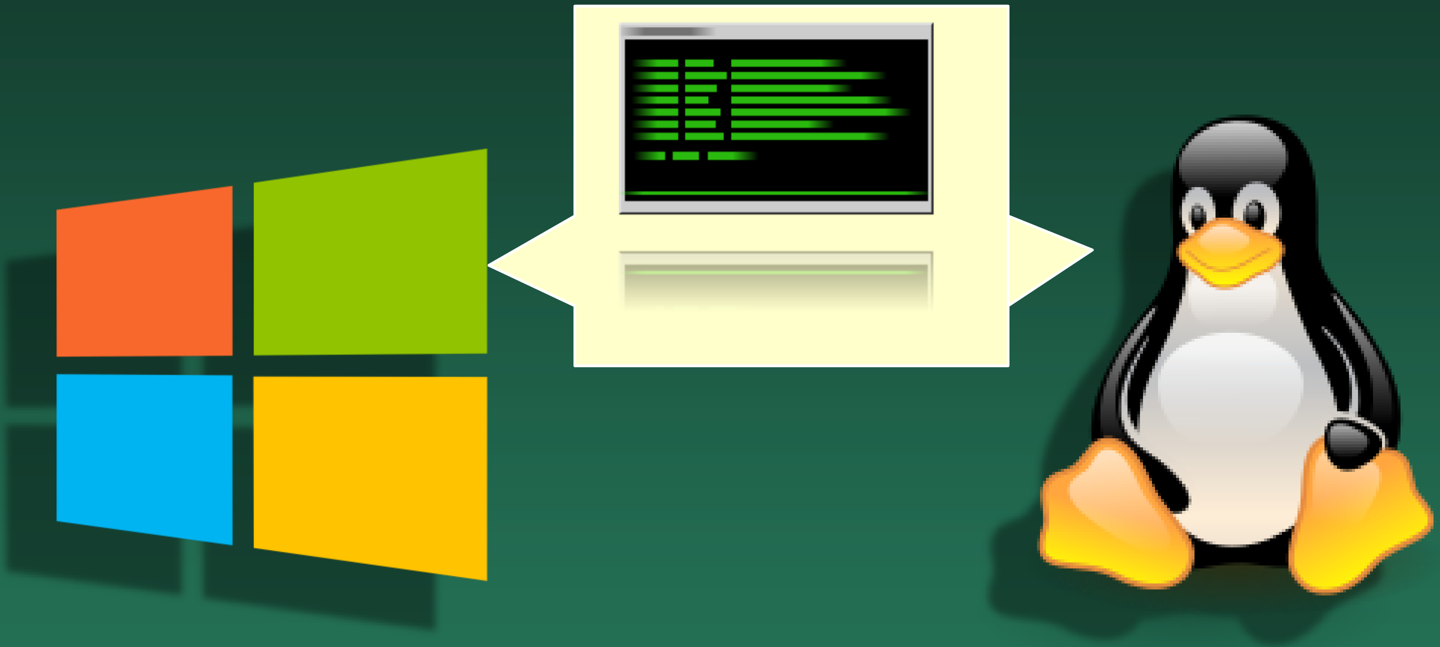
System shells

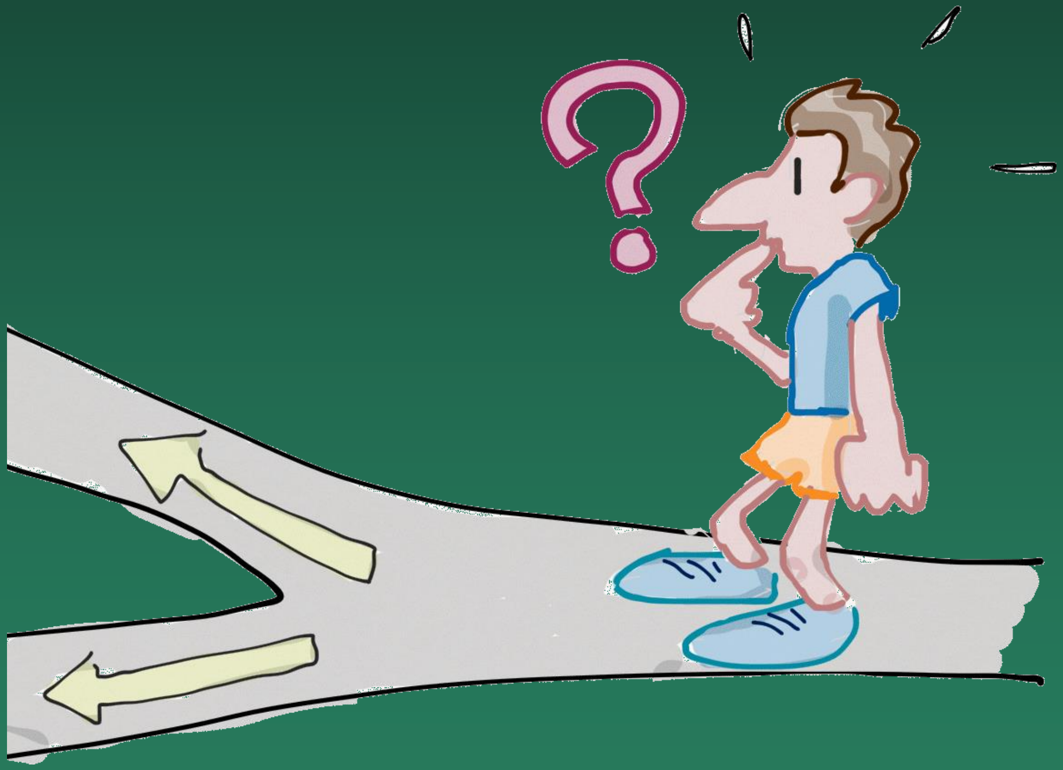


BASH syntax - 9

4 - Control Flow

if, case, while, for …

Choices



The if expression

```
if COND; then [elif COND; then cmd;]…[else cmd;] fi
```

- General structure
  - *COND* command returning a boolean value 0 or 1
  - Base commands like `grep`
  - Evaluation expression strings `[[ condition ]]`
  - Evaluation expression on numbers `(( condition ))`

  - Only one of the *cmd* is executed

Check information

- Sample script

```
#!/bin/bash
read -p "... Do you feel good (Y/N)? " ANSWER

if [[ ${ANSWER:0:1} == [Yy] ]]; then
   echo "YOU FEEL GOOD :-)"
elif [[ ${ANSWER:0:1} == [Nn] ]]; then
   echo "YOU FEEL BAD :-("
else
   echo "Wrong answer..."
fi
```

- Run the script
  - Users may type Y, Yes, y, yes
  - Users may typed N, No, nope, no
  - Information about wrong input

Check files & folders

- Sample script

```
#!/bin/bash
if [[ ! -d ${HOME}\LABS ]]; then
  mkdir ${HOME}/LABS
fi

cd ${HOME}/LABS
if [[ -f info.txt ]]; then
 echo $(date) >> info.txt
else
 echo "NEW" > info.txt
fi
```

- Run the script
  - Will create LABS if necessary
  - Will append date on existing file, write NEW otherwise

Check ERRORS

- Sample script

```
#!/bin/bash
mkdir $HOME/LABS 2> /dev/null
mkdir ${HOME}/LABS 2> /dev/null
if [[ $? == 1 ]]; then
   rm -r ${HOME}/LABS
   echo removed LABS
Else
   echo created LABS
fi
```

- Status of last command in `$?`
  - `$?` = 0 success
  - `$?` = 1 failure

Comparison == operator

- Sample script

```
#!/bin/bash
VAR1=AAA
VAR2='AAA'
VAR3="AAA"
if [[ "$VAR1" == $VAR2 ]]; then
   echo 1 equal
fi
if [[ $VAR1 == $VAR3 ]]; then
   echo 2 equal
fi
```

- The " and ' separators are eliminated

Extended string operators

- Use in `[[...]]` expression
- String Comparison
  - Operators `==`, `!=`, `<` and `>`
  - Give a result based on alphabetic order
- Generic comparison (use in `[[...]]`)

`-eq`	=
`-ne`	!=
`-lt`	<
`-le`	<=
`-gt`	>
`-ge`	>=

  - Operands are strings → alphabetic order
  - Operands conform with numbers → numeric order

Operator == and numbers

- Sample script

```
#!/bin/bash
ELEVEN=11
OELEVEN=013
HELEVEN=0xB

if (( $ELEVEN == $OELEVEN )); then (echo "OCTAL") fi
if (( $ELEVEN == $HELEVEN )); then (echo "HEXA ") fi
```

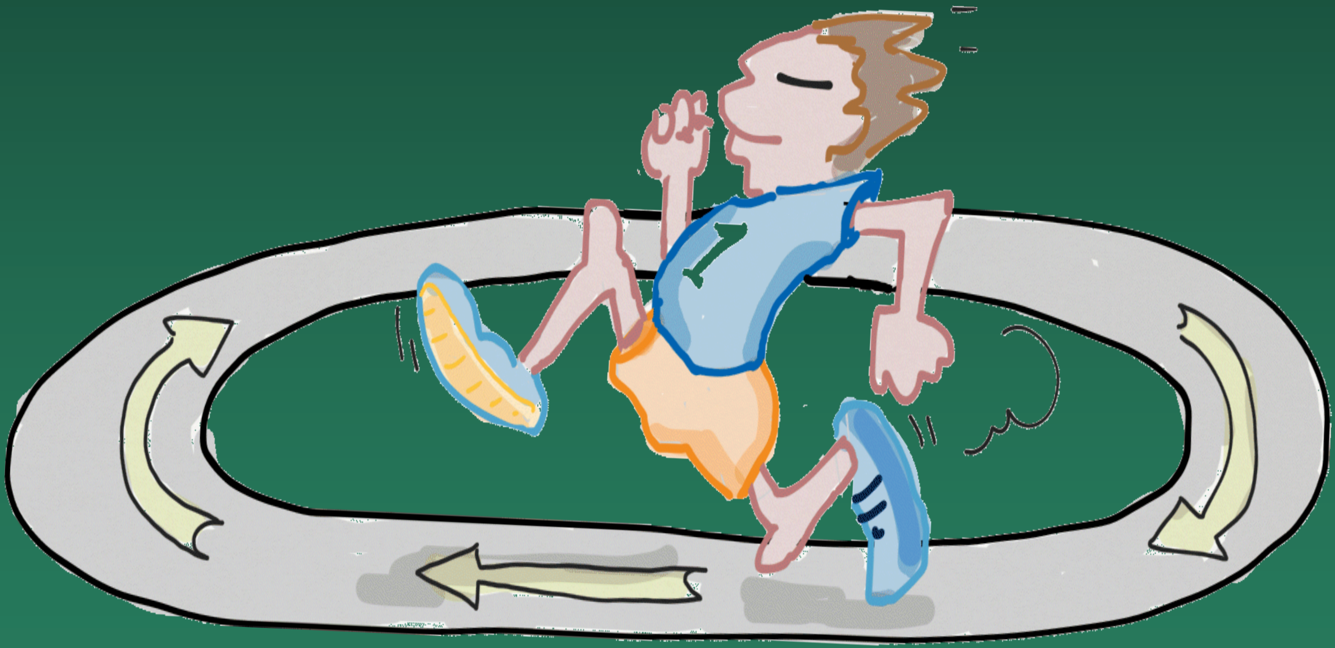
- Use `(( ... ))` to compare numbers
  - The « number behind » is evaluated
  - With `[[ ... ]]` the strings are compared
    - Unless arithmetic operators are used (`-eq -gt -ge` ...)

Extended number operators

- Integer comparison operators (use in `((...))`)
  - `==`
  - `!=`
  - `<`
  - `<=`
  - `>`
  - `>=`

- Use as an alternative to `[[ -eq`, `-gt` etc. `]]`

Loops



The while keyword

`while` *expr* `;` `do` *command* `done`

- Sample script

```
#!/bin/bash

INDX=1
while (( INDX < 10 )); do
  echo ${INDX}
  INDX=$((INDX+1))
done
```

- Useful for infinite loop with exit
  - `while true; do …`
  - End with `break`, or `exit`

Check information

- Sample script

```
#!/bin/bash
while true; do
   read -p "... Do you feel good (Y/N)? " ANSWER
   if [[ ${ANSWER:0:1} =~ [Yy] ]]; then
      echo "YOU FEEL GOOD :-)" ; exit
   elif [[ ${ANSWER:0:1} =~ [Nn] ]]; then
      echo "YOU FEEL BAD :-(" ; exit
   else
      echo "Wrong answer, retry‟
   fi
done
```

- Run the script
  - Inform users who typed "Y" (not Yes, y, yes)
  - What about "N" ?
  - What about other answers (Dunno, QWertY) ?
  - Can I have another chance to type ?

Process a file

- Sample script

```
#!/bin/bash
# will extract lines of the file [argument1]

while read -r LINE; do
  echo ${LINE}
done < $1
```

- Useful to process all lines of a file
  - … Remember we have many commands
  - `grep, head, tail, wc, sed`

- The until alternative construct

```
until expr; do command done
```

The for keyword

`for` *each-item* `in` *collection*`;` `do` *command* `done`

◆ Sample script

```
#!/bin/bash
for i in aa bb cc; do
   echo ${i}
done
for i in *01.sh *02.sh; do
   echo ${i}
done
for i in {1..5}; do
   echo ${i}
done
for (( i=0; i <= 5; i++ )); do
   echo ${i}
done
for i in {0..7..2}; do
   echo ${i}
done
```

list

Filter(s)

range

C style

min, max, increment

The case keyword

```
case item in (pattern|pattern…) list;;… esac
```

- Useful for filtering

```
#!/bin/bash
read -p "Enter a letter: " letter
case $letter in
  a|A)
    echo "You entered 'a' or 'A'."
    ;;
  b|B)
    echo "You entered 'b' or 'B'."
    ;;
  *)
    echo "You entered a letter other than a, or b.
    ;;
esac
```

User input re-loaded

- The `select` keyword

```
#!/bin/bash
choices=("Choice 1" "Choice 2" "Quit")
select choice in "${choices[@]}"
do
  case $choice in
    "Choice 1")
      echo "You chose Option 1"
      ;;
    "Choice 2")
      echo "You chose Option 2"
      ;;
    "Quit")
      echo "Goodbye!"
      break
      ;;
    *) echo "Invalid, try again.";;
  esac
done
```

```
1) Option 1
2) Option 2
3) Option 3
4) Quit
#?
```