



UiO : **University of Oslo**

Getting started with shell scripting

Thomas Röblitz

`thomas.roblitz@usit.uio.no`

RCS Course Week, October 22-25, 2012



Outline

- ▶ What is shell scripting ?
- ▶ Invocation
- ▶ Variables
- ▶ Basic control flow (if-then-else-fi)

Poll

- ▶ Who did already
 - ▶ use a shell?

Poll

- ▶ Who did already
 - ▶ use a shell?
 - ▶ write a shell script?

Poll

- ▶ Who did already
 - ▶ use a shell?
 - ▶ write a shell script?
 - ▶ write a script (R, MATLAB, Bioportal, ...)?

What is shell scripting?

- ▶ A way to store, edit & execute shell commands
 - ▶ store $\Rightarrow$ avoid re-typing again and again
 - ▶ edit $\Rightarrow$ easily change commands
 - ▶ execute $\Rightarrow$ execute commands (interactive or batch mode)
- $\Rightarrow$ Simplifies the repetition of long & complex command sequences.

What is shell scripting used for?¹

- ▶ A wrapper for a program
 - ▶ aka SLURM file
 - ▶ prepare job input data, run job, check results, copy results
 - ⇒ May help in avoiding (unnoticed) job failures.
- ▶ To manage research environment
 - ▶ create data/jobs layout (e.g., for parameter studies)
 - ▶ perform post-processing of results
 - ⇒ May help in reproducing results if necessary.

¹ on Abel

Simple script: Edit `simple.sh`

- ▶ use your favourite (ASCII) text editor: vim, emacs, ...

```
#!/bin/bash

WHO=All

if [ $# -gt 0 ]
then
    WHO=$1
fi

DATE=$(date)
echo "Welcome_ '$WHO' _to_RCS_Course_Week\n$DATE"
```

Simple script: Running

```
> sh simple.sh
```

```
Welcome 'All' to RCS Course Week  
Mon Oct 22 21:48:10 CEST 2012
```

```
> sh simple.sh Abel
```

```
Welcome 'Abel' to RCS Course Week  
Mon Oct 22 21:48:50 CEST 2012
```

Step-by-step

- ▶ hashbang
- ▶ invocation & arguments
- ▶ variables
- ▶ if-then-else-fi
- ▶ tests

Hashbang²

- ▶ 1st line in each shell script
- ▶ hashbang #!
- ▶ command interpreter, e.g. /bin/bash

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

- ▶ How about perl, /bin/perl ?

```
> which perl  
/usr/bin/perl
```

which *shows the full path of (shell) commands*
(see `man which`)

² aka shebang, pound-bang, hash-pling, ...

Invocation – Explicit

- ▶ use shell interpreter explicitly

```
> sh simple.sh
```

- ▶ Why `sh` and not `bash` ?

```
> which sh
```

```
/bin/sh
```

```
> file /bin/sh
```

```
/bin/sh: symbolic link to `bash`
```

- ⇒ `sh` is a symbolic link to `/bin/bash` (the default shell in a Linux system)

Invocation – Implicit

- ▶ Run script without `sh ...?`

```
> ./simple.sh
```

```
-bash: simple.sh: Permission denied
```

- ▶ Check

```
> ls -l simple.sh
```

```
-rw-r--r-- 1 thomarob users 117 Oct 22 18:04 simple.sh
```

- ▶ Fix

```
> chmod u+x simple.sh
```

```
> ls -l simple.sh
```

```
-rwxr--r-- 1 thomarob users 117 Oct 22 18:04 simple.sh
```

Invocation – Scientists have no time

- ▶ How about *saving two more keystrokes*?

```
> simple.sh
```

```
-bash: simple.sh: command not found
```

- ▶ Check

```
> which simple.sh
```

```
/usr/bin/which: no simple.sh in (/hpc/bin:...
```

`which` *searches the directories in the environment variable* **PATH**

```
> PATH=$PATH:$PWD
```

```
> simple.sh
```

```
Welcome 'All' to RCS Course Week  
Mon Oct 22 21:48:10 CEST 2012
```

- ▶ Alternative: use absolute path, i.e., `/path/simple.sh`

Arguments – Welcome 'everybody'

- ▶ Passing arguments

```
> arguments.sh everybody
```

```
Welcome 'everybody' to RCS Course Week
```

- ▶ Simplified script

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

```
echo "Welcome_ '$1' _to_RCS_Course_Week"
```

- ▶ `$1` is a *positional parameter*
 - ▶ positional parameters are set when the shell is invoked
 - ▶ set with *everybody* in the invocation above

Arguments – Welcome everybody

- ▶ Passing multiple arguments

```
> argumentsX.sh bash csh
```

```
Welcome 'bash' (1) to RCS Course Week  
Welcome 'csh' (2) to RCS Course Week  
Welcome '' (3) to RCS Course Week
```

- ▶ Enhanced script

```
#!/bin/bash  
echo "Welcome_'$1'_(1)_to_RCS_Course_Week"  
echo "Welcome_'$2'_(2)_to_RCS_Course_Week"  
echo "Welcome_'$3'_(3)_to_RCS_Course_Week"
```

- ▶ $\$n$ is a *positional parameter*
- ▶ unused positional parameters are empty

Variables

- ▶ Use (also Substitution)

`$VARIABLENAME`

- ▶ no need to declare before use (default: empty string)

- ▶ Assignment

`VARIABLENAME=any (un)quoted string`

- ▶ `VARIABLENAME` is of characters `a-z`, `A-Z`, `0-9`, `_`, ...
 - ▶ no space between `VARIABLENAME` and `=` and value
 - ▶ quoting (`"` or `'`) needed if value contains spaces
- ▶ Note the differences (with/out `$`)
- ▶ untyped (think: everything is a string)

Variables – Examples

▶ Script

```
#!/bin/bash
ABEL=Abel
ABLE="Abel_is_able"
echo "Welcome_' $ABEL' _ (ABEL) _to_RCS_Course_Week"
echo "Welcome_' $ABLE' _ (ABLE) _to_RCS_Course_Week"
echo "Welcome_' $UNSET' _ (UNSET) _to_RCS_Course_Week"
```

▶ Output

```
Welcome 'Abel' (ABEL) to RCS Course Week
Welcome 'Abel_is_able' (ABLE) to RCS Course Week
Welcome '' (UNSET) to RCS Course Week
```

▶ Experiment with quoting

- ▶ remove double quotes in assignment of ABLE
- ▶ remove single and/or double quotes in echo for UNSET

Variables – Environment (1)

- ▶ Many useful variables defined in the environment
- ▶ General (run `env` after logging in), e.g.

<code>PWD</code>	current path/directory
<code>HOME</code>	home directory
<code>PATH</code>	searched for executables/scripts
<code>SHELL</code>	the current shell

- ▶ Can you (now)³ explain the following command?

```
> PATH=$PATH:$PWD
```

³See example on slide 14.

Variables – Environment (2)

- ▶ Job (run `env` after `qlogin ...`), e.g.

<code>SLURM_JOBID</code>	job id
<code>SCRATCH</code>	directory for in/out data on <code>/work</code>
<code>SLURM_JOB_NODELIST</code>	list of nodes allocated to the job
<code>SLURM_TASKS_PER_NODE</code>	number of tasks per node
<code>SLURM_MEM_PER_CPU</code>	memory per core

Status

- ▶ You should now understand the **highlighted** parts.

```
#!/bin/bash

WHO=All

if [ $# -gt 0 ]
then
    WHO=$1
fi

DATE=$(date)
echo "Welcome '$WHO' to RCS Course Week\n$DATE"
```

Variables – Command substitution

- ▶ Suppose we want to use the same date multiple times

```
#!/bin/bash
date +"Script_starts_at_%c"
sleep 3 # do sth
date +"Script_started_at_%c"
date +"Script_ends_at_%c"
```

```
Script starts at Tue 23 Oct 2012 09:02:46 AM CEST
Script started at Tue 23 Oct 2012 09:02:49 AM CEST
Script ends at Tue 23 Oct 2012 09:02:49 AM CEST
```

- ▶ need to save the start time in a variable

Variables – Command substitution(2)

- ▶ Syntax

VARIABLENAME=\$(command with arguments)

- ▶ Variable is substituted with the command's output.

```
#!/bin/bash
START=$(date +%c)
echo "Script_starts_at_$START"
sleep 3 # do sth
echo "Script_started_at_$START"
date +"Script_ends_at_%c"
```

```
Script starts at Tue 23 Oct 2012 09:10:38 AM CEST
Script started at Tue 23 Oct 2012 09:10:38 AM CEST
Script ends at Tue 23 Oct 2012 09:10:41 AM CEST
```

- ▶ Command substitution is a powerful tool. Try it !

If-then-else – Example

► Example

```
#!/bin/bash
DATE=$(date +%s)
START=$(date -d "Tue_23_Oct_2012_01:15:00_PM_CEST" +%s)
if [ $DATE -gt $START ]
then
    echo "RCS_course_already_started"
else
    echo "RCS_course_not_yet_started"
fi
```

```
RCS course not yet started
```

If-then-else – Syntax

- ▶ `if [conditional_expression]
then
 statements
else
 statements
fi`
- ▶ `else` **branch** is optional
- ▶ `[ conditional_expression ]` **same as**
`test conditional_expression`
- ▶ **Note!** Must not remove spaces around `[` and `]`

Tests – Conditional expressions

- ▶ For details see `man test`
- ▶ Examples

expression	evaluates to true if
<code>! EXP</code>	<code>EXP</code> is false
<code>EXP1 -a EXP2</code>	both <code>EXP1/2</code> are true
<code>EXP1 -o EXP2</code>	either <code>EXP1/2</code> is true
<code>STRING1 = STRING2</code>	strings are equal
<code>STRING1 != STRING2</code>	strings are not equal
<code>INT1 -eq INT2</code>	integers are equal
<code>INT1 -gt INT2</code>	$\text{INT1} \geq \text{INT2}$
<code>-e FILE</code>	<code>FILE</code> exists
<code>-x FILE</code>	<code>FILE</code> exists incl. execute permission

Putting it all together

```
#!/bin/bash

WHO=All

# $# is the number of arguments
if [ $# -gt 0 ]
then
    WHO=$1
fi

DATE=$(date)
echo "Welcome_' $WHO' _to_RCS_Course_Week\n$DATE"
```

Become a shell scripting user/expert

- ▶ You just learned the ver basics: hashbang, invocation, arguments, variables, if-then-else & tests
- ▶ 2nd course this week:

Intermediate shell scripting

Andreas Buzh Skau

Wednesday, Oct 24, 12:15

- ▶ read man pages: `man bash`, `man test`
- ▶ search internet for tutorials
- ▶ study examples and scripts of co-workers
- ▶ **Most important!** Practice, practice, practice.

Happy scripting