Automatisierung mit Bash

Reto Kromer • AV Preservation by reto.ch

Weiterführender Memoriav-Workshop
Automatisierung von FFmpeg mit Bash
Lichtspiel, Bern, 12. Januar 2023

Didaktische Vereinfachung

- Skripte werden interpretiert und können unmittelbar ausgeführt werden.
- Programme müssen kompiliert werden, bevor sie ausgeführt werden können.

Einführung

Grundelemente

- Parsen der Eingabe
- Kontrollstrukturen: Verzweigung, Schleife, Rekursion
- Datentypen: Variable, Array
- Unterprogramme
- Formattieren der Ausgabe

Shebang

```
#!path_to_shell
#!path_to_programming_language

#!/bin/sh
#!/bin/bash
#!/usr/bin/env bash
```


Command structure

```
  $0        $1             ${n}
command argument_1 ... argument_n
```

common syntaxes of arguments include:

```
--parameter
--parameter=value
-p
-p value
```


Edit and run a script

```
nano script

chmod +x script
sudo chmod +x script

./script
drag-and-drop_script
```


Bash Syntax

Conventions used

```
${THIS_IS_A_CONSTANT}
${this_is_a_variable}

this_is_a_function() {
  commands
}
```


Quoting

```
tmp=test_best_test_best_test
echo tmp
echo $tmp
echo 'temp = $tmp'
echo "temp = $tmp"
echo 'temp = "$tmp"'
echo "temp = '$tmp'"
echo "temp = ${tmp}"
```


Variable substitution (1)

```
echo "${tmp/test/TEST}"
echo "${tmp/best/TEST}"
echo "${tmp//test/TEST}"
echo "${tmp//best/TEST}"
echo "${tmp/#test/TEST}"
echo "${tmp/%test/TEST}"
echo "${tmp/#best/TEST}"
echo "${tmp/%best/TEST}"
```


Variable substitution (2)

```
echo "${tmp/es/}"
echo "${tmp//es/}"

echo "${tmp^^}"
echo "${tmp,,}"
echo "${tmp}" | awk '{print toupper($0)}'
echo "${tmp}" | awk '{print tolower($0)}'
```

Substitution and quoting

```
echo "${tmp:-sorry, isn"'"t defined}"
unset tmp
echo "${tmp:-sorry, isn"'"t defined}"
echo "${tmp}"
```


Write to indexed arrays

```
names[1]='Adam'
names[0]='Eve'
names=([1]='Adam' [0]='Eve')
names=('Eve' 'Adam')

names+=('Cain' 'Abel')
```


Read from indexed arrays

```
${names}
${names[n]}
${names[*]}
${names[@]}
${#names[*]}
${#names[@]}
```


Associative arrays

write

```
declare -A array_name
id=([Eve]='1234' [Adam]='5678')
```

read

```
${id[Eve]}
${id[Adam]}
```

Flow control Bash statements

branching

- if
- case
- select

looping

- for
- while
- until

Syntax for an **if** statement

```
if condition ; then
  command
elif condition_n ; then
  command_n
else
  command
fi
```


Syntax for a **case** statement

```
case variable in
  value_n )
    commands_n
    ;;
esac
```


Syntax for a **while** statement

```
while condition ; do
  commands
done
```

until statement syntax

```
until condition ; do
  commands
done
```


Simple redirection

```
command > file

command >> file

command < file
```


Multiple redirection

```
command > file 2>&1
command &> file

command | tee [options] file

command 2>&1 | tee [options] file
```


Debugging

Symptoms of buggy scripts

syntax error

- script doesn't run

logic error

- script runs, but doesn't work as expected

logic bomb

- script runs and works as expected, but has nasty side effects

Trace variables and flow

- use **echo** or **printf** commands at critical points to trace the content of the variables
- use **tee** command at critical points to trace the flow

Built-in Debugging

Run in debugging mode (1)

```
#!/usr/bin/env bash -x

#!/usr/bin/env bash -x -v
#!/usr/bin/env bash -xv
```

Run in debugging mode (2)

bash -x *script*

bash -xv *script*

bash -xvn *script*

bash -vn *script*

Debug a section

```
# start debugging
set -x
code_to_debug
# stop debugging
set +x
```


Debugging options (1)

set -x

set -o xtrace

- print command traces before executing command

Debugging options (2)

set -v

set -o verbose

- print shell input lines as they are read

Debugging options (3)

```
set -n
set -o noexec
```

- check for syntax errors without actually running the script

Debugging options (4)

```
set -f
set -o noglob
```

- disable file name generation using meta-characters ("globbing")

Debugging options (5)

```
set -u
set -o nounset
```

- the script runs, but gives an error message and aborts if there are unbound variables

Change the tracing prompt

```
PS4='[${BASH_SOURCE}:${LINENO}]+ '

set -x  # start debugging
code_to_debug
set +x  # stop debugging
```

Debug output to a separate file

```
exec 5> debug_output.txt
BASH_XTRACEFD='5'
```


All together

```
PS4='[${BASH_SOURCE}:${LINENO}]+ '
exec 5> debug_output.txt
BASH_XTRACEFD='5'

bash -x script
```


Use **trap** with **DEBUG**

```
trap 'read -p
"[${BASH_SOURCE}:${LINENO}]
${BASH_COMMAND}"' DEBUG
```


Debugging Tools

Bash debugger script

bashdb [*options*] *script* [**--**] [*script_options*]

bashdb [*options*] **-c** *command_string*

bash --debugger [*bash_options*] *script* [[**--**] *script_options*]

Text editor

- syntax highlighting
- syntax verification

"A hackable text editor for the 21st Century"

➔ https://atom.io

Good Practices

Programming style

- comment the code
- use meaningful names
- structure the code in logical units
- use indentation
- be consistent

Value initialisation

- value passed:
  - at program call
  - interactively when running
- value read from a configuration file
- default value coded in the program

User interface

- always provide a built-in help (-h, --help)
- if you distribute complex scripts, provide a documentation, such as a manual page ("man")
- create temporary log files when useful

Chain Scripts


```
# chain successive scripts

while script in "${script_list}"; do
  ${script} "${parameters[@]}"
  if (( $? != 0 )); then
    report_error
  else
    report_success
  fi
done
```

```
# batch process successive jobs

while IFS= read -r job; do
  ${job}
  if (( $? != 0 )); then
    report_error
  else
    report_success
  fi
done < "${job_list}"
```



```
# batch process successive script chains

while IFS= read -r job; do
  while script in "${script_list["${job}"]}"; do
    ${script} "${param[@]}" &>/dev/null & wait $!
    if (( $? != 0 )); then
      report_error
      break
    else
      report_success
    fi
  done
done < "${job_list}"
```


AV Preservation by
reto.ch

Sandrainstrasse 3
3007 Bern
Switzerland

Web: reto.ch
Email: info@reto.ch