decimal representation of the rwx premissions
  - Self-explanatory Examples:
    - chmode a+rwx *filename*
    - chmode g-rw *filename*
    - chmode o-wx *filename*
    - chmode o+x *filename*
    - chmode 777 *filename*
    - chmode 755 *filename*
    - chmode 751 *filename*

Searching and Extracting Data from Files

- **grep** command the abbreviation of the phrase "**global regular expression print**" and its main functionality is to search within files for the specified pattern.
  - The command outputs the line containing the specified pattern highlighted in **red**.
  - Most common **options**:

    -i the search is case insensitive.

    -r recursive (it searches into all files within the given directory and its subdirectories).

    -c the search counts the number of matches.

    -v invert the match, to print lines that do not match the search term.

    -E turns on extended regular expressions (needed by some of the more advanced meta-characters like **|** , **+** and **?**).

Regular Expressions

- When working with regular expressions, it is very important to keep in mind that ***every character counts*** and the **pattern** is written with the purpose of ***matching a specific sequence of characters***, known as a **string**.
- Regular expressions meta-characters used to form the patterns

.	Match any single character (except newline)
[abcABC]	Match any one character within the brackets
[^abcABC]	Match any one character except the ones in the brackets
[a-z]	Match any character in the range
[^a-z]	Match any character except the ones in the range
sun|*moon*	Find either of the listed strings
^	Start of a line
$	End of a line

(Study regexp1.sh)

Regular Expressions

- On top of the previous explained meta-characters, regular expressions also have **meta-characters** that enable multiplication of the previously specified pattern:

*****	Zero or more of the preceding pattern
+	One or more of the preceding pattern
?	Zero or one of the preceding pattern

- Since the these meta-characters are extended regular expression characters, we need to pass the **-E** option to the **grep** command.

  $ grep -E "ab**.+**" *text.txt*

Bash Scripting

- We have been learning to execute commands from the shell.
- You also can enter commands into a file, and then set that file to be executable and execute it with the bash interpreter. As a result, commands will run one after the other.
- These executable files are called scripts, and they are an absolutely crucial tool for any Linux system administrator.

```
$ echo 'echo "Hello World!"' > myscript.sh
$ chmod +x myscript.sh
$ ./myscript.sh
```

- All scripts should begin with a shebang, which defines the path to the interpreter.
- All scripts should include comments to describe their use. **(use #)**

Bash Scripting

- Recall our discussion of global and local variables.
- Variables must contain only alphanumeric characters or underscores, and are case sensitive. Username and username will be treated as separate variables.
- Variable substitution may also have the format ${username}, with the addition of the { }. This is also acceptable.
- Variables in Bash have an implicit type, and are considered strings. This means that performing math functions in Bash is more complicated than it would be in other programming languages such as C/C++. ***(Study myscript1.sh)***

Arguments

- Arguments can be passed to the script upon execution
- The variables such as $1, $2, … $9, contain the value of positional arguments 1, 2, … $9
- The variable $# contains the number of arguments.

Branching with if then else fi

if [*condition* **]**

then

statement(s)

else

statement(s)

fi

- Note the spaces between the square brackets and the logic contained.
- ***-eq : equals, -ne : Not equal to, -gt : Greater than, -ge: Greater than or equal to, -lt : Less than, -le : Less than or equal to.***

Branching with if then else fi

```
if [ condition ]
then
        statement(s)
elif [ condition ]
then
        statement(s)
else
        statement(s)
fi
```

- Note the spaces between the square brackets and the logic contained.
- ***-eq : equals, -ne : Not equal to, -gt : Greater than, -ge: Greater than or equal to, -lt : Less than, -le : Less than or equal to.***

Exit Codes

- Any execution of the command utility will return an exit code.
- An exit code will tell us if the command succeeded, or experienced an error.
  - An exit code of zero indicates that the command completed successfully. This is true for almost every Linux command that you work with.
  - Any other exit code will indicate an error of some kind.
  - The exit code of the last command to run will be stored in the variable **$?**

Handling Many Arguments

- There are two built-in variables which contain all arguments passed to the script**: $@** and **$***. For the most part, both behave the same.
- Bash will parse the arguments, and separate each argument when it encounters a space between them.
- Arrays in Bash can be created simply by putting space between elements.
  - Fruits="orange apple grapes pineapple strawberry"

for Loop

- To unpack a list and access each individual value, one after the other

```
for item in $list
do
    statement(s)
done
```

- Using -n with echo will suppress the newline after printing.
- The shift command will remove the first element of the arguments list

for Loop

- C/C++ like for loop

  for ((initial statement(s); termination condition; end of each iteration statement(s))
  do
  statement(s)
  done

- Notice the use of double parentheses.
- Double parentheses permits spacing and no “$” preceding variables.
- Initial and/or end of iteration statements can be separated by comma “,”.

While do Loop

- while do loop

```
while [ continuationCondition ]
do
    statement(s)
done
```

- Notice the space after the opening and before the ending testing prackets.
- Initial and/or end of iteration statements can be separated by comma “,”.

```
while (( continuationCondition ))
do
    statement(s)
done
```

- Double parentheses permits spacing and no “$” preceding variables.

until do Loop

- while do loop

```
until [ terminationCondition ]
do
    statement(s)
done
```

- Notice the space after the opening and before the ending testing prackets.
- Initial and/or end of iteration statements can be separated by comma “,”.

```
until (( terminationCondition ))
do
    statement(s)
done
```

- Double parentheses permits spacing and no “$” preceding variables.

Common practice

- Semicolon ";" is common used to allow more than one statement on the same line specially with the previous structures of the if, for, while, and until.
- ***Example 1***

```
if [ condition ]; then
        statement(s)
else
        statement(s)
fi
```

- ***Example 2***

```
for item in $list; do
        statement(s)
done
```

continue and break

- ***continue*** skips the rest of the code in the current iteration of the loop.
- ***break*** skips the rest of the entire loop
- Both ***continue*** and ***break*** can be used with all the previous types of loops

- Notice in the case of nested loops ***break*** breaks out of its loop only.
- ***break*** can take a parameter “say N” to break out of N level loops in the case of nested loops.

- Similarly, ***continue*** can have a parameter “say N” terminating all remaining iterations at its loop level and continues the next iteration at the loop, N levels above.