MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

COMP7004 - Systems Scripting

Lecture 2: Bash Basics

Dr. Vincent Emeakaroha
vincent.emeakaroha@mtu.ie

Semester 2, 2025

www.mtu.ie

Shell Scripting: Intro

- Shell: An interface to the Operating System.
  - If we work with Mac/Linux, the shell offers an interface (a communication protocol) to the UNIX System.
  - Equivalent to the Windows cmd.exe (but much more powerful → UNIX allows users to do many more things than cmd.exe).
- Shell interface: How do we communicate with the OS?
  - Command prompt, through which we can run our commands, programs, or shell scripts.
  - To open a command prompt in Linux, type: Ctrl + Alt + T
  - To run a command on the open prompt, just type the command, e.g.,: *pwd*
  - To close the command prompt, type: exit

Shell Scripting: Intro

- UNIX: Two major types of shells (or interfaces to the OS)
  - Bourne Shell (provides a command prompt with $ character).
  - C Shell (provides a command prompt with % character).

- Within Bourne Shell, different variants:
  - Bourne Shell, Korn Shell, Bourne Again Shell, POSIX Shell.
  - Popularly known as BASH.

Bash Scripting

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- Scripts
  - Collection of commands stored in a file.
  - Shell reads and acts on commands like normal.
  - Shell provides programming features to make scripts powerful.

- Importance
  - Scripts are easy to write.
  - No additional setup or tools are required for developing or testing scripts.
  - Task automation.
  - Unleashing compute power especially on Linux machines.

Limitation of Shell Scripts

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- Every line in Shell script creates a new process in the operating system.
- Shell scripts are slower as compared to compiled programs.
- There may be problem with cross-platform portability.
- It is not suitable in situations where:
  - Extensive file operations are required.
  - Data structure such as linked lists or trees are required.
  - Direct access to system hardware is required.
  - Port or socket I/O is required.

Bash Command Structure

- *Command <Options> <Arguments>*
- Multiple commands separated by “;”
  - Will be executed one after the other
- Commands can be executed directly on a Terminal window or used in a script.

Basic Linux Commands

Commands	Descriptions
ls	This command is used to check the content of a directory.
pwd	This command is used to check the present working directory.
mkdir *work*	This command creates a directory called “work” inside the current working directory.
cd *work*	This command will change our current working directory to the newly created directory “work”.
touch *hello.sh*	This command creates an empty file called “hello.sh” in the current directory.
cp *hello.sh bye.sh*	This command does copying. It will copy the “hello.sh” as “bye.sh”.
mv *bye.sh welcome.sh*	This command is used to rename or move files. It will rename the file “bye.sh” into “welcome.sh”.

Help Facilities for Commands

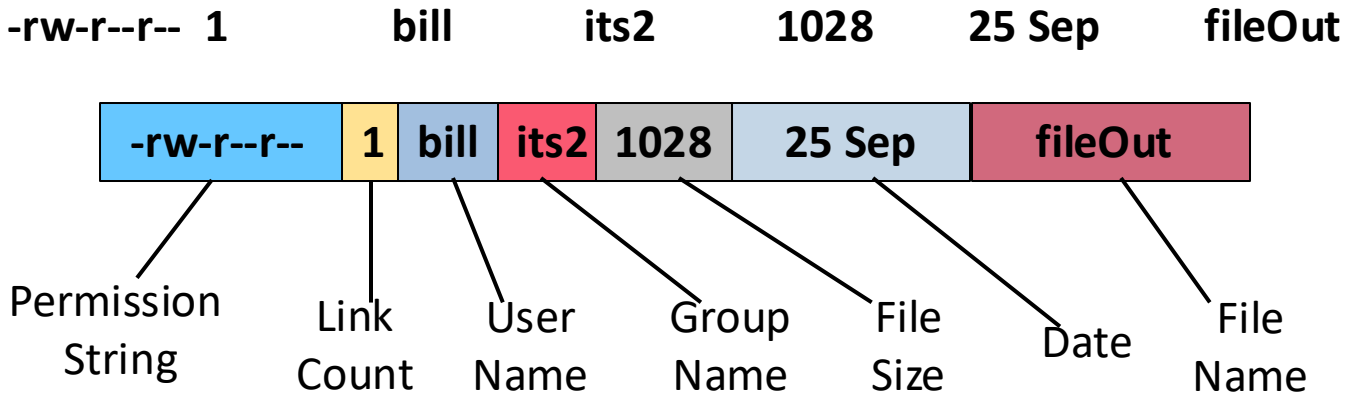
- To understand the working of a command and its possible options use
  - man *commandName*
- Use the GNU Info System (info, info command)
- Listing a Description of a Program (whatis command)
- Many tools have a long–style option, `––help', that outputs usage information about the tool, including the options and arguments the tool takes.
  - Example: id --help

Files and Directories

- Regular file
  - A collection of data items stored on disk
- Directories
  - List of files and inodes
  - “.” reference to the current directory
  - “..” reference to the parent directory
  - Root (/) – “.” and “..” are the same

Viewing File Attributes

- Command “ls –l” (Long listing)
  - Displays file properties in a directory
  - Example:



File Type Attribute

- Shows the type of file in a listing

```
-rw-r--r--  1    bill    its2    1028    25 Sep    fileOut
drw-r--r--  2    bill    its2    4096    20 Oct    movie
```

File Type	Meaning
-	Regular File
d	Directory
l	Symbolic Link
b	Block Device
c	Character Device
p	Named Pipe
s	Domain Socket

Permissions

- -rw-r-xr--
  - “-” indicates the type of file
  - “r” means read
  - “w” means write
  - “x” means execute
- 3 levels of access – user, groups and others

Operation	File	Directory
Read	Read file	List files
Write	Delete/Modify file	Create/Delete file
Execute	Run program	Access file

Manipulating Permission/User/Group

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- Linux commands
  - “chmod” for changing permission
  - “chown” for changing owner/user
  - “chgrp” for changing user group
- To add permission
  - chmod u+x filename (grant owner executable right)
- To remove permission
  - chmod g – r filename (remove read from group right)
- To change more than one permission
  - chmod u + rx filename (grant owner read and executable rights)

Numeric Permissions

Octal	Binary	Symbolic	English Translations
0	000	---	No permissions.
1	001	--x	Execute only
2	010	-w-	Write only
3	011	-wx	Write and execute
4	100	r--	Read only
5	101	r-x	Read and execute
6	110	rw-	Read and write
7	111	rwx	Read, write and execute

- Example: chmod 750 temp 7(rwx), 5(r-x), 0(---)

Types of Bash Commands

- Typically three types of commands
  - Built-in commands
  - Aliases
  - Programs
    - Written scripts
    - Most of them are part of the OS in **`/bin`**

All behave the same way

Build-in Commands

- Built-in commands are internal to the Bash and do not create a separate process. Commands are built-in when:
  - They are intrinsic to the language (**exit**)
  - They produce no side effects on the current process (**cd**)
  - They perform faster
    - No fork/exec
- Special built-in command examples
  - `: . break continue eval exec export exit readonly return set shift trap unset read`

Special Built-in Commands

exec	:	replaces shell with program
cd	:	change working directory
shift	:	rearrange positional parameters
set	:	set positional parameters
wait	:	wait for background proc. to exit
umask	:	change default file permissions
exit	:	quit the shell
eval	:	parse and execute string
time	:	run command and print times
export	:	put variable into environment
trap	:	set signal handlers

“export” Command

- The export command puts a variable into the environment so it will be accessible to child processes. For instance:

```
x=“hello”
bash            # Run a child shell.
echo $x         # Nothing in x.
exit            # Return to parent.
export x
bash
echo $x
hello           # It's there.
```

- If the child modifies x, it will not modify the parent’s original value. Verify this by changing x in the following way:

```
x=“ciao”
exit
echo $x
hello
```

Command History

- Most shells maintain a history of previously entered commands
- To view the list of commands
  - Input “history”
- To rerun a command
  - Input “!#”
  - Where # = command number

```
1993  gv ctpserver-trustlabel.eps
1994  cd output/
1995  ls
1996  cd ../
1997  ls
1998  rm output.tar
1999  mv ctpserver.eps output/
2000  exit
2001  history
user@local:~$
```

Aliases

- Many times you might want to create new commands from existing commands. Sometimes, existing commands have complex options to remember.
- We can create new commands as follows:
  - alias ll=‘ls –l’
  - alias copy=‘cp –rf’
- To list all declared aliases, use command:
  - alias
- To remove an alias, use command:
  - unalias commandName

Programs: Writing Bash Script

- Involves 3 key steps

  1) Write the script using editor of choice

  2) Assign permission to execute

  3) Put it where shell can find it

Writing Bash Script: File Naming

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- In Linux, filenames in lowercase and uppercase are different. For example:
  - The filenames “Hello” and “hello” are two distinct files
- As far as possible avoid using **spaces** in filenames or directory names such as:
  - Wrong file name – “Hello World.sh”
  - Correct file name – “Hello_World.sh” or “HelloWorld.sh”
- This is a serious pitfall and should be taken very seriously.

Writing Bash Script

- First script example (myScript.sh)

```
#!/bin/bash
# My first script

echo "Hello World!"
```

Line 1: A special clue given to the shell indicating what program is used to interpret the script. In this case it is **/bin/bash.** This line is called “shebang”. It must be the first line of every script.

Line 2: Comment - Everything that appears after a "#" symbol is ignored by bash.

Line 3: The **echo** command - simply prints what it is given on the screen (stdout)

Assigning Permission and Executing Script

Technique one:

- Assigning permission
  - **chmod 755 myScript.sh or chmod u+x myScript.sh**
- Running script
  - **./myScript.sh**
  - The ./ tells shell to execute the present script otherwise shell will look for it in the PATH environment variable.
  - One can add script to environment variable and then remove "./" when running it.

Technique two:

- Using bash command
  - bash myScript.sh
  - Assuming you saved the script in a file named myScript.sh
  - Not recommended

Filename Expansion

- Special characters representing multiple filenames
- Also referred to as globbing

Character	Matches
*	0 or more characters
?	1 character
[ ]	Matches any 1 character in [ ] (including ranges)
[^]	Matches any 1 character not in [ ] (including ranges)

Using Script to Run a Task

- We wants to run operations that copies all files into a directory, and then deletes the directory along with its contents. This can be done with the following commands:

```
mkdir trash
cp * trash
rm -rf trash
```

- Instead of having to type all that interactively on the shell, write a shell script to execute the task:

```
#!/bin/bash
# this script deletes some files
mkdir trash
cp  *  trash
rm  -rf  trash
echo “Deleted all files!”
```

Interactive Script: “read” Command

- The read command allows you to prompt for input and store it in a variable.
- Part of the built-in commands
- Syntax
  - **read [options] NAME1 NAME2……NAMEN**
- One line is read from the standard input (keyboard) at a time.
- First word stored in **NAME1**, second in **NAME2**, and so forth.
- Example:
  - read.sh

read.sh

```
1: #!/bin/bash
2: echo -n "Enter name of file to delete: "
3: read file
4: echo "Type 'y' to remove it, 'n' to change your mind ... "
5: rm -i $file
6: echo "That was YOUR decision!"
```

- Line 2 prompts for a string that is read in line 3. Line 4 uses the interactive remove (rm -i) to ask the user for confirmation.

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University