Introduction to shell script programming

Workshop - writing a simple script

Work in groups

- Work in groups during this workshop
  - 4-5 people, new or existing groups
- Help each other if you get stuck - even between groups (there’s no competition)

Relevance

- The aim of this workshop is to prepare you for your group assignment
- The goal is that everyone in class can write and execute at least a very basic Bash script
- Knowing how to write scripts will improve your daily work, both as a student and as a professional within IT

What is a script

- Bash is line-based
- Bash can be run interactively or as an application
- You can give a script as the argument to Bash, in which case it will execute the lines in the script one-by-one
- Anything you can do in the terminal, you can also do in a script
- Anything you can do in a script, you can also do in the terminal
- Script with execute permissions and a shebang line as the first line can be run as commands

  ```
  #!/bin/bash
  ```

Our first script

- Let’s make a script that can print basic information about your computer
- It should print a welcome message with your username and today’s time and date
- It should print the computer name and IP address
- Let’s first try all of the above in the terminal
- If it works there, it will work in the script

Some useful techniques

- `echo -n` - print without newline - next command goes to the same line
- Command substitution - `$(`*`command`*`)` - creates the text from *`command`*
  `echo "Today is $(date +%A)"` (prints e.g. Today is Monday)
- Always use quotes around text
- Semicolon allows for two commands on the same line

Printing basic information

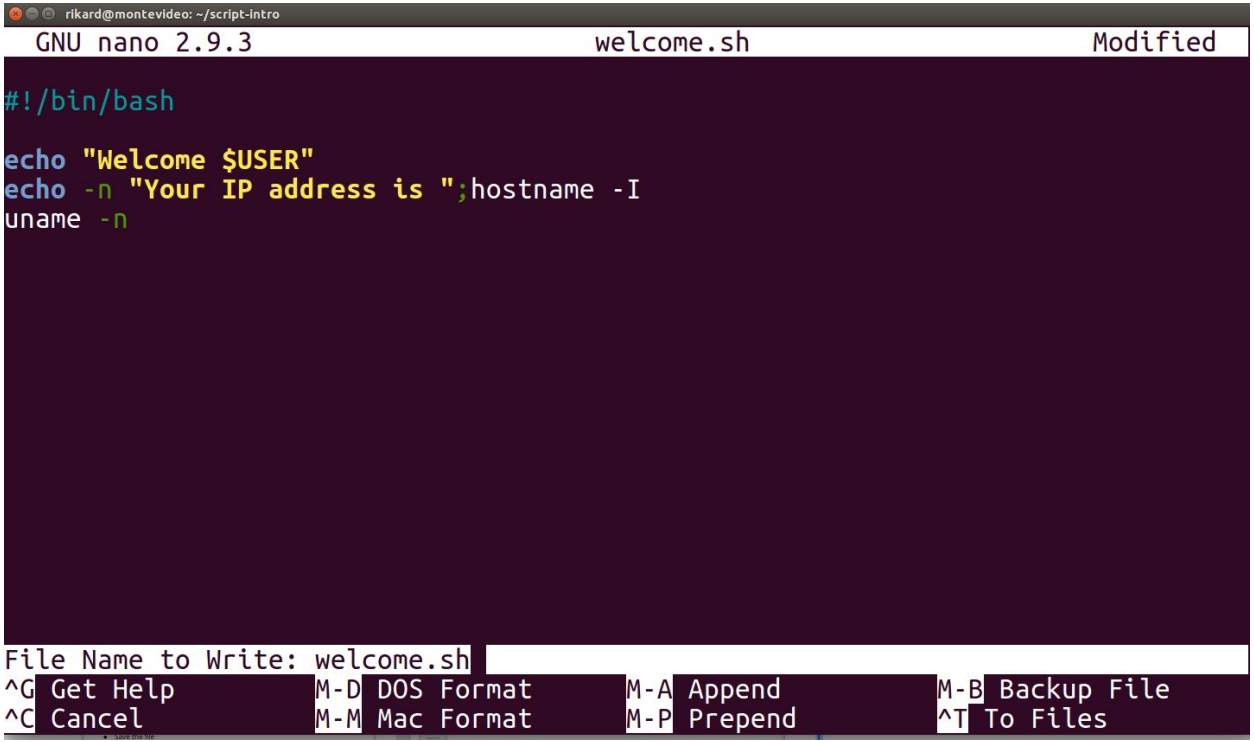
- We'll use `echo` to print and `date` for the time and date, `hostname` for the ip address and `uname` for the hostname (consusing, eh?) - The environment variable $USER contains your username

```
$ echo "Welcome $USER"
Welcome rikard
$ date
fre  2 aug 2019 11:47:35 CEST
$ echo -n "Your IP address is ";hostname -I
Your IP address is 130.241.23.135
$ uname -n
montevideo
```

Making it a script

- Create a directory for this exercise and cd down to it
- Use your editor to create the file welcome.sh
- Put the shebang line first in the script
- Put the commands in the same order in the script
- Save the file
- Add execute permissions to the file
- execute the file with `./welcome.sh` (you may use tab completion)

Using nano


rikard@montevideo: ~/script-intro
GNU nano 2.9.3 welcome.sh Modified
#!/bin/bash
echo "Welcome $USER"
echo -n "Your IP address is ";hostname -I
uname -n
File Name to Write: welcome.sh
^G Get Help M-D DOS Format M-A Append M-B Backup File
^C Cancel M-M Mac Format M-P Prepend ^T To Files

Changing permissions and running

```
$ chmod u+x welcome.sh
$ ./welcome.sh
Welcome rikard
Your IP address is 10.0.0.101
montevideo
```

We forgot the date, so edit the file and add it

```
$ cat ./welcome.sh
#!/bin/bash

echo "Welcome $USER"
echo -n "Time and date is "
date
echo -n "Your IP address is "
hostname -I
uname -n

$ ./welcome.sh
Welcome rikard
Time and date is fre  2 aug 2019 11:59:29 CEST
Your IP address is 10.0.0.101
montevideo
```

Using variables instead

- As you learned in the Bash introduction, you can create variables
- Using variables is a good habit
  - You can reuse the variables in many places in your script
  - The value can be changed in one place, and the variable used in many
  - Makes it easy to print text where part of the text may vary

```
GREETING="Welcome $USER"
TIME=$(date)
IP=$(hostname -I)
HOSTNAME=$(uname -n)
echo "$GREETING"
...etc
```

Script using variables

```
$ cat welcome.sh
#!/bin/bash

GREETING="Welcome $USER"
TIME=$(date)
IP=$(hostname -I)
HOST=$(uname -n)
echo "$GREETING"
echo "Time and date is $TIME"
echo "Your IP address is $IP and hostname is $HOST"

$ ./welcome.sh
Welcome rikard
Time and date is fre  2 aug 2019 12:09:20 CEST
Your IP address is 10.0.0.1  and hostname is montevideo
```

Create a backup with date as part of filename

- Next, we'll write a script that takes a backup of the welcome.sh script and names the backup welcome.sh.bak.2019-08-02 (if that's todays date) and then use `gzip` to zip the backup so that it takes less space
- Resulting file will be called welcome.sh.bak.YYYY-MM-DD.gz
- `gzip -9` uses the best possible compression
- To unzip, use `gunzip`
- You can format output from date:

```
$ date +%Y-%m-%d
2019-08-02
```

First attempt

```
$ cat do_backup.sh
#!/bin/bash

FILE="welcome.sh"
DATE=$(date +%Y-%m-%d)
BACKUP="$FILE.bak.$DATE"

cp "$FILE" "$BACKUP"
gzip -9 "$BACKUP"
```

Result

```
$ ls -l
total 12
-rwxrw-r-- 1 rikard rikard 119 aug  2 12:28 do_backup.sh
-rwxrw-r-- 1 rikard rikard 186 aug  2 12:08 welcome.sh
-rwxrw-r-- 1 rikard rikard 181 aug  2 12:29 welcome.sh.bak.2019-08-02.gz

# Yey! We saved five bytes from compression!
```

Arguments

- Scripts, just like most commands, can take arguments
- Arguments are for the *what* - like *what* should I backup?
- Arguments make scripts generic - you can use one script for many things
- We're thinking, `./do_backup`**`s`**`.sh` *`file1 [file2... fileN]`*
- Loop through the list of files and create a backup from each one of them
- Use the original name with the suffix of .bak.YYYY-mm-dd
- gzip -9 each backup
- Arguments go to $1 $2 etc
- You can loop over each argument

Using the for loop

- Syntax:
  `for VAR in LIST; do COMMAND(S); done`
- You can get a list of arguments using `$@`

Proof of concept

```
$ cat arguments.sh
#!/bin/bash

for ARG in "$@"
do
    echo "$ARG"
done
$ ./arguments.sh a b c d
a
b
c
d
```

The plan for the new backup

- Create a variable called SUFFIX with the value .bak.YYYY-mm-dd (use date to get the actual date)
- Loop over the arguments
- for each argument (a file to backup) make a copy that has the suffix at the end, zip it
- Things to look out for:
  - How does your script handle no arguments?
  - How does it handle arguments that aren't actually files?
  - `if [[ -e "$FILE" ]]; then command; else command; fi` - checks if file exists
- Work in your group to create the script

Example solution

```
$ cat do_backups.sh
#!/bin/bash

DATE=$(date +%Y-%m-%d)
SUFFIX=".bak.$DATE"
BAD_FILES=""

for FILE in "$@"
do
    if [[ -e "$FILE" ]]
    then
        cp "$FILE" "$FILE$SUFFIX" && gzip -9 "$FILE$SUFFIX"
    else
        BAD_FILES="$BAD_FILES $FILE"
    fi
done
if [[ -z "$BAD_FILES" ]]
then
   exit 0
else
    echo "These files were not found: $BAD_FILES"
    exit 1
fi
```

Using functions

```
$ cat functions.sh
#!/bin/bash

get_date() {
    date +%Y-%m-%d
}

add() {
    echo "$(($1 + $2))" # arguments are called $1, $2 etc in functions too
}

echo "This is done first"
get_date  # call get_date without arguments
add 10 30 # call add with two arguments
$ ./functions.sh
This is done first
2019-08-02
40
```

Summary

- Bash scripts start with #!/bin/bash
- Command substitution is useful - $(command)
- Variables are useful - TIME=$(date); echo "Today is $TIME"
- Arguments to scripts make them generic (more than one use)
  - You wouldn't want a separate ls command for each directory and file?
- You can compress a file using gzip
- You can loop through the list of arguments using
  for var in "$@"; do command(s); done
- You can test if a file exists using
  `if [[ -e "$FILE" ]]; then command; else command; fi`
- You can use functions with arguments (works also in the terminal)

Further reading

http://wiki.juneday.se/mediawiki/index.php/ITIC:Introduction_to_Bash_scripting

http://wiki.juneday.se/mediawiki/index.php/Bash_Programming

https://programminghistorian.org/en/lessons/intro-to-bash

https://ryanstutorials.net/linuxtutorial/

http://linuxcommand.org/lc3_writing_shell_scripts.php

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

https://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html