BASH PROGRAMMING CHEAT SHEET

1) Conditions

* if [test]; then supports:
- file-based conditions
- string-based conditions
- arithmetic conditions.

* if [[test]]; then supports:

All [ ]- conditions, but also:
- shell globbing: if [[$v == *[sS]tr]]
  returns true if v ends in string
  or String.
- prevents word splitting:
  v = "Hello World"
  if [[$v == "Hello World"]]; then
  works.
- NO filename expansion: *.sh
  means literally *.sh
  (note: does work in [ ], but if multiple
  files → error; crash)
- || and &&
- str =~ regex: true if str matches
  regex pattern.

* if ((test)); then supports only:
- n == n'
- n != n'
- n > n'
- n < n'
- n >= n'
- n <= n'
- || and &&

T1: file-based conditions

-a f	f exists
-e f	f exists
-d f	f is directory
-f f	f is regular file
-h f	f is symbolic link
-r f	f is readable
-s f	f > 0 bytes
-w f	f is writeable
-x f	f is executable
f -nt f'	f changed more recently than f'
f -ot f'	f " longer ago " f'
f -ef f'	f inode = f' inode.

T2: string-based condition

str == str'	str equals str'
str != str'	str doesn't equal str'
str > str'	str sorts after str'
str < str'	str sorts before str'
-n str	str is not empty
-z str	str is empty

T3: arithmetic conditions

n -eq n'	n = n'
n -ne n'	n != n'
n -gt n'	n > n'
n -ge n'	$n \geq n'$
n -lt n'	n < n'
n -le n'	$n \leq n'$

2) <u>for loops</u>

2 methods for for loops: * for var in list
                         * for ((i=0; i<x; i++))

```
1) for var in val1 val2 val3; do       | val1
       echo $var                        | val2
   done                                 | val3
2) values="val1 val2 val3"              | val1
   for var in $values; do               | val2
       echo $var                        | val3
   done                                 |
3) values="val1 val2 val3"              | val1 val2 val3
   for var in "$values"; do             |
       echo $var                        |
   done                                 |
4) for var; do                          | prints positional parameters
       echo $var                        | each on their own line
   done                                 |
5) for file in *; do                    | prints all files in working directory
       echo $file                       | each on their own line
   done                                 |
6) break;                               | skips statements in loop
                                        | continue after loop
7) continue;                            | skips statements in loop
                                        | continues with next iteration
8) for var in {1..5}; do                | 1
       echo $var                        | 2
   done                                 | 3
                                        | 4
                                        | 5
9) for var in {1..5..2}; do             | 1
       echo $var                        | 3
   done                                 | 5
```

3) Arrays

* creating arrays:

```
arr=("val1" "val2" "val3")
```

* put all files from working dir in array:

```
arr=(*)
```

DO NOTE USE LS

* adding / changing values:

```
arr[0]="new"
```

* using in for:

```
for val in "${arr[@]}"; do
```

* get amount of values:

```
amount=${#arr[@]}
```

* getting values:

```
val=${arr[0]}
```

4) Getopts

```
while getopts ":ab:c" opt; do
    case $opt in
        a) #verwerk
           ;;
        b) #verwerk; $OPTARG bevat argument
           ;;
        c) #verwerk
           ;;
        \?) echo "syntax: $0 [-a] [-b arg] [-c] args" 1>&2
            exit 1
    esac
done
shift $((OPTIND - 1))
```

→ dubbelpunt in begin v. optstring onderdrukt meldingen v getopts;
: na opt geeft verplicht argument
\?) verwerkt unknown arg
$OPTIND bevat pos v volgende arg; na while bevat dit pos v eerste echte argument.

5) Functions

* 2 ways of declaring:

```
func {                      func () {
     commands;                   commands;
}                           }
```

* calling a function: like a normal command

```
func {              |   func
     echo "func"    |
}                   |
func                |
```

* functions cannot be empty

* functions must be defined before being used

* functions can be called from within other functions.
Note: a function in another function can be used before its definition, as long as the encapsulating function precedes its definition.

6) While loops

* While loops use the same conditions as "if".

* Reading a file using while:

```
while read line
do
     commands;
done < file.
```

7) Arithmetic

* 2 ways of doing arithmetic:

simple: var = $((expr))   advanced: var = $(echo "expr" | bc)

simple supports:

* + : addition	echo $((20+5))	25
* - : subtraction	echo $((20-5))	15
* / : division	echo $((20/5))	4
* * : multiplication	echo $((20*5))	100
* % : modulus	echo $((20%3))	2
* ++ : post-incr.	x=5; x++; echo $x	6
* -- : post-decr.	x=5; x--; echo $x	4
* ** : exponentiation	x=2; y=3; echo $((x**y))	8

- Can only work with integers. $((4/3)) gives 1.
- parameter dereferencing is optional.

bc supports all simple commands, and: (exception: exp)

- fractions: $(echo "3/4" | bc) returns 0,75 (instead of 0)
- scale: $(echo "scale=3; 1/3" | bc) amount of digit after comma. Returns 0.333.
- sqrt: $(echo "sqrt(16)" | bc) returns 4
- exponentiation with ^ : $(echo "2^3" | bc) returns 8
- brackets
- obase, ibase: output and input base. Note: if you change ibase; obase must be defined in hex:

  $(echo "obase=2; 12" | bc) returns 1100

  $(echo "ibase=2; obase=A; 10" | bc) returns 2

Misc

* Using read to ask for input:
```
read -p "Give value:" var
```
* All command line args: $@
* Getting a random line from a file:
```
line = $(shuf -n 1 file)
```
* Using trap to catch all interrupt signals:
```
trap 'commands' 1 2 3 15 20
```
Don't forget to add exit to 'commands'
* Getting stdin if present
```
if [[ -t 0 ]]; then
    # STDIN is empty
    # (file descriptor 0 (input) is a terminal (keyboard)
else
    # file descriptor 0 is not a terminal
    # command is being used in a pipe or with <
    input = $(cat)
fi
```
* Force variable expansion with eval:
```
m=2; n=5; echo {$m..$n}  →  {2..5}
m=2; n=5; eval echo {$m..$n} → 2 3 4 5
```