# ***ITDEFINED COURSE MATERIAL AWS DEVOPS***

**LINUX-BASICS**

[**Linux - Connect to AWS EC2**](https://itdefined.org/course/training-material/2/23/701/) **20/05/2024**

Create an ec2 instance (Server, Virtual Machine, Node)

Launch ec2 instance in AWS

1. Create an AWS account   
2. wait for the account to get activated   
3. Search for EC2 service -> Goto Instances -> Launch Instances   
    a. Give a name to the machine (instance)  
    b. Choose the OS type (AMI type)  
    c. Create and save the Key Pair to login   
4. Create an instance

**Note:** Follow the video to create ec2 machine.

[**Linux - ls, mkdir, touch, cd**](https://itdefined.org/course/training-material/2/23/704/) **21/05/2024**

Linux keynotes to know

In Linux commands are case-sensitive

A folder is called a Directory in Linux

**pwd (present working directory)**Know your current path/location

**ls (list files and directories)**

To list with details (long format): ls -l

To list with creation time sorted (recent at the top): ls -t

To list reverse order: ls -r

All the above together: ls -lrt (OR) ll

**cd (change directory - cd)**

switch/change to a folder/Directory in Linux  
Syntax: cd <path>/<location>  
 example: 1. To change to a Directory in the current location  
                        cd test

**mkdir (make directory - mkdir)**

To create a directory: mkdir <directory\_name>  
            example: mkdir test

To create multiple directories:

      mkdir <directory\_name> <directory\_name1> <directory\_name2> ..... <directory\_name>  
**Note:** directory names with space separated      
           example: mkdir test test1 test2 ...

**tree (structured and recursive directory listing)**

[Linux - which, VI editor, sudo, apt, yum](https://itdefined.org/course/training-material/2/23/707/) **22/05/2024**

**Which**

The which command in Linux is used to locate the executable file associated with a given command by searching in the directories listed in the environment variable PATH.

Syntax : which <command>

**Vi Editor**

Command Mode

we can't add any content. This is used to run vi-related commands.

Initially when we do vi to a file command mode is the default mode.

    Only Save the changes to the file -  ':w'

    To quit from vi - ':q!'

    Save and quit from vi - ':wq!' or (<escape> - Hold shift + zz)

    Save to another file - ':w <new\_file>'

    To search a word - '/<word>'

    To display line numbers - ':set nu'

    To replace word - s/<old\_word>/<new\_word>/g

    To display line numbers - ':nu'

    dd - cut the entire line where the cursor is

    yy - copy the entire line where the cursor is

    p - paste the clipboard content next to the cursor

Insert Mode

Is used to add the content to the file.

Press 'i' to enter to insert mode

Press '<escape>' to exit insert mode and move to command mode.

Linux path   
  Always Linux path is separate with /       
   . -> current directory   
   .. -> one step back / Previous one directory   
   ../.. -> Previous two directory

**tree**

The tree command in Linux is used to display the contents of current directory in a tree-like format. It recursively lists all files and directories within a specified directory.

Display Only Directories: tree -d

List All Files Including Hidden Files: tree -a

Limit the Depth of the Tree: tree -L 2

Display Full Path Prefix for Each File: tree -f

Print the Size of Each File: tree -h

**root**

Root is the superuser account in Unix and Linux which is created automatically at the time of OS installation.

The root user has complete and unrestricted access to the system, including the ability to modify or delete any file or directory and to start or stop any service.

It is a user account for administrative purposes and has the highest access rights on the system.

The default user ID of the root user is 0 (Zero)

**sudo**

The sudo command in Linux stands for "superuser do"

The user can execute any command as the superuser (root).

This is useful for performing administrative tasks that require elevated/root privileges.

sudo <command>

**Package Manager**

apt and yum are package management tools used in Linux distributions to install, update, and manage software packages.

apt is typically used in Debian-based distributions (like Ubuntu)

yum is used in RPM-based distributions (like CentOS and Fedora).

**Update Package List:**

sudo apt update

sudo yum check-update

**Upgrade all Installed Packages:**

sudo apt upgrade

sudo yum update

**Install a Package:**

sudo apt install <package\_name>

sudo yum install <package\_name>

**Remove a Package:**

sudo apt remove <package\_name>

sudo yum remove <package\_name>

**Remove a Package with Its Configuration Files:**

sudo apt purge <package\_name>

sudo yum purge <package\_name>

**Search for a Package:**

sudo apt search package\_name

sudo yum search package\_name

[Linux - rm, mv, cp, chmod, cat, head, tail](https://itdefined.org/course/training-material/2/23/710/) **23/05/2024**

**chmod**

The chmod command is used to change the access mode of a file.

The name is an abbreviation of change mode.

   example: 1. Read, write and execute permissions to the file owner:

                             chmod u+rwx <file\_name>

                                             (OR)

                             chmod 700 <file\_name>

                   2. Read, write to file owner, Read to groups and Wrtie, Execute to Others:

                             chmod u+rw, g+r, o+wx <file\_name>

                                                     (OR)

                             chmod 643 <file\_name>

**Delete files/directories (rm)**

Use the rm command to delete files and directories. But rm cannot simply delete a directory. Use “rm -r” to delete a directory. In this case, it deletes both the folder and the files in it.  
    To remove/delete a file       
        rm file\_name   
    To remove/delete a Directory      
        rm -r directory

**rename or move the file from one location to another (mv)**

**mv** command to move files and directories through the command line.

We can also use the mv command to rename a file/directory.

 To rename a file/directory  
        mv old\_filename new\_filename  
        mv old\_directory new\_directory

 To move to a different location (cut and paste)      
        mv <source> <destination>

**Copy the file from one location to another (cp)**cp command copies files and directories through the command line.

To copy to a different location

          cp <source> <destination>

[Linux - |, &&, ||, ;, >, >>, free, top, htop, df, du, who, whoami, w, last](https://itdefined.org/course/training-material/2/23/712/) **Fifth class 24/05/2024**

**free**

Free command is used to check the used and available space of physical memory and swap memory (ram/memory) in KB. The free command displays:

• Total amount of free and used physical memory

• Total amount of swap memory in the system

• Buffers and caches used by the kernel

du

“du” (Disk Usage) is a standard Unix/Linux command, used to check the information of disk usage of files and directories on a machine.

1) Using “-h” option with the “du” command provides results in “Human Readable Format“. This means you can see sizes in Bytes, Kilobytes, Megabytes, Gigabytes etc.

du -h

df

The command df stands for "disk filesystem". With the -h option (df -h) it shows the disk space in "human readable" form

top

The top command also gives you a real-time update on how much of your swap space is being used.

/proc/meminfo

cat /proc/meminfo will contain dynamic information about the kernel and the system.

**&&**

The ‘&&’ command in Bash is a logical operator that allows you to execute multiple commands in sequence.

The next command in the sequence will only execute if the previous command has been executed successfully.

command\_1 && command\_2

**||**

The ‘||’ command in Bash is a logical OR operator. It will execute the next command if the previous command fails.

command\_1 || command\_2

**| (pipe)**

The output of one command can given as input to another command

The Unix/Linux systems allow the stdout of a command to be connected to the stdin of another command.

The pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command

command\_1 | command\_2 | command\_3 | .... | command\_N

**>**(Write to a File using the Redirection Operator)

Redirection allows you to capture the output from a command and overwrite it as input to another command or file.

It will overwrite/Erase all the content of a file and add new content to the file.

Whatever the content it will written to a file and nothing will be printed in the console.

     echo "This is Redirection of text to file" > file.txt

**>>**(Append to a File using the double Redirection Operator)

Double Redirection allows you to capture the output from a command and send it as input to another command or file.

It will append the content to the end of the file by keeping the old content.

     echo "This is Redirection of text to file" >> file.txt

**tee**

To write to a file and also to print the written content in the console/terminal

     echo "This is Redirection of text to file" | tee file.txt

To append to a file and also to print the written content in the console/terminal

     echo "This is Redirection of text to file" | tee -a file.txt

                                        (OR)

     echo "This is Redirection of text to file" | tee --append file.txt

**uname**

uname command displays important information about the system such as — Kernel name, Host name, Kernel release number, Processor type, etc.,

    To check the linux version

        cat /etc/os-release (or) lsb\_release -a

**who**

who command is used to find out the following information:

1. Time of last system boot

    2. Current run level of the system

    3. List of logged-in users and more.

    a) To display all details of the current logged-in user: who -a

    b) To display a current run level of the system: who -r

    c) To show the time of the system when it booted last time: who -b -H

    d) To show a list of users logged in to the system: who -u

**whoami**

To display the system’s username

**w**

The **‘w’** command in Linux gives us important information about who is currently using the computer, how much the computer is being used, and what programs are running.

**last**

The 'last' command in Linux is used to display the history of last logged in users.

**lsb\_release -a (cat /etc/lsb-release)**

The *lsb\_release* command prints certain LSB (Linux Standard Base) and Distribution information.

[Linux - find, grep](https://itdefined.org/course/training-material/2/23/714/) **Sixth class 25/05/2024**

**find**

find command can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the ‘-exec’ other commands can be executed on files or folders found.

SYNTAX: find <location\_to\_find> [options]

   1) Search a file with specific name ->

       find . -name file.txt

   2) Search a file with specific name with ignorecase ->

       find . -iname file.txt

   3) search a files in multiple directories ->

       find . /home /user -name file.txt

   4) Search only files containing name ->

       find . -type f -iname file.txt

   5) Search only directories containing name ->

       find . -type d -iname file.txt

   6) Search for empty files and directories ->

       find . -empty

   7) Search for file with permissions(655) ->

       find . -perm 655

**grep**(To search pattern/string in a file or content)

‘global search for the regular expression’: The grep command is a filter that is used to search for lines

matching a specified pattern and print the matching lines to standard output.

1) Match all lines that start with ‘hello’.E.g:“hello there” ->

     grep “^hello” file1

2) Match all lines that end with ‘done’.E.g:“welldone” ->

     grep “done$” file1

3) Match all lines that contain any of the letters ‘a’,‘b’,‘c’,‘d’or‘e’ ->

     grep “[a-e]” file1

4) Match all lines that do not contain a vowel ->

     grep “[^aeiou]” file1

5) Match all lines that start with a digit following zero or more spaces. E.g:“1.”or“2.” ->

     grep “ \*[0-9]” file1

6) Searching in all files recursively using ->

     grep-r grep -r "ramesh" \*

7) Search multiple patterns ->

     grep -v -e "pattern" -e "pattern"

   8) Search text within multiple files.

       find./-typef-name"\*.txt"-execgrep'search\_string' {}\;

   9) find files by last modification time ->

       find . -mtime days

       # 24 hours -> find . -mtime 1

       # modified less than 7 days (7 days to till) -> find . -mtime -7

       # Last 50-100 Days Modified Files -> find . -mtime +50 –mtime -100

   10) Find Last 50 Days Accessed Files -> find . -atime 50

   11) Find Changed Files in Last 1 Hour -> find / -mmin -60

   12) Find Accessed Files in Last 1 Hour -> find / -amin -60

[Linux - sed, cut, awk, BASH](https://itdefined.org/course/training-material/2/23/717/) **7th class 27/05/2024**

**sed**

SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, finding and replacing, insertion or deletion.

Consider the following text file as the input file for all cases below. (file.txt)

    Unix is a great os. Unix is open source. Unix is a free os.

    learn operating systems.

    Unix Linux which one you choose.

    Unix is easy to learn. Unix is a multiuser os. Learn unix .unix is a powerful.

    1) Replacing or substituting string

        sed 's/unix/linux/' file.txt

    2) Replacing the nth occurrence of a pattern in a line

        sed 's/unix/linux/2' file.txt

    3) Replacing all the occurrences of the pattern in a line

        sed 's/unix/linux/g' file.txt

    4) Replacing from nth occurrence to all occurrences in a line

        sed 's/unix/linux/3g' file.txt

5) Replacing string on a specific line number

        sed '3 s/unix/linux/' file.txt

    6) Printing only the replaced lines

        sed -n 's/unix/linux/p' file.txt

    7) Deleting lines from a particular file

        To Delete a particular line, say 5 in this example

            sed '5d' file.txt

        To Delete a last line

            sed '$d' file.txt

        To Delete line from range x to y

            sed '3,6d' file.txt

        To Delete from the nth to the last line

            sed '12,$d' file.txt

        To Delete pattern-matching line

            sed '/pattern/d' file.txt

    8) Print a specific line from a file

            sed -n '2p' file.txt

**CUT**

The cut command in UNIX is a command for cutting out the sections column using a delimiter from each line of files and writing the result to standard output.

cut -d "delimiter" -f (field number) file.txt

    1) To cut the data with “ “ (space) as a delimiter and print the first column of data

        cut -d " " -f 1 state.txt

    2) To cut the data with “ “ (space) as delimiter and print first to fourth column data

        cut -d " " -f 1-4 state.txt

**awk**

The awk command in UNIX is a command for cutting out the sections column using a field separator from each line of files and writing the result to standard output.

    Consider the following text file as the input file for all cases below.

        $cat > employee.txt

          Ajay manager account 45000

          Sunil clerk account 25000

          varun manager sales 50000

          Amit manager account 47000

    1) Default behaviour of Awk: By default, Awk prints every line of data from the space-separated field of the file.

            awk '{print $1}' employee.txt

                            (or)

            awk -f " " '{print $1}' employee.txt

    2) To print the last column of a file using NF – Number of Fields/Columns

        How to get the last word from a line in the file.

            awk  ‘{print $NF}’ employee.txt

shebang (#!)

Used to tell the operating system kernel which interpreter to use to parse the rest of the file.

#!interpreter [arguments]

The directive must be the first line in the script.

The directive must start with shebang #!

White space after the shebang characters is optional.

Interpreter is the full path to a binary file (ex: /bin/sh, /bin/bash).

Interpreter arguments are optional.

Examples:

#!/bin/bash - Uses bash to parse the file.

#!/usr/bin/env perl - Uses the env command to find the path to the Perl executable.

#!/usr/bin/python Executes the file using the python binary.

[BASH Scripting - Variable, Escape character (\), Single quote ('') vs Double quote ("")](https://itdefined.org/course/training-material/2/23/719/) **28/05/2024Defining and Using Variables**

Variables are the containers which store data or a useful piece of information as the value inside.

### variable\_name=value

**Defining Variables**

No spaces around the equal sign (=)

* Variable names should be alphanumeric and can include underscores (\_)
* Variable names should not contain any operators
* # Correct
* variable=value
* variable\_name=value
* # Incorrect
* variable\_name = value
* variable-name = value
* variable$name = value

variable.name = value

**Accessing Variables**Use the dollar sign ($) before the variable name to access its value.

echo $variable\_name

**78**

>$ echo "\$var"

output: $var

If a \n pair appears, and the backslash itself is not quoted, a new line will be added.

echo -e "\nThis is to add newline before this echo"

Multiple newlines

echo -e "\nThis is \n to add newline \n before this echo\n"

**Single quote ('') vs Double quote ("")**

var=10

('')

Considers all inside single quotes as String

The $var variable output does not get replaced with the variable value but displays the variable name within the quotes, i.e., its literal form.

>$ echo '$var'

output: $var

("")

* Looks for assigned values to variables and executes the content.
* Running the echo command with double quotes expands the $var variable and outputs the assigned value instead of printing the characters within the quotes.

>$ echo "$var"

output: 10

[BASH Scripting - Special Variables](https://itdefined.org/course/training-material/2/23/722/) **29/05/2024**

# **Special Variables** These are special shell variables which are set internally by the shell and which are available to the user:

|  |  |
| --- | --- |
| **VARIABLE** | **DESCRIPTION** |
| **$0** | The filename of the current script. |
| **$n** | These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on). User flower braces for double or more digits like ${10}, ${100} |
| **$$** | The process ID of the current shell. For shell scripts, this is the process ID under which they are executing. |
| **$#** | The number of arguments supplied to a script. |
| **$@** | All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2. |
| **$\*** | All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2. |
| **$?** | The exit status of the last command executed. |
| **$!** | The process ID of the last background command. |
| **$\_** | The last argument of the previous command. |

[BASH Scripting - If condition, Arithmetic Operators](https://itdefined.org/course/training-material/2/23/726/) **30/05/2024**

### **Integer Comparison (int1 -operator int2)**

|  |  |
| --- | --- |
| **Operator** | **Purpose** |
| **-eq** | Integer equality |
| **-ne** | Integer inequality |
| **-lt** | Integer less than |
| **-le** | Integer less than or equal to |
| **-gt** | Integer greater than |
| **-ge** | Integer greater than or equal to |

**If Condition Single Brackets [ ] vs Double Brackets [[ ]]**

The single brackets [ ] are a symbolic link to the test command, which is a built-in command in most Unix-like operating systems including Bash.

Single brackets have fewer features compared to double brackets.

Single brackets are more portable across different shells, while [[ ]] is a Bash-specific construct.

It’s possible to use the comparison operators with the double brackets in newer versions of BASH-like (>, <, >=, <=, !=) with integers also.

# [BASH Scripting - If String checks, Comment, Command to variable](https://itdefined.org/course/training-material/2/23/728/) **BASH String Operations**

**# Replace first occurrence of 'World' with 'Universe'**  
new\_string=${string//World/Universe}

**# Replace all occurrences of 'o' with '0'**  
new\_string=${string//o/0}

**# Convert to lowercase**  
lowercase=${string,,}

**# Convert to uppercase**  
uppercase=${string^^}

**#number of characters**  
number\_of\_characters=${#string}

**# Extract substring from index 3 and extract 5 characters**  
substring=${string:3:5}

**# Extract the substring from the end**  
substring=${string: -5}

**Comparison Operators**

**Equal (==) or (=):** This operator tests if two strings are equal.

**Not Equal (!=):** It checks if two strings are not equal.

**Greater Than (>):** This operator compares two strings alphabetically and checks if the first string is greater than the second.

**Less Than (<):** It compares two strings alphabetically and checks if the first string is less than the second.

**Greater Than or Equal To (>=):** This operator checks if the first string is greater than or equal to the second.

**Less Than or Equal To (<=):** It checks if the first string is less than or equal to the second.

**Pattern matching (=~):** This operator checks if a string matches a specific pattern using regular expressions.

#!/bin/bash

string="$1"

#To check if string variable is equal to 'hi'

if [[ ${string,,} == 'hi' ]]; then

echo "string are equal"

fi

#To check if string variable contains/substring 'hi'

if [[ ${string,,} == \*"hi"\* ]]; then

echo "$string contains hi"

fi

#To check if string variable contains/substring 'hi'

if [[ ${string,,} =~ "hi" ]]; then

echo "$string contains hi"

fi

**To execute a command and store its output in a variable using command substitution.**

$(...) syntax

#!/bin/bash

# Execute the command and store its output in a variable

output1=$(ls -lrt)

echo "$output1"

`...` syntax (backticks)

#!/bin/bash

# Execute the command and store its output in a variable

output=`ls -lrt`

echo "$output"

The $() syntax is generally considered more readable than backticks.

$() syntax is easier to understand at a glance and would be less error-prone if you were to add more commands or nesting.

**comments  
Single-Line Comments:**

You can add single-line comments using the # character.

Everything after # on the same line is considered a comment.

#!/bin/bash

# This is a single-line comment

echo "Hello, world" # This is also a comment

#!/bin/bash

# This is a first-line comment

# Here's the second line of the comment

# And the third line of comment

echo "Hello, world"

**Multi-Line Comments:**

#!/bin/bash

: '

This is a long explation

multi comment

in bash

'

[BASH Scripting - IF Options](https://itdefined.org/course/training-material/2/23/730/)

* **Empty string (-z):** This operator checks if a string is empty.
* **Non-empty string (-n):** This operator checks if a string is non-empty.
* **Directory exists (-d):**To check if Directory exists and it is a directory.
* **File exists (-f):** To check if FILE exists and it is a directory.
* **Read permission (-r):**To check if FILE exists and the read permission is granted.
* **File size (-s):**To check if FILE exists and its size is greater than zero (which means that it is not empty).
* **Read permission (-w):**To check if FILE exists and the write permission is granted.
* **Read permission (-x):**To check if FILE exists and the execute permission is granted.

[BASH Scripting - for and while loopBASH Scripting - for and while loop](https://itdefined.org/course/training-material/2/23/732/)

**for loop**

**for:** This keyword initiates the for loop.

v**ariable:** This is a variable that will hold each value from the list as the loop iterates. You can choose any variable name you like.

**in:** This keyword separates the variable from the list of values.

**list:** This is a list of values that the loop will iterate over. It can be an array, a sequence of numbers, or a list of strings separated by spaces/lines.

**do:** This keyword marks the beginning of the loop body.

**commands to execute:** These are the commands that will be executed in each iteration of the loop. You can put any valid bash commands here.

**done:**This keyword marks the end of the loop.

**for loop BASH syntax**

for variable in list

do

# commands to execute

done

**for loop using an space seperated of string;**

#!/bin/bash

for n in a b c; do

echo $n

done

**for loop using an Array of string**

#!/bin/bash

declare -A fruits

fruits=(

[0]="Apple"

[1]="Mango"

[2]="Banana"

[3]="Orange"

)

for i in ${fruits[@]}; do

echo "fruits $i"

done

## **Range-based for loop**

#!/bin/bash

for n in {1..10}; do

echo $n

done

#!/bin/bash

# Print odd number series

for n in {1..15..2}; do

echo $n

done

**C Style for loop**

#!/bin/bash

for (( i=1 ; i<=10 ; i++ )); do

echo $i

done

## **Infinite for loop**

#!/bin/bash

n=0

for (( ; ; )); do

n=`expr $n + 1`

[ $n -eq 99 ] && exit

echo $n

done

**for loop on a command output**

#!/bin/bash

for i in $(find $1 -type f -iname "\*.sh"); do

if [[ -x "$i" ]]; then

echo "EXECUTABLE $(basename $i)"

else

echo "NOT EXECUTABLE $i"

sudo chmod 777 $i

if [[ -x "$i" ]]; then

echo "Changed the permission to EXECUTABLE: $i"

fi

fi

done

**while loop**

A **while loop** is a statement that iterates over a block of code till the condition specified is evaluated to false.

A while loop will run untill the condition is true and once condition is false it will stop.

Syntax

while [ condition ];

do

# statements

# commands

done

example

#!/bin/bash

var=7

while [[ $var -gt 0 ]]

do

echo $var

((var--))

done

while loop to read a file line by line

#!/bin/bash

while read line; do

echo $line

done < log.txt

@reboot CMD