The System Administrator's Guide to Bash Scripting

https://learn.acloud.guru/course/bccc6769-38e7-4a7f-8255-6914b7244caf/dashboard

Table of Contents

All scripts from course:

/Users/billy/Documents/IT-Learning/2024-A-Cloud-Guru/The System Administrator's Guide to Bash Scripting/all-scripts.txt

CH 1 (skip) Syllabus and History of Bash

CH 2 Core Concepts

Bash Files

.bash_profile

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.235.14.51	172.31.36.98

add alias to .bash_profile:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

alias lZ='ls -laZ | more'

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
```

.bashrc

add aliases to .bashrc:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# Added aliases (by me)

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Uncomment the following line if you don't like systemctl's auto-paging
feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

.bash_history

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	3.81.131.142	172.31.36.98

```
keep command out of bash history

export HISTCONTROL=$HISTCONTROL:ignorespace
```

```
once you exit that session, the HISTCONTROL gets set back to waht it
was before
```

.bash_logout

can include scripts or commands to execute upon logout. Like restoring .bashrc.original, for example (if you have it set up that way).

What makes a file a shell script?

add $HOME/bin directory to path in .bash_profile, and that will make scripts in that directory directly callable by name

```
[cloud_user@master ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
```

Displaying Environment Variables in a Script

```
#!/bin/bash

clear

echo "Give env info"
echo "+++++++++++++++++++++++++++++++"
echo
echo "Hello user: $USER"
echo
echo "Your home dir is: $HOME"
echo
echo "Your History file file will ignore: $HISTCONTROL"
echo
echo "Your Terminal Session type is: $TERM"
echo
```

Using Variables on the Command Line

use a command in the export of a variable with back-ticks. For example:

```
[cloud_user@master ~]$ export TODAY=`date`
[cloud_user@master ~]$ echo $TODAY
Sun Dec 31 16:23:35 UTC 2023

But this must be re-exported in order to update the value of the variable
```

Alternatively, do it like this:

```
Notice how the command must be in (), otherwise it wont work:
     [cloud_user@master ~]$ export TODAY=$date
     [cloud_user@master ~]$ echo $TODAY

[cloud_user@master ~]$ export TODAY=$(date)
[cloud_user@master ~]$ echo $TODAY
Sun Dec 31 16:24:48 UTC 2023
```

Setting and Using Variables in Scripts

The variables set and called in scripts are not global, meaning, they are only set and used during the execution of the script. This is because the script starts a new bash session (ie #!/bin/bash):

```
[cloud_user@master bin]$ cat varexample.sh
#!/bin/bash

MYUSERNAME="username"
MYPASSWORD="password123"
STARTOFSCRIPT=`date`

echo -e "\nMy login name for this application is:\t$MYUSERNAME"
echo -e "My login password is:\t\t\t$MYPASSWORD"
echo -e "I executed this script at:\t\t$STARTOFSCRIPT"

sleep 5

ENDOFSCRIPT=`date`

echo -e "This script completed at:\t\t$ENDOFSCRIPT\n"

[cloud_user@master bin]$ varexample.sh

My login name for this application is:username
My login password is:        password123
```

```
I executed this script at:        Sun Dec 31 16:40:21 UTC 2023
This script completed at:     Sun Dec 31 16:40:26 UTC 2023

[cloud_user@master bin]$ echo $MYUSERNAME

[cloud_user@master bin]$
```

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.172.29.93	172.31.36.98

Comment Types and Structures

```
[cloud_user@master bin]$ cat comments.sh
#!/bin/bash

#this line is intended as a general description of the script

# Comment up your blocks of script

clear   #you can even add comments in line

#MYUSERNAME="Alex"   #the username for this application

MYUSERNAME="Billy"   #username added later

echo "We just added a username, $MYUSERNAME."

DATETIMESTAMP=`date`

echo "This is when the script was run: $DATETIMESTAMP" #this is the
timestamp of run
```

Command Substitution

```
[cloud_user@master bin]$ cat substitution.sh
#!/bin/bash

#this script is intended to show how to do simple substitution
```

```
clear

#do it with variables
#Notice the commands when setting the variables have to be inside of $()

TODAYSDATE=$(date)
USERFILES=$(find /home -user $USER)

echo "Today's date: $TODAYSDATE"
echo "All files owned by $USER: $USERFILES"


#do it with aliases
#notice the alias (in the final echos) have to be in $(), because an alias
is a command

shopt -s expand_aliases

alias TODAY="date"
alias UFILES="find /home -user $USER"

echo -e "\n\n##########\nWith Alias, TODAY is: $(TODAY)"
echo -e "\nWith Alias, UFILES is:\n$(UFILES)\n"
```

Exit Status

https://itsfoss.com/linux-exit-codes/

Exit code	Meaning of the code
0	Command executed with no errors
1	Code for generic errors
2	Incorrect command (or argument) usage
126	Permission denied (or) unable to execute
127	Command not found, or PATH error
128+n	Command terminated externally by passing signals, or it encountered a fatal error
130	Termination by Ctrl+C or SIGINT (*keyboard interrupt*) 128+2
137	Termination by SIGKILL 128+9
143	Termination by SIGTERM (*default termination*) 128+15
255/*	Exit code exceeded the range 0-255, hence wrapped up

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	54.197.212.202	172.31.36.98

```
use echo $? to get the most recent command's exit code
exit 0 is a success (no error)

[cloud_user@master bin]$ ls
comments.sh  env.sh  substitution.sh  test.sh  varexample.sh
[cloud_user@master bin]$ echo $?
0
[cloud_user@master bin]$ rm noodles
rm: cannot remove ‘noodles’: No such file or directory
[cloud_user@master bin]$ echo $?
1


###############
File /home/cloud_user/bin/errors-exit.sh

#!/bin/bash

#this is to show exit types

clear

#the moment there is an error, exit the shell (aka end the script)
set -e

echo -e "\nDo some math"
expr 1 + 5
echo "Status code: $?"

echo -e "\ncreate a file 'noodles.sh'"
touch noodles.sh
echo "Status code: $?"

echo -e "\nremove the file 'noodles.sh' twice, should fail on second rm"
rm noodles.sh
echo -e "'noodles.sh' has been removed, now try it the second time.\nThis
is where the script will error out and quit.\n"
rm noodles.sh
echo -e "Status code: $?\n"

echo -e "\nAnother successfull command (this echo, actually)"
echo -e "Status code: $?"

echo


###############
File /home/cloud_user/bin/errors-keep-moving.sh
```

```
#!/bin/bash

#this is to show exit types

clear

echo -e "\nDo some math"
expr 1 + 5
echo "Status code: $?"

echo -e "\ncreate a file 'noodles.sh'"
touch noodles.sh
echo "Status code: $?"

echo -e "\nremove the file 'noodles.sh' twice, should fail on second rm"
rm noodles.sh
echo "'noodles.sh' has been removed, now try it the second time"
rm noodles.sh
echo -e "Status code: $?\nThe script still continues, even though this
command errored out"

echo -e "\nAnother successfull command (this echo, actually)"
echo -e "Status code: $?"

echo
```

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	34.204.90.27	172.31.36.98

Arithmetic Operations (expr, perl -e 'print ...', shell arithmetic)

Bash shell arithmetic Expansion (compound notation)

https://phoenixnap.com/kb/bash-math

`$((expression))`
The syntax consists of:

- Compound notation **`(())`** which evaluates the expression.
- The variable operator **`$`** to store the result.

```
[cloud_user@master bin]$ echo $((2+3))
5
[cloud_user@master bin]$ FIVE=$((2+3)); echo ${FIVE}
5
```

FACTORIALS SCRIPT:

perl and expr

```
* Numeric expressions::         + - * / %
* Relations for expr::          | & < <= = == != >= >

info coreutils 'expr invocation'
```

man expr

```
       ARG1 | ARG2: ARG1 if it is neither null nor 0, otherwise ARG2
       ARG1 & ARG2: ARG1 if neither argument is null or 0, otherwise 0
       ARG1 < ARG2: ARG1 is less than ARG2
       ARG1 <= ARG2: ARG1 is less than or equal to ARG2
       ARG1 = ARG2: ARG1 is equal to ARG2
       ARG1 != ARG2: ARG1 is unequal to ARG2
       ARG1 >= ARG2: ARG1 is greater than or equal to ARG2
       ARG1 > ARG2: ARG1 is greater than ARG2
       ARG1 + ARG2: arithmetic sum of ARG1 and ARG2
       ARG1 - ARG2: arithmetic difference of ARG1 and ARG2
       ARG1 * ARG2: arithmetic product of ARG1 and ARG2
       ARG1 / ARG2: arithmetic quotient of ARG1 divided by ARG2
       ARG1 % ARG2: arithmetic remainder of ARG1 divided by ARG2

[cloud_user@master bin]$ cat arithmetic.sh
#!/bin/bash

#more versitile, not just integers
perl -e 'print 23 / 4'; echo

#expr only does integer math, and only gives whole number output.
#expr will do modular math though, to get the remainder. Just no decimals
echo -e "$(expr 23 / 4) remainder $(expr 23 % 4)"

#regroup for different order of operations.
#Some characters have to be escaped
expr 2 + 2 \* 4
expr \( 2 + 2 \) \* 4
```

Global and Local Environment Variables

printenv -

env -

set -

Special Characters - Quotes and Escapes

```
[cloud_user@master bin]$ echo $COL

[cloud_user@master bin]$ echo \$COL
$COL
[cloud_user@master bin]$ C=hello
[cloud_user@master bin]$ echo $C
hello
[cloud_user@master bin]$ echo $COL

[cloud_user@master bin]$ echo "$COL"

[cloud_user@master bin]$ echo ${C}OL
helloOL
[cloud_user@master bin]$ echo \${C}OL
${C}OL
[cloud_user@master bin]$ echo $C
hello
[cloud_user@master bin]$ echo \$C
$C
[cloud_user@master bin]$ echo '$C'
$C
[cloud_user@master bin]$ echo "$C"
hello

[cloud_user@master bin]$ echo date
date
[cloud_user@master bin]$ echo `date`
Tue Jan 2 00:02:39 UTC 2024
[cloud_user@master bin]$ echo "`date`"
Tue Jan  2 00:02:46 UTC 2024
[cloud_user@master bin]$ echo "\`date\`"
`date`
[cloud_user@master bin]$ echo "${date}"

[cloud_user@master bin]$ echo "$(date)"
Tue Jan  2 00:03:18 UTC 2024
```

Using /dev/null

```
[cloud_user@master bin]$ cat null.sh
#!/bin/bash

#redirect to /dev/null example

echo "This goes to the console"

echo "This goes to /dev/null" >> /dev/null
```

```
[cloud_user@master bin]$ null.sh
This goes to the console
[cloud_user@master bin]$ null.sh >> /dev/null
```

The Read Statement

https://phoenixnap.com/kb/bash-read

```
[cloud_user@master bin]$ cat readsample.sh
#!/bin/bash

#interactive script for user input

echo -e "\nYour responses will not show as you type them.\n"

read -p "Enter your first name: " -s FIRST
echo
read -p "Enter your last name: " -s LAST
echo
read -p "Enter your age: " -s AGE

echo -e "\n\nHello, ${FIRST} ${LAST}.\nIn 10 years, you will be $(expr $
{AGE} + 10 ).\n"

unset FIRST
unset LAST
unset AGE
```

Shell Expansion

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	18.212.134.85	172.31.36.98

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	3.90.178.168	172.31.36.98

```
[cloud_user@master ~]$ echo sh{op,oot,ort,out}
shop shoot short shout
[cloud_user@master ~]$ echo sh{op,oot,ort,out}{it,at,asta}
shopit shopat shopasta shootit shootat shootasta shortit shortat shortasta
shoutit shoutat shoutasta
```

```
[cloud_user@master ~]$ pwd
/home/cloud_user
[cloud_user@master ~]$ cd bin/
[cloud_user@master bin]$ pwd
/home/cloud_user/bin
[cloud_user@master bin]$ echo ~+
/home/cloud_user/bin
[cloud_user@master bin]$ echo ~-
/home/cloud_user
```

Types of Variables (implicit versus explicit)

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.158.51.191	172.31.36.98

```
[cloud_user@master ~]$ MYVAR=4
[cloud_user@master ~]$ echo $(expr $MYVAR + 5)
9
[cloud_user@master ~]$ declare -p MYVAR
declare -- MYVAR="4"
[cloud_user@master ~]$ echo $(expr $MYVAR + s)
expr: non-integer argument

[cloud_user@master ~]$ echo "$(expr $MYVAR + 5)"
9
[cloud_user@master ~]$ declare -i NEWVAR=s
[cloud_user@master ~]$ declare -p NEWVAR
declare -i NEWVAR="0"
[cloud_user@master ~]$ declare -i NEWVAR=5
[cloud_user@master ~]$ declare -p NEWVAR
declare -i NEWVAR="5"
[cloud_user@master ~]$ MYVAR="New Value"
[cloud_user@master ~]$ declare -p MYVAR
declare -- MYVAR="New Value"
```

Arrays

zero based - start counting at zero

space delimited

```
[cloud_user@master ~]$ MYARRAY=("First" "second" "third")
[cloud_user@master ~]$ for i in {1..3}; do echo ${MYARRAY[${i}]}; done
second
third
```

```
[cloud_user@master ~]$ for i in {0..2}; do echo ${MYARRAY[${i}]}; done
First
second
third

[cloud_user@master ~]$ echo ${MYARRAY[*]}
First second third
[cloud_user@master ~]$ MYARRAY[3]="fourth"
[cloud_user@master ~]$ echo ${MYARRAY[*]}
First second third fourth

[cloud_user@master ~]$ NEWARRAY=("1st first" "2nd second" "3rd third")
[cloud_user@master ~]$ echo ${NEWARRAY[1]}
2nd second
[cloud_user@master ~]$ echo ${NEWARRAY[*]}
1st first 2nd second 3rd third
```

```
[cloud_user@master bin]$ cat array-example.sh
#!/bin/bash

#simple array

SERVERLIST=("srvr1" "srvr2" "srvr3" "srvr4")

COUNT=0

for INDEX in ${SERVERLIST[@]}
do
    echo "Processing Server: ${SERVERLIST[${COUNT}]}"
    #COUNT="$(expr ${COUNT} + 1)"
    #COUNT="$((${COUNT}+1))"
    COUNT=$((${COUNT}+1))
done
```

CH 3 Conditional Statements

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.227.69.155	172.31.36.98

Passing variables to scripts at CLI

```
[cloud_user@master bin]$ cat cmdline-var.sh
#!/bin/bash

#demo of command line variables into a shell script

echo -e "The following item was passed into the script  at runtime:\n\n$1\
n"

[cloud_user@master bin]$ cmdline-var.sh helloScript
The following item was passed into the script  at runtime:

helloScript

[cloud_user@master bin]$ cmdline-var.sh $(expr 2 + 2)
The following item was passed into the script  at runtime:

4

=======

[cloud_user@master bin]$ cat cmdline-var.sh
#!/bin/bash

#demo of command line variables into a shell script

echo -e "The following item was passed into the script  at runtime:\n\n$1\
n$2\n$3\n"
[cloud_user@master bin]$ cmdline-var.sh how are you
The following item was passed into the script  at runtime:

how
are
you

=============
[cloud_user@master bin]$ cat cmdline-var.sh
#!/bin/bash

#demo of command line variables into a shell script

USER=$1
PASS=$2
NOTE=$3

echo -e "\nThe following item was passed into the script  at runtime:\n"

echo -e "Username entered:\t$USER"
echo -e "Password entered:\t$PASS"
echo -e "Note entered:\t\t$NOTE\n"

[cloud_user@master bin]$ cmdline-var.sh MyUser MyPass MyNote

The following item was passed into the script  at runtime:

Username entered:    MyUser
Password entered:    MyPass
Note entered:        MyNote
```

The if statement

```
[cloud_user@master bin]$ cat simple-if.sh
#!/bin/bash

#simple if for guessing a number

BREAK="#########################"

echo -e "\nGuess the secret number\n${BREAK}\n"

echo -e "Enter a number between 1 and 5: "

read GUESS

if [ ${GUESS} -eq 3 ]
then
    echo -e "\nYou guessed the correct value: ${GUESS}\n\n${BREAK}\n"
fi

if [ ${GUESS} -ne 3 ]
then
    echo -e "\n${GUESS} is not correct.\n\n${BREAK}\n"
fi
```

```
[cloud_user@master bin]$ simple-if.sh

Guess the secret number
#########################

Enter a number between 1 and 5:
2

2 is not correct.

#########################

[cloud_user@master bin]$ simple-if.sh

Guess the secret number
#########################

Enter a number between 1 and 5:
3

You guessed the correct value: 3

#########################

==================

[cloud_user@master bin]$ cat test-if-file.sh
#!/bin/bash

#test if a file exists

BREAK="#######################"
FILENAME=$1

echo -e "\nTesting for the existence of a file named ${FILENAME}\n${BREAK}\
n"

if [ -a ${FILENAME} ]
then
    echo -e "${FILENAME} does exist. See details below:\n$(ls -la $
{FILENAME})\n"
fi

[cloud_user@master bin]$ test-if-file.sh man-expr.txt

Testing for the existence of a file named man-expr.txt
#######################

man-expr.txt does exist. See details below:
-rw-rw-r--. 1 cloud_user cloud_user 3419 Jan  1 13:54 man-expr.txt
```

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	3.93.16.166	172.31.36.98

```
[cloud_user@master bin]$ cat if-expression.sh
#!/bin/bash

#test multiple expressions in an if statement

clear

FILENAME=$1
BREAK="################################"

echo -e "\ntesting for existence and readbility of ${FILENAME}.\n\n$
{BREAK}\n"

if [ -f ${FILENAME} ] && [ -r ${FILENAME} ]
then
    echo -e "\nFile ${FILENAME} exists and is readable.\n\n${BREAK}\n\n"
fi

if [ ! -f ${FILENAME} ]
then
    echo -e "${FILENAME} does NOT exist and/or is not readable.\nHere are
the permissions for files that do exist in $(pwd):\n\n$(ls -ltr)\n"
fi
[cloud_user@master bin]$ if-expression.sh tehgfst.sh


testing for existence and readbility of tehgfst.sh.

################################

tehgfst.sh does NOT exist and/or is not readable.
Here are the permissions for files that do exist in /home/cloud_user/bin:

total 84
-rwxrwxr-x. 1 cloud_user cloud_user   84 Dec 31 14:02 test.sh
-rwxrwxr-x. 1 cloud_user cloud_user  261 Dec 31 16:10 env.sh
-rwxrw-r--. 1 cloud_user cloud_user  334 Dec 31 16:36 varexample.sh
-rwxrw-r--. 1 cloud_user cloud_user  405 Dec 31 19:32 comments.sh
-rwxrw-r--. 1 cloud_user cloud_user  613 Dec 31 22:44 substitution.sh
-rwxrw-r--. 1 cloud_user cloud_user  542 Jan  1 13:39 errors-keep-moving.sh
-rwxrw-r--. 1 cloud_user cloud_user  610 Jan  1 13:45 errors-exit.sh
-rw-rw-r--. 1 cloud_user cloud_user 3419 Jan  1 13:54 man-expr.txt
-rw-rw-r--. 1 cloud_user cloud_user 7626 Jan  1 14:01 expr.manual
```

```
-rwxrw-r--. 1 cloud_user cloud_user  388 Jan  1 22:53 arithmetic.sh
-rwxrw-r--. 1 cloud_user cloud_user  121 Jan  2 00:21 null.sh
-rwx------. 1 cloud_user cloud_user  363 Jan  2 00:58 readsample.sh
-rwxrw-r--. 1 cloud_user cloud_user  253 Jan  5 12:50 array-example.sh
-rw-rw-r--. 1 cloud_user cloud_user 6422 Jan  5 13:56 all-scripts.txt
-rwxrw-r--. 1 cloud_user cloud_user  571 Jan  5 13:56 cat-all-dot-sh.sh
-rwxrw-r--. 1 cloud_user cloud_user  270 Jan  6 20:37 cmdline-var.sh
-rwxrw-r--. 1 cloud_user cloud_user  367 Jan  6 20:52 simple-if.sh
-rwxrw-r--. 1 cloud_user cloud_user  398 Jan  6 21:09 test-if-file.sh
-rwxrw-r--. 1 cloud_user cloud_user  496 Jan  7 00:20 if-expression.sh
```

if/then/else

```
[cloud_user@master bin]$ cat if-then-else.sh
#!/bin/bash

#a script to illustrate if/then/else and nested if statements

BREAK="###################"

echo -e "Enter a number from 1 to 10: "
read VALUE

if [ "${VALUE}" -gt "0" ] 2>/dev/null && [ ${VALUE} -lt "11" ] 2>/dev/null
then
    echo -e "You entered ${VALUE}, which is from 1 to 10."

else
    echo -e "You entered ${VALUE}, which is not from 1 to 10."

fi


[cloud_user@master bin]$ if-then-else.sh
Enter a number from 1 to 10:
dsa
You entered dsa, which is not from 1 to 10.
[cloud_user@master bin]$ if-then-else.sh
Enter a number from 1 to 10:
3
You entered 3, which is from 1 to 10.
[cloud_user@master bin]$ if-then-else.sh
Enter a number from 1 to 10:
13
You entered 13, which is not from 1 to 10.
```

if/elif/else

```
[cloud_user@master bin]$ cat if-elif-else.sh
#!/bin/bash

#a script to illustrate if/elif/else and nested if statements

BREAK="###################"

echo -e "Enter a number from 1 to 10: "
read VALUE

if [ "${VALUE}" -eq "3" ] 2>/dev/null
then
    echo -e "You entered ${VALUE}, which is from 1 to 10, and also happens
to be my favorite number."

elif [ "${VALUE}" -gt "0" ] 2>/dev/null && [ ${VALUE} -lt "11" ]
2>/dev/null
then
    echo -e "You entered ${VALUE}, which is from 1 to 10."

elif [ "${VALUE}" -eq "13" ] 2>/dev/null
then
    echo -e "You entered ${VALUE}, which is not from 1 to 10, and is also
an unlucky number."

else
    echo -e "You entered ${VALUE}, which is not from 1 to 10."

fi
[cloud_user@master bin]$ if-elif-else.sh
Enter a number from 1 to 10:
3
You entered 3, which is from 1 to 10, and also happens to be my favorite
number.
[cloud_user@master bin]$ if-elif-else.sh
Enter a number from 1 to 10:
5
You entered 5, which is from 1 to 10.
[cloud_user@master bin]$ if-elif-else.sh
Enter a number from 1 to 10:
11
You entered 11, which is not from 1 to 10.
[cloud_user@master bin]$ if-elif-else.sh
Enter a number from 1 to 10:
asd
```

```
You entered asd, which is not from 1 to 10.
[cloud_user@master bin]$
```

for loops

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.84.249.143	172.31.36.98

```
[cloud_user@master bin]$ cat cat-all-dot-sh.sh
#!/bin/bash

clear

OUTFILE=all-scripts.txt
DIV="########################################"

rm -f ${OUTFILE}

echo -e "\nContent from https://learn.acloud.guru\nThe System
Administrator's Guide to Bash Scripting\n$(date)\n\nList of .sh files in $
(pwd)/:\n\n$(ls -ltr | awk '{ print $6 " " $7 " " $9}' | grep ".sh")\n" >>
${OUTFILE}

for FILE in $(ls -tr | grep ".sh")
do
    echo -e "\n${DIV}\nFile $(pwd)/${FILE}\n$(ls -ltr ${FILE} | awk
'{ print $6 " " $7 }')\n${DIV}\n" >> ${OUTFILE}
    cat ${FILE} >> ${OUTFILE}
    echo -e "\n\n" >> ${OUTFILE}
done

echo -e "\n\n"

clear

echo -e "\n${DIV}\n\nReview scripts by running:\n\n\t'cat $(pwd)/$
{OUTFILE}'\n\n${DIV}\n"
```

case statement

build a menu with options

```
[cloud_user@master bin]$ cat case-sample.sh
#!/bin/bash

#demo of the case statement

clear

BREAK="#########################\n"

echo -e "MAIN MENU:\n${BREAK}"

echo -e "1) Choice 1\n2) Choice 2\n3) Choice 3\n\nEnter Choice: "

read MENUCHOICE

echo -e "\n-----\nYou chose ${MENUCHOICE}\n"

case ${MENUCHOICE} in

    1) echo -e "Congrats for chosing option ${MENUCHOICE}.\n"
    ;;

    2) echo -e "Option ${MENUCHOICE} is a great one!\n"
    ;;

    3) echo -e "I would have gone with option ${MENUCHOICE} as well.\n"
    ;;

    *) echo -e "${MENUCHOICE} is not an option. Choose from the above
options.\n"
    ;;

esac
```

######################

```
MAIN MENU:
#########################

1) Choice 1
2) Choice 2
3) Choice 3
```

```
Enter Choice:
2

-----
You chose 2

Option 2 is a great one!
```

while loops

increment and decriment

https://linuxize.com/post/bash-increment-decrement-variable/

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	3.82.7.121	172.31.36.98

```
[cloud_user@master bin]$ cat while-sample.sh
#!/bin/bash

#while loop example

clear

echo -e "How many times do you want to show "Hello World"?"

read MANY

#while [ ${MANY} -gt 0 ]
#do
#   echo -e "Hello World - ${MANY}"
#   MANY=$(( ${MANY} - 1 ))
#done

COUNT=1

while [ ${COUNT} -le ${MANY} ]
do
    echo -e "Hello World\t${COUNT}"
    #COUNT=$((${COUNT}+1))
    #let COUNT+=1
    #let "COUNT++"
    let "++COUNT"
done
```

execution operators (&& and ||)

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	54.198.119.194	172.31.36.98

|| for "or"

&& for "and"

```
[cloud_user@master bin]$ cat execops.sh
#!/bin/bash

#execution operators example

clear

read -p "Enter a number from 1 to 5: " VALUE

echo -e "You entered ${VALUE}."

if [ ${VALUE} -lt "1" ] || [ ${VALUE} -gt "5" ]
    echo -e "${VALUE} is out of range."

elif [ ${VALUE} -eq "1" ] || [ ${VALUE} -eq "3" ] || [ ${VALUE} -eq "5" ]
then
    echo -e "${VALUE} is an odd number."

else
    echo -e "${VALUE} is an even number."

fi
```

CH 4 Input and Output

Reading Files

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.204.138.199	172.31.36.98

```
[cloud_user@master bin]$ cat readfile.sh
#!/bin/bash

#simple script to read a file

read -p "Enter a filename to read: " FILE

while read -r SUPERHERO
do
    echo -e "Superhero Name: ${SUPERHERO}"
done < "$FILE"
```

file descriptors and handles

formally open and close a file

Using 5 as an arbitrary number and file.txt as an arbitrary filename

open file descriptor: exec 5 <>file.txt

read only: exec 5<

write only: exec 5>

read and write: 5<>

close a file descriptor: exec 5>&-

```
[cloud_user@master bin]$ cat filedescriptor.sh
#!/bin/bash

#demo of reading and writing to a file using a file descriptor

read -p "Enter a filename to read: " FILE

exec 5<>${FILE}
```

```
while read SUPERHERO
do
    echo "Superhero name: ${SUPERHERO}"
done <&5

echo "This file was read on: $(date) by $(echo ${USER})." >&5

exec 5>&-
```

IFS and delimiting

IFS is "Internal Filed Separator"

IFS is, by default, a "space" character

script from lesson online does not work.

traps and signals

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.198.116.202	172.31.36.98

```
[cloud_user@master bin]$ cat trap-example.sh
#!/bin/bash

#example of trapping events and limiting the shell stopping

clear

trap 'echo " - Please press Q to exit."' SIGINT SIGTERM SIGTSTP

while [ "${CHOICE}" != "Q" ] && [ "${CHOICE}" != "q" ]
do
    echo -e "\nMAIN MENU\n==============\n"
    echo "1) Choice ONE"
    echo "2) Choice TWO"
    echo "3) Choice THREE"
    echo -e "\n----------------\nQ/q:\tEXIT\n----------------\n"
    read -p "Choose from the options above: " CHOICE
    clear

done
```

CH5 Debugging and Error Handling

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.198.153.37	172.31.36.98

debugging your script

bash -x scriptname.sh

```
=======================================

[cloud_user@master bin]$ cat env.sh
#!/bin/bash

clear

echo "Give env info"
echo "+++++++++++++++++++++++++++++++"
echo
echo "Hello user: $USER"
echo
echo "Your home dir is: $HOME"
echo
echo "Your History file file will ignore: $HISTCONTROL"
echo
echo "Your Terminal Session type is: $TERM"
echo

=========================================

+ echo 'Give env info'
Give env info
+ echo +++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
+ echo

+ echo 'Hello user: cloud_user'
Hello user: cloud_user
+ echo

+ echo 'Your home dir is: /home/cloud_user'
Your home dir is: /home/cloud_user
+ echo
+ echo 'Your History file file will ignore: ignoredups'
Your History file file will ignore: ignoredups
+ echo

+ echo 'Your Terminal Session type is: xterm-256color'
Your Terminal Session type is: xterm-256color
+ echo
```

set command

```
[cloud_user@master bin]$ cat debug-readfile.sh
#!/bin/bash

#simple script to read a file

read -p "Enter a filename to read: " FILE

#start debug

set -x

while read -r SUPERHERO
do
    echo -e "Superhero Name: ${SUPERHERO}"
done < "$FILE"

#stop debug

set +x
```

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.174.143.99	172.31.36.98

Error Handling

```
[cloud_user@master bin]$ cat error-exit.sh
#!/bin/bash

#demo of using error handling with exit

echo "Change to a dir and list contents."

DIR=$1

cd ${DIR} 2>/dev/null
```

```
if [ "$?" = "0" ]
then

    echo "Changed into directory: $(pwd)"
    ls

else

    echo "Cannot cd to ${DIR}, please try again."
    exit 1

fi
```

CH 6 Functions

https://linuxize.com/post/bash-functions/

https://phoenixnap.com/kb/bash-function/

Simple Functions

```
[cloud_user@master bin]$ cat simple-function.sh
#!/bin/bash

#example of a simple bash function

#before function
echo "This is before the function"

#define the function:
funcExample () {

echo "This is my function, named funcExample"

}

#call the function:
funcExample

#after function
echo "The function was called, and now we are at the end of the script."
```

Structure of Functions

```
[cloud_user@master bin]$ cat function-structure.sh
#!/bin/bash

#demo of functions within a shell script structure

###################
#variables section:
###################

CMDLINE=$1
COMMAND=$0

##############################
#function definitions section:
##############################
```

```
#This function displays a message:
funcExample1 () {

echo -e "This is an example function, named funcExample1\n"

}

#this function displays another message:
funcExample2 () {

echo -e "This is another example function, named funcExample2"
echo -e "The command that was just now executed was:\n${COMMAND} $
{CMDLINE}\n"

}


#############################
#beginning of script section
#############################

clear

echo -e "\nThis is where the script section actually begins\n"

#call the functions:
echo -e "\nHere are my functions being called:\n"

funcExample1

funcExample2

echo -e "This is after the functions were called\n"
```

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	3.84.102.55	172.31.36.98

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	54.89.242.173	172.31.36.98

Variable Scope in Fnctions

```
[cloud_user@master bin]$ cat var-scope.sh
#!/bin/bash

#demonstrating variable scope

#global var declaration


####################
# GLOBAL VARIABLES #
# SECTION          #
####################

GLOBALVAR="***Globally Visible***"


#####################
# FUNCTIONS SECTION #
#####################

functExample () {

    ####################
    # LOCAL FUNCT VARS #
    ####################

    LOCALVAR="***Locally visible***"
    echo "From within the function, the variable is ${LOCALVAR}"

}


##################
# SCRIPT SECTION #
##################

clear

echo -e "This step happens first\n\n"
echo -e "GLOBALVAR variable = ${GLOBALVAR} - before function call\n\n"
echo -e "LOCALVAR variable = ${LOCALVAR} - before function call\n\n"

echo -e "Calling function:\nfunctExample()\n\n"

functExample

echo -e "Function has been called:\n\n"
echo -e "GLOBALVAR variable = ${GLOBALVAR} - after function call\n\n"
echo -e "LOCALVAR variable = ${LOCALVAR} - after function call\n\n"
```

Functions with Parameters

```
[cloud_user@master bin]$ cat func-parameters.sh
#!/bin/bash

#example of parameters in from the command line
#aka "functional parameter passing"

# VARIABLE SECTION #
#####################

USERNAME=$1


# FUNCTION SECTION #
####################

#calculate days in age:
funcAgeInDays () {

#Global variable USERNAME was what was typed into the teminal when running
the command
#function variable (or parameter?) USERAGE was set when script was run, and
it was typed into the prompt

    echo "Hello $USERNAME, you are $1 Years old."
    echo "That makes you approximately $(( $1 * 365 )) days old."

}


# SCRIPT SECTION #
##################

clear

#ask for and store age in years
read -p "Enter your age: " USERAGE

#calculate approx age in days
funcAgeInDays $USERAGE
```

Nested Functions

```
[cloud_user@master bin]$ cat func-nested.sh
#!/bin/bash

#demonstrate the use of nested functions


# VARIBALE SECTION #
####################

#global variable
GENDER=$1


# FUNCTION SECTION #
####################

#create a human being
funcHuman() {
    ARMS=2
    LEGS=2

    funcMale() {
        BEARD=1

        echo -e "This $GENDER has $ARMS arms, $LEGS legs and $BEARD
beard.\n\n"

    }

    funcFemale() {
        BEARD=0

        echo -e "This $GENDER has $ARMS arms, $LEGS legs, and $BEARD
beard.\n\n"

    }

}


# SCRIPT SECTION #
##################

clear

echo -e "Determining the characteristics of the gender $GENDER.\n\n"

if [ "$GENDER" == "male" ]
```

```
then
    funcHuman
    funcMale

elif [ "$GENDER" == "female" ]
then
    funcHuman
    funcFemale

else
    echo -e "Must select either male or female.\n\n"

fi
```

Function Return and Exit

hostname	**username**	**pass**	**pub IP**	**priv IP**
master	cloud_user	1 m	35.173.188.225	172.31.36.98

```
[cloud_user@master bin]$ cat return-val.sh
#!/bin/bash

#demo of return values and testing results

####################
# VARIABLE SECTION #
####################

NO=1
YES=0
FIRST=$1
SECOND=$2
THIRD=$3
#FOURTH=$4



####################
# FUNCTION SECTION #
####################

#check command line parameters passed in

funcParams () {
    #did we get three parameters?
    # ! -z means if it is not a null value
    if [ ! -z "$THIRD" ]
    then
        echo "We got at least three parameters."
        return $YES
    else
        echo "We got less than three parameters."
        return $NO
    fi

}

#did we get exactly three or not?

funcThreeOrNah () {
    if [ "$RETURN_VALS" -eq "$YES" ]
    then
        echo -e "We received three parameters, and they are:\nParam1=\
t$FIRST\nParam2=\t$SECOND\nParam3=\t$THIRD"
    else
        echo -e "The script expects three parameters. See below...\n\n\
tUSAGE:\treturnval.sh [param1] [param2] [param3]\n"
        exit 1
    fi

}


##################
# SCRIPT SECTION #
##################

funcParams

RETURN_VALS=$?

funcThreeOrNah


echo "The return value was $RETURN_VALS."
```

Factorials Function Example:

```
[cloud_user@master bin]$ cat factorials-script.sh
#!/bin/bash

# example usage:
# factorial 5 will perform 5!
# 5 * 4 * 3 * 2 * 1 = 120

factorial () {
    if (($1 > 1))
    then
        echo $(( $( factorial $(($1 - 1)) ) * $1 ))
    else

        echo 1
        return
    fi
}

factorial
```

CH 7 Samples and Use Cases

using the info box

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	3.85.144.249	172.31.36.98

ncurses

https://linuxconfig.org/how-to-use-ncurses-widgets-in-shell-scripts-on-linux

sudo yum install dialog

```
[cloud_user@master bin]$ cat simple-info-box.sh
#!/bin/bash

#script for simple info box with dialog and ncurses
# sudo yum install dialog

# VARIABLE SECTION #
####################

INFOBOX=${INFOBOX=dialog}
TITLE="Default"
MESSAGE="Something to say"
XCOORD=10
YCOORD=20


# FUNCTION SECTION #
####################

#display the info box and our message
#1 is the title, 2 is the message, 3 is x-coord, and 4 is y-coord
funcDisplayInfoBox () {
    $INFOBOX --title "$1" --infobox "$2" "$3" "$4"
    sleep "$5"
}


# SCRIPT SECTION #
##################
```

```
if [ "$1" == "shutdown" ]
then
    funcDisplayInfoBox "!*!*!WARNING!*!*!" "We are SHUTTING DOWN the
system!" "11" "21" "5"
    echo "Shutting Down..."
else
    funcDisplayInfoBox "Information..." "You are not doing anything fun..."
"11" "21" "5"
    echo "Not doing anything"
fi

#script complete
echo "Script complete"
```

Display a message box for confirmation

```
[cloud_user@master bin]$ cat simple-message-box.sh
#!/bin/bash

###
# VARIABLES #
#############

MSGBOX=${MSGBOX=dialog}
TITLE="Default"
MESSAGE="Some message"
XCOORD=10
YCOORD=20


###
# FUNCTIONS #
#############

#display message box with our message
funcDisplayMsgBox () {
    $MSGBOX --title "$1" --msgbox "$2" "$3" "$4"
}


###
# SCRIPT #
##########

if [ "$1" == "shutdown" ]
then
    funcDisplayMsgBox "WARNING SHUTTING DOWN" "Please select OK when you're
ready to shutdown."
    echo "SHUTTING DOWN NOW"
```

```
else
    funcDisplayMsgBox "Boring..." "You arent doing anything fun..." "10"
"20"
    echo "Not doing anything. Back to normal..."
fi
```

Advanced UI - Building a Menu System

hostname	username	pass	pub IP	priv IP
master	cloud_user	1 m	54.81.90.233	172.31.36.98

menu system case statement choose options

```
#########################################
#########################################
File /home/cloud_user/bin/simple-dialog.sh
Jan 20
#########################################
#########################################

#1/bin/bash

#demo of a dialog box that will display a menu


###
# VARIABLES SECTION #
#####################

MENUBOX=${MENUBOX=dialog}


###
# FUNCTIONS SECTION #
#####################

#function to display simple menu
#numbers at the end are height, width, number of choices available

funcDisplayDialogMenu () {
    $MENUBOX --title "[ --- M A I N   M E N U --- ]" --menu "Use UP/DOWN
arrows or the number of your choice to select, then press ENTER." 15 45 4 1
"Display 'Hello World'" 2 "Display 'Goodbye World'" 3 "Display 'Nothing'" X
"EXIT" 2>choice.txt
}
```

```
###
# SCRIPT SECTION #
######################

#call the menu

funcDisplayDialogMenu

case "$(cat choice.txt)" in
    1) echo "Hello World";;
    2) echo "Goodbye World";;
    3) echo "Nothing";;
    X) echo "Exit";;

esac
```

The Input Box

```
#!/bin/bash

#demonstration of an input box for user input during a script


###
# VARIABLES SECTION #
#####################
INPUTBOX=${INPUTBOX=dialog}
TITLE="Default"
MESSAGE="Something to display"
HEIGHT=10
WIDTH=40
TMPFILE=tmpfile.txt


###
# FUNCTIONS SECTION #
#####################

funcDisplayInputBox () {
    $INPUTBOX --title "$1" --inputbox "$2" "$3" "$4" 2>${TMPFILE}

}



###
# SCRIPT SECTION #
```

```
##################

funcDisplayInputBox "Display File Name" "which file in the current
directory $(pwd)/ do you want to display?" "${HEIGHT}" "${WIDTH}"

if [ "$(cat ${TMPFILE})" != "" ]
then
    cat "$(cat ${TMPFILE})"
else
    echo "Nothing to do"
fi

rm ${TMPFILE}
```

Overriding Events

```
[cloud_user@master bin]$ cat override.sh
#!/bin/bash

#override/trap the system exit and execute a custom function

###
# VARIABLE SECTION #
####################

TMP="tempfile.txt"
TMP2="tempfile2.txt"

trap 'funcMyExit' EXIT

###
# FUNCTION SECTION #
####################

#run this exit, instead of default exit when called

funcMyExit () {
    echo "Exit intercepted"
    echo "Cleaning up temp files"
    rm -rf temp*.txt
    exit 255
}


###
# SCRIPT SECTION #
##################
```

```
echo "Write something to the tmp file for later use..." > $TMP
echo "Write something to the tmp2 file for later use..." > $TMP2

cp -rf $1 newfile.txt 2>/dev/null

if [ "$?" -eq "0" ]
then
    echo "Everything worked out okay"
else
    echo "I guess it did not work okay..."
    exit 1
fi
```

Quickly checking command line parameters