BASH — BASH naming limitations

- Forbidden character in UNIX names
  - `/` slash = namespace separator
  - `<NewLine>` = display separator

- BASH special characters `<>|&;*?()"'`\${}[]`

- + Separators `<Space>`, `<TAB>` and `<NewLine>`
  - All can be escaped with `\`
  - Or deactivated inside quotes `'...'` *excluding* `'`
  - Or deactivated inside dbl-quotes `"..."` *excluding* `"`\$`
  - Possible file names:

```
'mon fichier.txt'
"my'file.txt"
'my>file.txt'
'my?file.txt'
```

```
mon fichier.txt
my'file.txt
my>file.txt
my?file.txt
```

→ *BEWARE: file & folder names with BASH characters may cause bugs in « unaware » scripts*

BASH — Pattern matching for names

- Generalize names with *metacharacters*
  - `*` Zero or any number of any character
  - `?` One of any character
  - [class] One character in the group i.e `[Aez12]`
  - [^class] One character NOT in the group `[^ezEZ]`
  - [x-y] One character in the interval `[0-9]`
  - `\` Escape for literal usage of a metacharacter `\*`
- Works with `ls`, `rm`, `mv`, `cp`, etc.

BASH — Commands on multiple lines

- Type `(md AA`
  - Will not execute command
  - Asks for more information until we type `)`
  - May be nested
- To split a command on multiple lines
  - Escape `<EOL>` with `\`
  - Type `(mkdir \` ← beware the space separator
  - Type `AA \` ← beware the space separator
  - Type `BB\`
  - Type `CC)` ← will create folder "BBCC"

BASH — Use variables

- Variables and special characters
  - `VAR1="my variable"`
  - `$VAR1` → `my variable`
  - `"$VAR1"` → `my variable`
  - `'$VAR1'` → `$VAR1`
  - `${VAR1}` → `my variable`
  - `{$VAR1}` → `{my variable}`
- Variable substitution
  - `VAR2="VAR1"`
  - `${!VAR2}` → `my variable`
  - `${VAR3:-"none"}` → `none`
  - `${VAR3:-$VAR1}` → `my variable`

BASH — Variable operations

String variable length	`${#VAR}`
Substring substitution	`${VAR/pattern/replace-string}`
Prefix remove to first pattern	`${VAR#*delete-pattern}`
Prefix remove to last pattern	`${VAR##*delete-pattern}`
Suffix remove from last pattern	`${VAR%delete-Pattern*}`
Suffix remove from first pattern	`${VAR%%delete-pattern*}`
Reduction to end	`${VAR:offset}`
Reduction of length	`${VAR:offset:length}`
Upper case	`${VAR^^}`
Lower case	`${VAR,,}`

BASH — Declare & use Arrays

- Declare `declare -a TABL=(AAA BB CCC)`
- Use index `echo ${TABL[1]}` → BB
  - `echo ${TABL[@]:1:2}` → BB CCC
- Serialize `echo ${TABL[@]}` → AAA BB CCC
- Cardinal `echo ${#TABL[@]}` → 3
- Element size `echo ${#TABL[1]}` → 2
- Clear element `unset ${TABL[1]}`

BASH Brace expansion

- Special brace expression
  - `{a..z}` → `a b c d e f ... x y z`
  - `{0..1}` → `0 1 2 3 4 5 6 7 8 9`
  - `{$from..$to}`
- Array construction
  - `ALPHA=({A..Z})`
  - `${#ALPHA[@]}` → `26`
- `String generation`
  - `DIGIT=({0..9})`
  - `TYPO=(${ALPHA[@]} ${DIGIT[@]})`
  - `PASS=$(shuf -n8 -er ${TYPO[@]} | paste -sd "")`

BASH 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

BASH 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

BASH Extended number operators

- Integer comparison operators (use in `((...))`)
  - `==`
  - `!=`
  - `<`
  - `<=`
  - `>`
  - `>=`
- Use as an alternative to `[[ -eq, -gt etc. ]]`

BASH 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`

BASH 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