LAB CONTINUE

UNIVERSITY OF THE
FREE STATE
UNIVERSITEIT VAN DIE
VRYSTAAT
YUNIVESITHI YA
FREISTATA

UFS

LAB CONFIG

- We will continue prepping our lab environment

- For sanity, the lab has been reinitialised using a predefined username and password…..chances are you already forgot the username and password

- Log in to the Jump host:

  ssh usrXX@cloud.examplesdomain.com -p 2200

  Password: Your_Password_From_The_Events_Page

UFS

LAB CONFIG

- Open a **second** PowerShell/PuTTY/Tabby/SSH Terminal

- Log in to the Jump host:
  ssh usrXX@cloud.examplesdomain.com -p 2200
  Password: Your_Password_From_The_Events_Page

  FROM THIS SESSION IN THE JUMP NODE

- Log in to your HN:
  ssh ern_admin@10.200.0.1XX

  Username: **ern_admin**
  Password: **Leggings:Nutcase:Daybed:Cut3:Gradation**

UFS

WRITING BASH SCRIPTS

UNIVERSITY OF THE
FREE STATE
UNIVERSITEIT VAN DIE
VRYSTAAT
YUNIVESITHI YA
FREISTATA
UFS

EXECUTING SCRIPTS

- A bash script is essentially only a few commands that are executed one after the other

- The first line should start by defining the shell that is used:
  #!/bin/bash
  #The above line is also known as a shebang

- After the file has been created, it can be made executable by changing the file permission:
  chmod ugo+x my_script

- The file extension does not matter, but sometimes users will add the ".sh" extension to the filename

- To execute a script (after chmod has been executed at some stage):
  ./my_script
  or
  sh my_script

Don't be passive; execute the commands in a terminal

UFS

BASH SCRIPTS CONTENT

- As mentioned, the first line in a bash script should define the shell to be used. You will notice that it starts with a **#**

- In a script, anything after a # is seen as a comment
  - There are some cases, like in the first line of a bash script (shebang), where the comment has meaning to the interpreter.
  - Another instance of "special comments" is for PBS jobs:

```
#PBS -n  TestJob
#PBS -l walltime=300:00:00
```

- If (for formatting reasons) you want to continue a command on the next line, you can use a "\" followed by nothing other than an enter/line break:

```
./configure \
        --prefix=/usr/local \
        --with-ssl
```

- The above command will be interpreted as a single command:

```
./configure --prefix=/usr/local --with-ssl
```

**It is essential not to have a space after the **
Otherwise, the space is escaped and not the new line character as intended

UFS

ENVIRONMENT VARIABLES

- Environmental variables can be used inside the script
- Environmental variables declared in the shell (the same terminal session) before the script is executed can also be addressed
- Parameters can be parsed to the script when it is executed:

  ./my_script parameter1 parameter2
- A variable is declared, and the value is assigned as follows:

  myName="Albert"

Special Variables	Description
$0	Name of the script
$1	First Parameter parsed to script
$#	Number of parameters parsed
$@	All parameters parsed to the script
$$	Process ID of the script

TESTING

- Some tests can be done in, for example, an if statement:
  [-e /home] && echo "/home exists"

Expression	Meaning
-d	Is a directory
-e	Exists (can be a file, directory or link)
-h	File is a symbolic link
-x	File/directory is executable
-eq	Number is equal to
-ne	Number is not equal
-gt	Number is greater than
-ge	Number is greater than or equal

- For more info:
  man test

UFS

EXECUTING COMMANDS INSIDE A SCRIPT

- You can execute a command within another command
- The old format used to be:
  echo "The date today is: `date +%F` " **#Character below ~**
- The new format that should be used is:
  echo "The date today is: $(date +%F) "

The output of the date command is parsed to the echo command

Another example is:

ls -l $(which yum)

Also, note the difference between the output of:

echo "**The date today is: $(date +%F)**"

AND

echo '**The date today is: $(date +%F)**'

UFS

IF & ELSE STATEMENTS

- Testing values (if statement)
- An "if" statement is closed by a "fi" statement (inverse of if)

```
if [ "$USER" == "root" ]; then
        echo "You are running as root....."
fi
```

- It is important to have a space after "[" and before "]"
- An "if not" statement is written as:

```
if ! [ "$USER" == "root" ]; then
        echo "Good, you are not root"
else
        echo "Why are you root?"
fi
```

Note the space between the "!" and the "[", also note the use of the double quotes that enclose the text variables

UFS

IF STATEMENT OPERATORS

- If you want to test two match cases, the “**and**” operator (&&) can be used:

```
i=8
if [[ $i -gt 5 &&  $i -lt 12 ]]; then
        echo "The value '$i' is between 5 and 12"
fi
```

  Note the additional “[“ at the beginning and the “]” at the end of the statement for both the above and below statements.

- The **or** operator is **||** and used as follows:

```
if [[ $today == "Saturday" || $today == "Sunday" ]]; then
        echo "Today is a weekend"
else
        echo "Ugh, it is still not the weekend"
fi
```


UFS

CASE STATEMENT

- The following case statement should be easy to interpret:

```
result=
you_selected=apple

case "$you_selected" in
"apple" | "banana" | "tomato" )
        result="a fruit"
        ;;
"cabbage" | "carrot" )
        result="a vegetable"
        ;;
*)
        result="an unknown item"
        ;;
esac
echo "You have selected '$you_selected', which is $result"
```


UFS

FOR LOOPS

- Looping through strings:

```
names="Mike John Peter Scott Anny"
for name in $names; do
        echo "The name is: $name"
done
```

- Looping through numbers:

```
for i in $(seq -w 1 100); do
        echo "Value: $i"
        sleep 0.1
done
```

The -w option was given to the sequence (seq) command to make the result automatically fit the same width (three characters):

001 002 … 008 009 010 … 099 100

UFS

WHILE LOOPS

- A while loop should be used with caution, because an endless loop can easily occur:

```
j=5
i=2
while [ $i -lt 10 ]; do
      i=$(( $j + 1 ))
      echo "$(date)  j=$j   i=$i"
done
```

- This while loop will run indefinitely because we set the value of **i** equal to 5 + 1 each iteration, without incrementing j

- Press Control+c to cancel out of the while loop

UFS

WHILE LOOP

- A while loop can be used to read the content of a file line-by-line:

```
i=0
while read current_line; do
        i=$(( $i + 1))
        user=$(echo $current_line | sed "s|:.*||g")
        # The above sed command searches for the first : in the line
        #  and removes the remainder of the line, only leaving the
        #  username, eg:  usr123:x:123:Example   becomes  usr123
        echo "Line $i:  $user"
done < /etc/passwd
echo "The file /etc/passwd contains $i entries"
```


UFS

BASIC MATH

- You can do some simple addition, multiplication etc:

```
income=5000
expenses=3250
myTotal=$(( ($income - $expenses) / 24 ))
echo "I should not be spending more than $myTotal per day"
```

- Or by using the **b**asic **c**alculator:

```
echo "400 * 2 / 5^2"  |  bc
```

- Incrementing the value of ”x" in a for loop:

```
x=0
for i in $(seq 1 50); do
        [ $i -gt 30 ] && ((x++))        #Reads: if i > 30 then increment x by 1
done
echo $x
```


UFS

FUNCTIONS

```
function my_add()
{
    first=$1
    second=$2
    result=$(( $1 + $2 ))
    echo $result
}

#Calling the function:
my_add 33 11
```


UFS

RETURN VALUES

- The output/result of a command can be assigned to a variable:

```
profile_files=$( ls /etc/profile.d )
echo "The following profile files are executed upon login:"
echo "$profile_files"
```

- The return (exit) code of a command is also very useful in scripts:

```
rpm -q chrony
result=$?
if [ $result -eq 0 ]; then
    echo "Chrony NTP is installed"
else
    echo "Chrony NTP is not installed"
fi
```

A zero (0) return code always indicates the command's success and a non-zero code indicates failure.

UFS

HERE DOCUMENTS

- A here document is a file/document generated within a script/command. It is almost like a template file that generates static or dynamic content:

```
#This example will only work if you are the root user
cat > /etc/profile.d/ufs.sh <<-EOF
alias vi='vim '
alias s='sudo -u - '
alias l='ls -la --color '
EOF
```

- The cat command redirects (>) the content to a file, until the EOF is the only content in the line.

- If you want to append to an existing file, instead of writing
  cat > write cat >>

- A dash (-) is used in front of the EOF to ignore any indentation that may exist in the script….only works inside scripts, not in the terminal.

UFS

REGULAR EXPRESSIONS

- The grep command uses regular expressions (an expression that defines a condition without specifically expressing the condition statically).

- **grep** searches for text in a file or standard output

- Examples:

  #Return (-i = insensitive) occurrences of admin in a file:

  grep -i "admin" /etc/passwd

  #Return (-o = only the matching) IPv4 addresses in a file:

  grep -o "[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*" /etc/hosts

  #View all the lines of text in a config file:

  sudo cat /etc/selinux/config

  #Show only lines that are not (-v) commented out:

  sudo cat /etc/selinux/config | grep -v "^#"

UFS

REGULAR EXPRESSIONS

- The sed command can be used to change values using regular expressions

#Disable SELinux permanently (persistent after reboots):

sudo sed -i "s|^SELINUX=.*|SELINUX=disabled|g" /etc/selinux/config

The above command reads:

Search (**s**) for text starting (**^**) with **SELINUX=**, followed by any number of characters (**.***) and replace it with SELINUX=disabled globally (**g**)

Delete all empty lines from the same file:

Definition of an empty line:

An empty line is one beginning (^) with no content up to the ending ($) of the line:

cat /etc/selinux/config |wc -l #count the number of lines in the file

sed -i "/^$/d" /etc/selinux/config #The -i option = modify the file in place

cat /etc/selinux/config |wc -l #count the number of lines in the file again

UFS

SCRIPTING EXECUTION SPEED MIGHT MATTER

- **Medical Physics**

- Script to transform a text file with 94 168 lines into a CSV file

Attempt 1:	Just make it work.	**77.37 minutes**
Attempt 2:	Adding needed arrays.	**16.45 minutes**
Attempt 3:	Performing file formats.	**8.97 minutes**

UFS

SCRIPTING EXERCISE

- Log in to the Jump host
  ssh usrXX@cloud.examplesdomain.com -p 2200

In a second terminal (if you haven't created one yet):

- Log into the HN and become root
  ssh ern_admin@10.200.0.1XX

  Username: **ern_admin**
  Password: **Leggings:Nutcase:Daybed:Cut3:Gradation**

  You should now have **two terminal sessions** open
  One: [usrXX@**login** ~]$
  And the other: [ern_admin@**usrXX-hn01** ~]$

UFS

SCRIPTING EXERCISE

- Write a script (on the jump node) to install MySQL
  - (which will later be executed on your HN)

- As a reference, use (link also on the events page as Install MySQL, under slides):

  https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-centos-7

  1. Define a variable MYSQL_PASS at the top of the script and set the root user's MySQL password to:
     Percent-Gope-Dumping-Uninsured-Maybe4

  2. Before continuing, the script should test to see if the short hostname is usrXX-hn01 (Don't hardcode your user number; test for numbers in that position)

  3. The script should continue **without ANY** user intervention

  Continue on next slide…..

UFS

SCRIPTING EXERCISE

The script should perform the following, too:

4. Install MySQL as per Step 1 on the webpage. You can use the version (7-5) that is mentioned on the page itself.
   Note that the site mistakenly executes the rpm -ivh against a different version (7-9) than the one downloaded using the curl command. Either download version 7-9 or install the downloaded version (7-5), both will work for our purposes.

DON'T execute steps 3 and 4; we'll do our own securing and testing hereon

5. Enable the MySQL service to start (now and) automatically after reboots

6. Read the value of the temporary password (last part of step 2 on the site) into a new variable **temp_pass**

7. Create a "here document" (secure_mysql.sql) with the following content:

```
UPDATE mysql.user SET Password=PASSWORD('$MYSQL_PASS') WHERE User='root';
DELETE FROM mysql.user WHERE User='root' AND Host NOT IN ('localhost', '127.0.0.1', '::1');
DELETE FROM mysql.user WHERE User='';
DELETE FROM mysql.db WHERE Db='test' OR Db='test_%';
FLUSH PRIVILEGES;
END OF DOCUMENT
```

← Don't add this line to the document itself

The value of the **MYSQL_PASS** variable should be written to the file, not the referenced name.

Continue on next slide…..

UFS

SCRIPTING EXERCISE

8. Create another here document (~/.my.cnf) with the content:
   [mysql]
   user=root
   password=$temp_pass
   END OF DOCUMENT ← Don't add this line to the document itself

9. Execute the mysql command with an additional parameter (--connect-expired-password), and redirect the file (secure_mysql.sql) to the command

10. If the above command was successful, delete the secure_mysql.sql file

11. Using a regular expression, change the password in the (~/.my.cnf) file to the value of MYSQL_PASS

12. Execute the following command; you should not be prompted for a password:
    mysql -e "SELECT Host, User from mysql.user;"

UFS

SCRIPTING EXERCISE

- Study a shell script such as **/etc/profile** and the one created
- See if you can understand what is happening in said scripts
- Practice writing your own scripts
- Copy your scripts somewhere where you can access them later

- Writing (and keeping) installation scripts for the applications discussed in these sessions is highly recommended