**Bash basic**

Unix basic command

**Navigating**

* Cd <path> : change directory
* Ls <path> :print list of file + folder
* Pwd : current working directory

**Files and Dir**

* cp ‘/path/to/source/file.txt’ ‘/path/to/destination/’:copy file
* Mkdir <path> create dir
* Mv <dir of file> <new path>: moves a file
* Rm <path>: deletes a file
* Touch<file> : update last modified

**Pipes and filters**

* Cat: show content (single or batch, can count line)

**Local and global, envirionment variable**

* Local var only be used inside a function. Syntax : local var=$1
* Global var can be used inside a file. Syntax : var=$1
* Environment var can be used everywhere. Syntax : export var

**Declare**

declare option:

* -a: treats variable as an array
* -f: lists function name as and definition
* -F: lists just function names
* -i: makes variables integer types
* -r: makes variables read-only
* -x: exports variable names to subshells

Ex:

* declare -a nums=(45 33 100 65)

Variable expansion modifier

1. **${variable:-word}**: if variable is empty return word

variable="Hello" echo "${variable:-World}" # Output: Hello

unset variable

echo "${variable:-World}" # Output: World

1. **${variable:=word}**: If var is empty -> set word to var

echo "${variable:=Goodbye}" # Output: Goodbye

echo "$variable" # Output: Goodbye

variable="Hello"

echo "${variable:=Goodbye}" # Output: Hello

1. **${variable:+word}**: if var is not empty return word, when var is empty return nothing

echo "${variable:+World}" # Output: (nothing)

variable="Hello"

echo "${variable:+World}" # Output: World

1. **${variable:?word}**: print "word" and stop if var is not defined or empty

unset variable

echo "${variable:?Variable is not set}" # Output: Variable is not set variable="Hello"

echo "${variable:?Variable is not set}"

# Output: Hello

1. **${variable:offset}**: print start from offset

string="Hello World" echo "${string:6}" # Output: World

1. **${variable:offset:length}**: print start from x with length y (x:y)

string="Hello World" echo "${string:0:5}" # Output: Hello

**Positional parameter**

|  |  |
| --- | --- |
| $0 | References the name of the current shell script |
| $1-$9 | Positional parameters 1-9 |
| ${10} | Positional parameter 10 |
| $# | Evaluates to the number of positional parameters |
| $\* | Evaluates to all the positional parameters |
| $@ | Same as $\*, except when double quoted |
| “$\*” | Evaluates to “$1 $2 $3”, and so on |
| “$@” | Evaluates to “$1” “$2” “$3”, and so on |
| $? | Get result of previous command (0 – no error, 1 – error) |

**Awk** [**https://www.cyberciti.biz/faq/bash-scripting-using-awk/**](https://www.cyberciti.biz/faq/bash-scripting-using-awk/)

* Positional parameter

$0: whole line will be input

$1-9: each separated part will be in put

Default delimeter is blank or tab

EX: $0 -> name:john:age:18 (with delimeter is ‘:’. Syntax : -F':')

$1 -> name $2 -> john

* Print a Text File

awk '{ print }' /etc/passwd  
awk '{ print $0 }' /etc/passwd

* Use : as the input field separator and print first field only i.e. usernames (will print the the first field. all other fields are ignored), send output to sort command using a shell pipe:

awk -F':' '{ print $1 }' /etc/passwd | sort

* Print Lines Containing tom, jerry AND vivek

Print pattern possibly on separate lines:

awk '/tom|jerry|vivek/' /etc/passwd

* Print 1st Line From File

awk "NR==1{print;exit}" /etc/resolv.conf

awk "NR==$line{print;exit}" /etc/resolv.conf

* Simply Arithmetic

You get the sum of all the numbers in a column:

awk '{total += $1} END {print total}' earnings.txt

Shell cannot calculate with floating point numbers, but awk can:

awk 'BEGIN {printf "%.3f\n", 2005.50 / 3}'

* AWK and Shell Pipes

List your top 10 favorite commands:

history | awk '{print $2}' | sort | uniq -c | sort -rn | head

sort: sort from a to z

uniq –c : count duplicate line -> [number of iterations] [word]

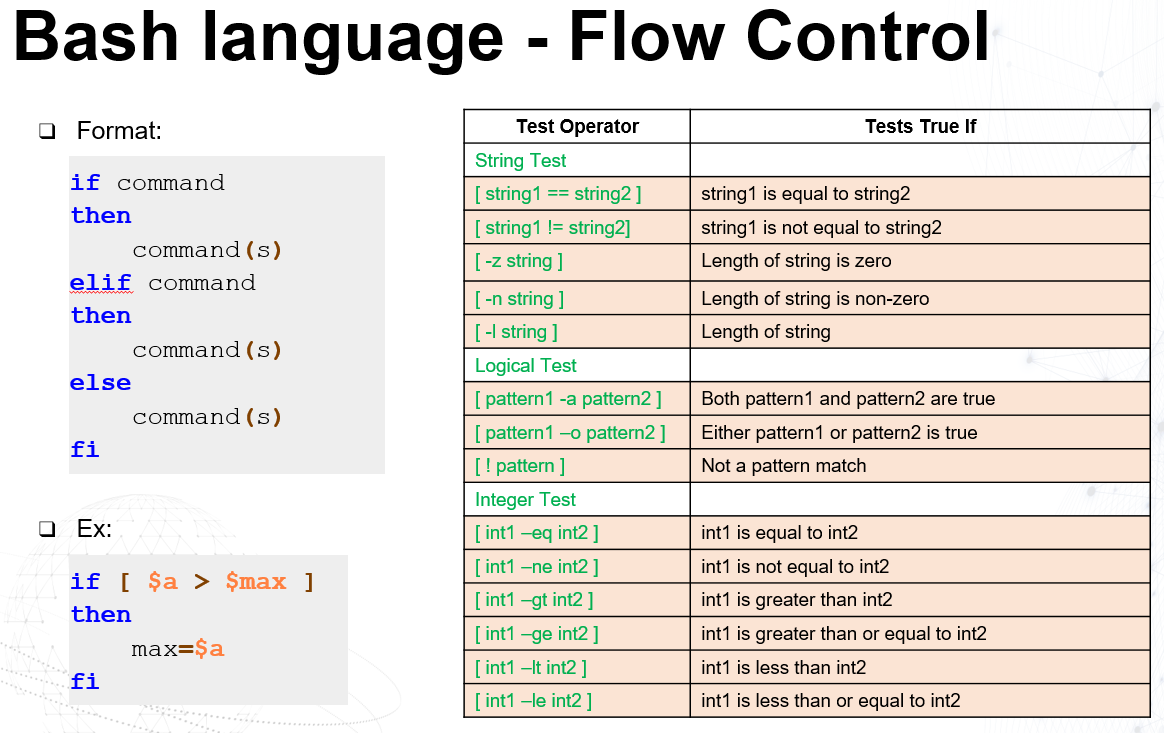
sort –rn : sort reverse number (from high to low)

head: take first 10 (default, can change with –n [num of line]) lines

* Another example to find out domain expiry date:

$ whois cyberciti.com | awk '/Registry Expiry Date:/ { print $4 }'

**If… then**



**Shebang**

* if the first line of a script is:

#!/bin/bash

It means the interpreter should be bash shell. If the first line is:

#!/bin/zsh

It means the interpreter to be used is Z shell.

There are several shells available for Linux and UNIX systems. While these shells have mostly common syntax, they do have different syntax or different way of handling things.

This is why it becomes important to specify the correct shell interpreter in the script, otherwise some scripts may produce different results when run in different shells.

<https://linuxhandbook.com/shebang/>

* How exactly the shebang work?

Basically, **#!/bin/zsh** is equivalent to:

/bin/zsh script\_name

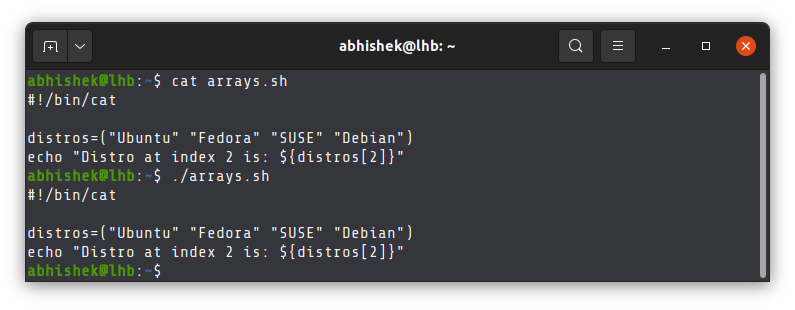
Replace **#!/bin/zsh** with **#!/bin/cat**. /bin/cat is the executable of the cat command:

#!/bin/cat

distros=("Ubuntu" "Fedora" "SUSE" "Debian")

echo "Distro at index 2 is: ${distros[2]}"

This means that now this script will run with **cat** command and display the content of the script.



It will work as long as it points to an executable command. If you put some random stuff, it will throw error.

Change the shebang line to:

#!/home/abhishek

Clearly, it does not point to the executable of any command and thus it throws a bad interpreter error.

abhishek@lhb:~$ cat arrays.sh

#!/home/abhishek

distros=("Ubuntu" "Fedora" "SUSE" "Debian")

echo "Distro at index 2 is: ${distros[2]}"

abhishek@lhb:~$ ./arrays.sh

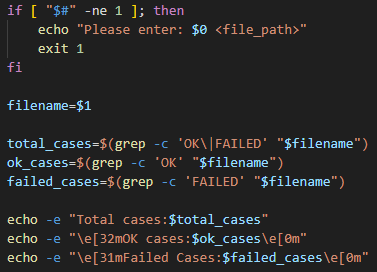
bash: ./arrays.sh: /home/abhishek: bad interpreter: Permission denied

**Conclusion**

Let me answer a few more questions before ending this article:

* It is important that there is no space between **#** and **!**. You CANNOT use it like this: **# !/bin/bash**.
* Most system allow space between **#!** and **/bin/bash** however it is a good practice to avoid space between **#!** and **/bin/bash**.
* **#!** has to be on the first line, otherwise the shebang line will be treated as comment. You cannot even have a blank line before it.

**Excerise 1**: !!! Open \*.sh using VS code or file will be broken due to Unicode8

****

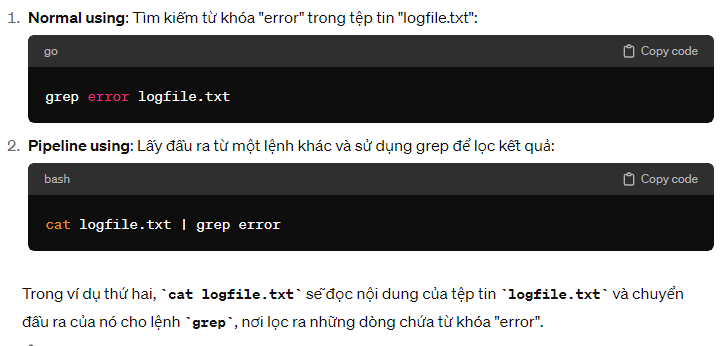
**Comparison**

1. **=**: So sánh bằng.
2. **-eq**: (equal).
3. **-gt**: (greater than).
4. **-lt**: (less than).
5. **-ge**: (greater than or equal to).
6. **-le**: (less than or equal to).
7. **!=**: (not equal).
8. **-z**: (empty string).
9. **-n**: (non-empty string).
10. –nq : not equal

Ex:

* **if [ "$a" -eq "$b" ]**: is **a** == **b** ?.
* **if [ "$a" -lt "$b" ]**: Is **a** < **b** ?.
* **if [ -z "$var" ]**: Check if **var** is empty or not.

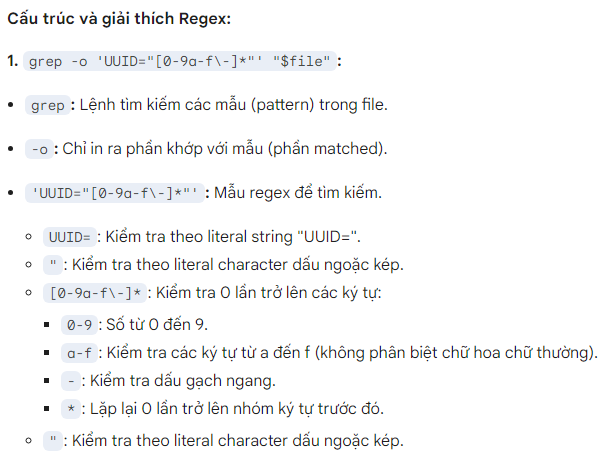
**Regular Expression**

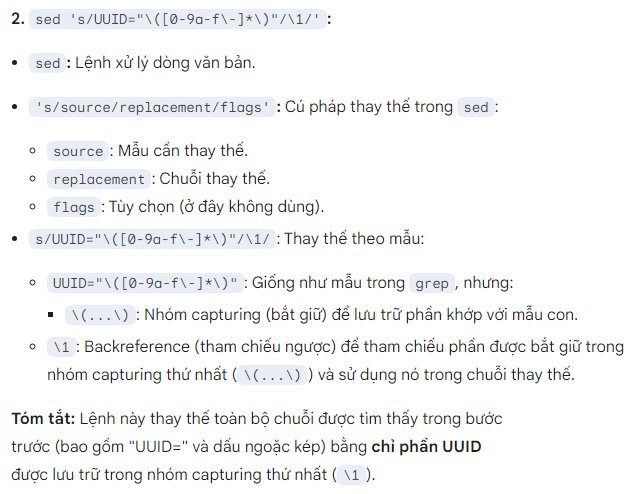


Command:  
file="99\_exercise\_00\_BASH\_New\_Training\_TCODE\_SampleApp\_cdf\_App\_ADC\_U2AEVA1\_BGA516.arxml"

# Use grep to find lines containing UUID and then use sed to extract the UUID

grep -o 'UUID="[0-9a-f\-]\*"' "$file" | sed 's/UUID="\([0-9a-f\-]\*\)"/\1/'





**grep -E <word> <filename>:**

For example: grep -E 'pattern' file.txt

Explanation: Search for a word or pattern using an extended regular expression. This allows you to use more complex regular expressions than standard grep.

**grep -i <word> <filename>:**

For example: grep -i 'error' file.txt

Explanation: Searches for a word or pattern without regard to case sensitivity.

**grep -v <word> <filename>:**

For example: grep -v 'pattern' file.txt

Explanation: Selects lines that do not contain the specified word or pattern.

**grep -c <word> <filename>: (we are going to use this to count)**

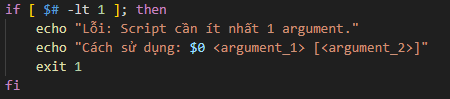
For example: grep -c 'pattern' file.txt

Explanation: Prints the number of lines in the file that match the specified word or pattern, instead of printing the actual lines.

**Coloring:**

* \e[32m…\e[0m : green
* \e[31m…\e[0m : red

**Checking number of input file**

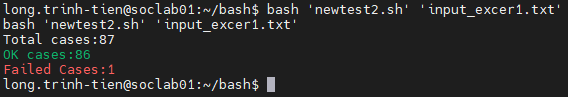
****

"$#": The special variable stores the number of arguments passed to the script.

-ne: Comparison operator, check if the value on the left is different from the value on the right.  
$0: The special variable that stores the name of the running script.

fi: Punctuation to end the if conditional command block.

Result:



**Exercise 2:**

! when input a value, do not use space between <variable>/<value> and ‘=’: ‘var=<value>’ not ‘~~var = <value>~~’

\r : return to the beginning of line

Breakdown ex 2 :

* Display process bar

print\_progress() {

    local iteration=$1

    local total=$2

    local decimals=${3:-1}

    local length=${4:-100}

    local fill=${5:-'█'}

    percent=$(bc -l <<< "scale=2; $iteration/$total\*100.00")

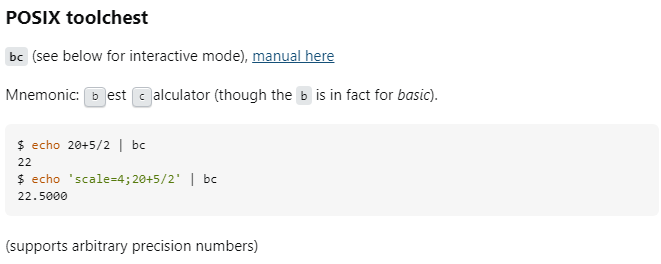
    filledLength=$(bc <<< "$length\*$iteration/$total")

    bar=$(printf "%${filledLength}s" | tr ' ' "$fill")

    empty=$(printf "%$(bc <<< "$length-$filledLength")s"| tr ' ' '-')

    printf "|%s%s| %s%%\r" "$bar" "$empty" "$percent"

}

Bc –l : [https://unix.stackexchange.com/questions/40786/how-to-do-integer-float-calculations-in-bash-or-other-languages-frameworks](https://unix.stackexchange.com/questions/40786/how-to-do-integer-float-calculations-in-bash-or-other-languages-frameworks)

* Create large file

Read an amount of byte: head -c <number\_of\_bytes> /dev/urandom

* Use apend ‘>>’ to write at the last part of file. Ex:

head -c 1024 /dev/urandom >> “file\_name.txt”

Write an amount of byte to a file: dd if=/dev/urandom of=file\_name bs=<number\_of\_bytes> count=1

Dd: data definition

If : input file

Of : output file

Bs: blocksize

Count: number of block (ex: blocksize =1MB, count = 2 means 2MB will be taken from input to write on output)

write\_large\_file(){

    file\_name="sample\_file.txt"

    file\_size=$((30 \* 1024 \* 1024)) # 30MB

    file\_smallest\_part=$((1024 \* 1024)) # 1MB = 1024 KBs \* 1024 bytes

    total\_step=$(($file\_size / $file\_smallest\_part))

    #Delete exist file (if any)

    if [ -f "$file\_name" ]; then

        rm $file\_name

    fi

    echo "Creating a file name \"sample\_file.txt\" with random contents (size is $((file\_size/(1024\*1024))) MB)"

    for i in $(seq 1 $total\_step); do

        # Generate random content and append to file

        head -c $file\_smallest\_part /dev/urandom >> "$file\_name"

        # Update progress bar

        print\_progress -i $i -t $total\_step -d 1 -l 50 -f '█'

    done

    echo -ne '\n' # Move to a new line after completion

    # Check if the file exists and its size

    if [ -f "$file\_name" ]; then

        echo "$file\_name has been created."

        file\_result\_size=$(stat -c%s "$file\_name")

        echo "Size: $((file\_result\_size/(1024\*1024))) MB"

    else

        echo "Failed to create the file."

    fi

}

* Input

While…do

while [[condition]]; do

# TODO

done

Case..in

month\_number=5

case $month\_number in

1) echo "Tháng 1" ;;

2) echo "Tháng 2" ;;

...

12) echo "Tháng 12" ;;

\*) echo "Số tháng không hợp lệ" ;;

Esac

* Tháng 5

(echo –e to use \n \t as end line or tab)

(echo –n to keep line stay so that it will not go to the next line)

    while [[ "$#" -gt 0 ]]; do

        case "$1" in

            -i) iteration="$2"; shift 2;;

            -t) total="$2"; shift 2;;

            -d) decimals="$2"; shift 2;;

            -l) length="$2"; shift 2;;

            -f) fill="$2"; shift 2;;

            \*) echo "Invalid argument: $1"; exit 1;;

        esac

    done

Exercise 3

[$char =~ [0-9]]: check if 0-9 appear anywhere in $char string

[$char == [0-9]]: check if 0-9 is equal exactly to $char string

Other source:  
[Các lệnh SSH command cơ bản](https://www.hostinger.vn/huong-dan/lenh-ssh-co-ban#:~:text=SSH%20Command%20trong%20Linux,-SSH%20Command%20trong&text=SSH%20Command%20d%C3%B9ng%20m%E1%BB%99t%20user,%E1%BB%A9ng%20d%E1%BB%A5ng%2C%20qu%E1%BA%A3n%20l%C3%BD%20server.)

<https://xuanthulab.net/ssh-secure-shell/>