***CONCEPTS***

* **SCRIPT1**

#!/bin/bash

echo "Scripting is fun!"

* **TO GET CURRENT DATE: DATE=$(date +%F)**
* **TIMESTAMP $(date +"%Y-%m-%d %T")**
* **Rename File: mv $file $renameFile**
* **To Count No. of files in a dir ls $dir | wc -l**
* **PID "$$"**
* **$RANDOM for random number**
* **DEFINE VARIABLES:**
* **MY\_SHELL="bash"**
* **ACCESS:**
* $Variable\_name--- **echo "I like the ${MY\_SHELL} shell."**
* if we want to add something after the Variable\_name then use
  + - **echo "I am ${MY\_SHELL}ing on my keyboard."**
* **SERVER\_NAME=$(hostname)**

**echo "You are running this script on ${SERVER\_NAME}."**

* To assign the output of the "hostname" command to the variable SERVER\_NAME?
* **SERVER\_NAME=`hostname`**

**echo "You are running this script on ${SERVER\_NAME}."**

* **SERVER\_NAME =$( hostname)."**
* **FILE OPERATORS**
  + **-d FILE-** True if file is a directory
  + **-e FILE-**True if file exists
  + **-f FILE-** True is file exists and is a regular file
  + **-r FILE-** if file is readable by you
  + **-s FILE**- if file exists and not empty
  + **-w FILE-** if file is writable by you
  + **-x FILE-** if file is executable by you.
  + **ls -l $file –** to list all files
* **STRING OPEARTORS:**
* **-z STRING-** true if string is empty
* **-n STRING-** true if string is not empty
* **STRING1=STRING2—** true if equal
* **STRING1!**=STRING2— true if not equal
* **ARITHMETIC OPERATORS:**
* **arg1 -eq arg2—** if both are equal
* **arg1 -ne arg2--** if both are not equal
* **arg1 -le arg2--** if arg1 is less than agr2
* **arg1 -lt arg2--** if arg1 is less equal to arg2
* **arg1 -gt arg2--** if arg1 is greater than arg2
* **arg1 -ge arg2--** if arg1 is greater than equal to arg2
* **CONDITIONAL STATEMENTS:**
  + MY\_SHELL="bash"

if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

fi

* if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

else

echo "You don't seem to like the bash shell."

fi

* MY\_SHELL="csh"

if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

elif [ "$MY\_SHELL" = "csh" ]

then

echo "You seem to like the csh shell."

else

echo "You don't seem to like the bash or csh shells."

fi

* **LOOPS:**
* for COLOR in red green blue

do

echo "COLOR: $COLOR"

done

* COLORS="red green blue"

for COLOR in $COLORS

do

echo "COLOR: $COLOR"

done

* **POSITIONAL PARAMETERS:**
* $0, $1, $2
* $@--- if we don’t know how much arguments user will enter then use ths, we can give the arguments in command line, accrdngly it will take

**Example ---- $ ./add-user.sh tom richard harry**

* **To take argument from user**----- USER=$1
* USER=$1 # The first parameter is the user.

echo "Executing script: $0"

echo "Archiving user: $USER"

# Lock the account

passwd -l $USER

# Create an archive of the home directory.

tar cf /archives/${USER}.tar.gz /home/${USER}

* for USER in $@

do

echo "Archiving user: $USER"

# Lock the account

passwd -l $USER

# Create an archive of the home directory.

tar cf /archives/${USER}.tar.gz /home/${USER}

done

* **ACCEPTING USER INPUTS:**
* **read -p “Enter a username:” USER**
* read -p "Enter a user name: " USER

echo "Archiving user: $USER"

# Lock the account

passwd -l $USER

# Create an archive of the home directory.

tar cf /archives/${USER}.tar.gz /home/${USER}

* **CHECKING EXIT STATUSES:**
  + **“$?” contains the return code of the previously executed command**

**Example: ls /not/here**

**echo “$?”**

* + **All command returns an exit status**
  + **0-255**
  + **0=success**
  + **Other than 0= error condition**
  + **$?= contains the exit status**
  + **Decision making=if, &&, ||**
  + **Exit- to control exit status of your scripts**
  + **We can use semicolon to write some commands in one line, commands separated by a semicolon will be executed no matter the exit status of the previous command**.

**ls -l dir; hostname**

**EXAMPLES:**

1. HOST="google.com"

ping -c 1 $HOST **#c means one packet**

**if [ "$?" -eq "0" ]**

then

echo "$HOST reachable."

else

echo "$HOST unreachable."

fi

1. HOST="google.com"

ping -c 1 $HOST

**if [ "$?" -ne "0" ]**

then

echo "$HOST unreachable."

fi

1. HOST="google.com"

ping -c 1 $HOST

**RETURN\_CODE=$?**

if [ "$RETURN\_CODE" -ne "0" ]

then

echo "$HOST unreachable."

fi

1. HOST="google.com"

**ping -c 1 $HOST && echo "$HOST reachable."**

1. HOST="google.com"

**ping -c 1 $HOST || echo "$ HOST unreachable."**

1. HOST="google.com"

ping -c 1 $HOST

**if [ "$?" -ne "0"]**

then

echo "$HOST unreachable."

**exit 1**

fi

**exit 0**

* **FUNCTIONS**
* function hello() {

echo "Hello!"

}

hello **# calling function**

* function hello() {

echo "Hello!"

now

}

function now() {

echo "It's $(date +%r)"

}

hello **# calling function**

* function hello() {

echo "Hello $1" **# passing argument**

}

hello Jason **# calling function by passing argument**

* function hello() {

for NAME in $@

do

echo "Hello $NAME"

done

}

hello Jason Dan Ryan **# calling function by passing number of s arguments**

* my\_function() {

echo "$GLOBAL\_VAR"

}

GLOBAL\_VAR=1

**# The value of GLOBAL\_VAR is available to my\_function**

my\_function

* my\_function() {

echo "$GLOBAL\_VAR"

}

**# The value of GLOBAL\_VAR is NOT available to my\_function since GLOBAL\_VAR was defined after my\_function was called.**

my\_function

GLOBAL\_VAR=1

* my\_function() {

GLOBAL\_VAR=1

}

**# GLOBAL\_VAR not available yet.**

echo "GLOBAL\_VAR value BEFORE my\_function called: $GLOBAL\_VAR"

my\_function

**# GLOBAL\_VAR is NOW available**

echo "GLOBAL\_VAR value AFTER my\_function called: $GLOBAL\_VAR"

* my\_function() {

**local** LOCAL\_VAR=1 **#local keyword used**

echo "LOCAL\_VAR can be accessed inside of the function: $LOCAL\_VAR"

}

my\_function

**# LOCAL\_VAR is not available outside of the function.**

echo "LOCAL\_VAR can NOT be accessed outside of the function: $LOCAL\_VAR"

* function backup\_file () {

**# This function creates a backup of a file.**

**# Make sure the file exists.**

if [ -f "$1" ]

then

**# Make the BACKUP\_FILE variable local to this function.**

local BACKUP\_FILE="/tmp/$(basename ${1}).$(date +%F).$$"

**#$$ is used to get the ProcessID**

echo "Backing up $1 to ${BACKUP\_FILE}"

**# The exit status of the function will be the exit status of the cp command.**

cp $1 $BACKUP\_FILE

else

**# The file does not exist, so return an non-zero exit status.**

return 1

fi

}

**# Call the function**

backup\_file /etc/hosts

**# Make a decision based on the exit status of the function.**

**# Note this is for demonstration purposes. You could have**

**# put this functionality inside of the backup\_file function.**

if [ $? -eq "0" ]

then

echo "Backup succeeded!"

else

echo "Backup failed!"

**# Abort the script and return a non-zero exit status.**

exit 1

fi

* **WILDCARDS**
  + **\* -- matches 0 or more characters**
  + **?— matches exactly one character**
  + **?? – matches exactly two characters**
  + **[]-matches all chars that are included in the brackets, ex: ca[nt]\***
  + **[!]—matches all chars that are not included in the brackets**
  + **[0-3]—range of characters**
  + **[[:digit:]]**
  + **[[:alpha:]]**
  + **[[:alnum:]]**
  + **[[:upper:]]**
  + **[[:lower:]]**
  + **[[:space:]]**
  + **\ -- escape char to escape wildcard,**
* **ex: \*\?—matches all chars and a question mark at end (done?)**

**EXAMPLES:**

* cd /var/www

for FILE in \*.html

do

echo "Copying $FILE"

cp $FILE /var/www-just-html

done

* for FILE in /var/www/\*.html

do

echo "Copying $FILE"

cp $FILE /var/www-just-html

done

* for FILE in \*.html

**# This will loop through all the "html" files in the current directory.**

do

echo "Copying $FILE"

cp $FILE /var/www-just-html

done

* **CASE STATEMENTS:**
* case "$1" in

start)

/usr/sbin/sshd

;;

stop)

kill $(cat /var/run/sshd.pid)

;;

esac

* case "$1" in

start)

/usr/sbin/sshd

;;

stop)

kill $(cat /var/run/sshd.pid)

;;

\*)

echo "Usage: $0 start|stop" ; exit 1

;;

esac

* case "$1" in

start)

/usr/sbin/sshd

;;

stop)

kill $(cat /var/run/sshd.pid)

;;

\*)

echo "Usage: $0 start|stop" ; exit 1

;;

esac

* case "$1" in

start|START)

/usr/sbin/sshd

;;

stop|STOP)

kill $(cat /var/run/sshd.pid)

;;

\*)

echo "Usage: $0 start|stop" ; exit 1

;;

esac

* read -p "Enter y or n:" ANSWER

case "$ANSWER" in

[yY]|[yY][eE][sS])

echo "You answered yes."

;;

[nN]|[nN][oO])

echo "You answered no."

;;

\*)

echo "Invalid answer."

;;

esac

* read -p "Enter y or n: " ANSWER

case "$ANSWER" in

[yY]\*)

echo "You answered yes."

;;

\*)

echo "You answered something else."

;;

Esac

* **LOGGING**
* **The syslog uses facilities and severities to categorizing messages**
* **Facilities: kern, user, mail, daemon, local0, local7**
* **Severities: emerg, alert, crit, err, warning, notice, info, debug**
* **Logger utility**
* **By default creates user.notice messages**
* **If we want to specify the facility and severity the use -p option:**
  + **logger -p local0.info “message”**
* **If we want to tag a message(u want to use scriptname as tag)the use -t option:**
  + **logger -t myscript -p local0.info “message”**
* **If we want to pair the processId in the log then use the -i option**
  + **logger -I -t myscript “message”**
* **-s option is used to display on screen**
  + **logger -s -p local0.info “message”**

**EXAMPLE:**

VERBOSE=false

HOST="google.com"

PID="$$"

PROGRAM\_NAME="$0"

THIS\_HOST=$(hostname)

logit () {

local LOG\_LEVEL=$1

shift **# to shift positional parameters to left**

MSG=$@ **#contains everything except the first positional parameter**

TIMESTAMP=$(date +"%Y-%m-%d %T")

if [ $LOG\_LEVEL = 'ERROR' ] || $VERBOSE

then

echo "${TIMESTAMP} ${THIS\_HOST} ${PROGRAM\_NAME}[${PID}]: ${LOG\_LEVEL} ${MSG}"

fi

}

logit INFO "Processing data."

**#calling function with log\_level info and msg**

fetch-data $HOST || logit ERROR "Could not fetch data from $HOST"

**#the command is executed then logit is called with log\_level error and the msg**

* **WHILE LOOPS:**
* INDEX=1

while [ $INDEX -lt 6 ]

do

echo "Creating project-${INDEX}"

mkdir /usr/local/project-${INDEX}

((INDEX++))

Done

* while [ "$CORRECT" != "y" ]

do

read -p "Enter your name: " NAME

read -p "Is ${NAME} correct? " CORRECT

done

* while ping -c 1 app1 >/dev/null

do

echo "app1 still up..."

sleep 5

done

echo "app1 down, continuing."

* LINE\_NUM=1

while read LINE

do

echo "${LINE\_NUM}: ${LINE}"

((LINE\_NUM++))

done < /etc/fstab

* grep xfs /etc/fstab | while read LINE

do

echo "xfs: ${LINE}"

done

* FS\_NUM=1

grep xfs /etc/fstab | while read FS MP REST

do

echo "${FS\_NUM}: file system: ${FS}"

echo "${FS\_NUM}: mount point: ${MP}"

((FS\_NUM++))

Done

* while true

do

read -p "1: Show disk usage. 2: Show uptime. " CHOICE

case "$CHOICE" in

1)

df -h

;;

2)

uptime

;;

\*)

break

;;

esac

done

* mysql -BNe 'show databases' | while read DB

do

db-backed-up-recently $DB

if [ "$?" -eq "0" ]

then

continue

fi

backup $DB

done

* **DEBUGGING**
  + **-x option prints the commands s they execute**
  + **To include debugging in script use= #!/bin/bash -x**
  + **To do in command line= set -x and to stop= set +x**
  + **To do it for a particular section in a script add set -x before the lines u want to debug and set +x where u want to stop**
  + **-e option used for exit on error, if any command exits with nonzero status this will stop the script there**
  + **We can use it along with -x option, order doesn’t matters**
  + **#!/bin/bash -xe, #!/bin/bash -ex, #!/bin/bash -x-e, #!/bin/bash -e-x**
  + **-v option prints shell input lines as they read**
  + **We can also use DEBUG keyword, DEBUG=true, DEBUG=false**

**EXAMPLES:**

* #!/bin/bash

TEST\_VAR="test"

set -x

echo $TEST\_VAR

set +x

hostname

* #!/bin/bash -e

FILE\_NAME="/not/here"

ls $FILE\_NAME

echo $FILE\_NAME

* #!/bin/bash -ex

FILE\_NAME="/not/here"

ls $FILE\_NAME

echo $FILE\_NAME

* #!/bin/bash -v

TEST\_VAR="test"

echo "$TEST\_VAR"

* #!/bin/bash -vx

TEST\_VAR="test"

echo "$TEST\_VAR"

* DEBUG=true

if $DEBUG

then

echo "Debug mode ON."

else

echo "Debug mode OFF."

fi

START SERVICE

* Sudo systemctl start apache2
* Sudo service apache2 start

STOP SERVICE

* Sudo systemctl stop apache2
* Sudo service apache2 stop