Oscar Petersson, Matteus Laurent

oscpe262, matla782

TDDI41 Lab report

Högskoleingenjörsutbildning i datateknik, 180 hp

SCT

1-1

`$@` looks at each argument individually, whereas `@*` regards them as a collection as shown by the example below:

```
#foo.sh
#!/bin/bash
printf "%s\n" "$@"
printf "%s\n" "$*"
#EOF

 $ ./foo.sh meep moop
  meep
  moop
  meep moop
```

2-1

`$@`

will make another evaluation of the parameter which can yield unwanted results with reserved characters. For instance, if fed `'ls -F'` when an executable is present in the path (annotated in the output with `'*'`, e.g. `foo*`), it will evaluate the expression (`foo*` becomes *"all files beginning with*`"foo"`) which might result in duplicate arguments.

2-2

Yes, and no. `$*` could work, but we would still run into the issues mentioned in *2-1* with evaluation of the arguments provided. `$*` would not work however, unless it is only one file provided as argument.

3-1

```
#/bin/bash
i=10
while [ $i -ne 0 ]
do
```

```
    echo "$i"
    i=$(($i - 1))
done
```

5-1

Evaluates `$line`, string position 0, 1 character(s).

5-2

: == true, which is needed as conditional blocks can't be empty.

5-3

`$(command)`

5-4

Arithmetic expansion. Allows evaluation of arithmetic expressions and substitution of the result, allowing nesting for instance.

5-5

`$ ls -a | grep "^\." | wc -l`

6-1

a) A backslash is used as en escape character in bash, and in this case it escapes the newline, allowing the script to be written on several (two here) lines while being evaluated as if it was a single line.

b) Earlier, we saw that command scope within () are executed in a new sub-shell. If replaced by {} the scope is executed in the current shell context.

6-2

`local` creates a temporary object that is local to the current call frame, in this case a variable `$mailserver`.

6-3

To demonstrate `sleep`?

7-1, 8-1

See separate files.