BASH SCRIPT CHEAT SHEET

BETA EDITION - WORK IN PROGRESS - USE AT YOUR OWN RISK

Extras

Debugging

```
#!/bin/bash                  shell bash script
set -e                     exit if script fail
set -x                          verbose output

export PS4='+($                  add to script --
{BASH_SOURCE}:$               great for debugging
{LINENO}): $                          one line!
{FUNCNAME[0]:+$
{FUNCNAME[0]}(): }'

bashdb                         if all else fails
<bang head on keyboard>           final solution
```

Functions

```
function functionname {
   <commands … can access global vars>
}

passing vars:
function functionname2 {
   echo $1                          #stdout hello
}
functionname2 hello

returning vars:
function functionname3 {
   # eval of last command, return, exit, echo
   # globals or $1 pass trick -
   echo $(( 7 * 6 ))
}

echo $(functionname3)                    #sdtout 42

command substitution:
function countchars {
   cat $1 | wc -c              # $1 from fun. call
}

charsinfile=$( countchars $1 )
# $1 from commandline
echo "file $1 has $charinfile characters."
```

Shell Definitions

```
command1; command2                      do 1 then 2
command1 & command2          1 and do 2 w/o waiting
:                                     empty command
(command)                    run command in subshell
```

I/O

```
command > file              command output to file
command < file              command stdin from file
command >> file              append cmd stdout file
command1 | command2          out c1 to in c2 (pipe)
0                                    (stream) stdin
1                                   (stream) stdout
2                                   (stream) stderr
stream>file                          stream to file
stream>>file                  stream to append file
stream1>&stream2                 merge out s1 w/ s2
stream<&stream2                   merge in s1 w/ s2
```

Flow Control

If

```
if [ expr1 ]       # [ expr1 ] can be a command
then                                  # or (( ))
  <TRUE commands for expr1>
elif [ expr2 ]
  <TRUE commands for expr2>
else
  <FALSE commands for expr1 AND expr2>
fi
```

combo expr - > **[e -a e2]** in bool expr:

```
if [ expr1 ] && [ expr2 ] … or …
if [ expr1 ] || [ expr2 ]
```

shorthand if:

```
[ expr ] && <statement for expr>
((bool =  expr ? val1 : val2 ))       #awesome!
```

For

```
for (( i=1;i<10; i++ ))   # (( ; ; )) inf. loop
do
   <LOOP commands>
done
```

For List

```
for var in $varlist     # no `in list` = `in $@`
do
   <LOOP statements using $var passing
    items from list>
done
```

Case

```
case $var in
   pattern1)   # pattern can be multiple | (or)
   <pattern1 statements>
   ;;
   pattern2)                # wildcards allowed
   <pattern2 statements>
   ;;
   *)
   <not p1 or p2 statements - think default>
   ;;                          # last ;; optional
esac
```

While

```
while [ expr ]
do
   <WHILE statements>
done
```

Until (ignore if ! [understand])

```
until <command>
do
   <UNTIL statements>
done
```

Select

(for menus; mentioned for completeness)

Variables & Examples

Basic Shell Built-in

```
$0                          script filename + path
$(basename "$0")                   script filename
$1 $2 $3 … ${10} …            command line params
$#                               number of params
"$*"                           all cmd line as word
"$@"                             "$*" but separated
shift "$@"                          drop last param
$-                                     only options
$?                             exit status (f or s)
$$                                   script proc id
$!                                  pid of last job
$_                                     last command
!!                               repeat last command
$(fun)                           function return val
$'\000'                                  ASCII char
$'\x00'                                    hex char
$'\n'                          (any esc. char works)
${str}xyz                         appends xyz to str
${#str}                             same as strlen()
${str:pos}                       del after pos in str
${str:pos:len}                 extract part of string
${str#wild}                        remove up to match
${str##wild}                 remove up to last match
${str/wild/rep}                  replace first match
${str//wild/rep}                 replace all matches
```

Operators (Order of Precedence)

```
x++ x--                      post inc and dec (( ))
++y -y                        pre inc and dec (( ))
- +                                       num sign
! ~                        logic and bit neg (( ))
**                                     exp. (( ))
* / %                           multi, div, mod
+ -                                     add, sub
<< >>                   lt and rt shift (( ))
<= >=                        comparison (( ))
== !=                          eq and not eq
&                                       bit AND
^                                       bit XOR
|                                        bit OR
&&                                     logic AND
||                                      log OR
expr?val1:val2                 cond op (( ))
= *= /= %= += -= <<=                 assign ops
>>= &= ^= |=
```

More Arthmetic Examples

```
(( x=x+1 ))                              let "x=x+1"
(( ++x ))                                  let "++x"
:$(( ++x ))                  (ok, but ugh) :$[ ++x ]
$(( ++x ))                       (kills script) BAD!
(( $x+=1 ))                          ($ is ugh) BAD!
y=$(( i+x ))                             Let "y=i+x"
y = $(( i+x ))                         (spaces) BAD!
y=(( i+x ))                  ($ not included) BAD!
```

Note 2 (philosophy):

(( )) for int expr
***like* ((x > 1 && x < 11))**
[[ ]] for str expr [ ] for file expr
"$var" for eval zero cases

Expressions

Numbers

```
[ $num1 -eq $num2 ]                  num1 == num2
[ $num1 -ne $num2 ]                  num1 != num2
[ $num1 -lt $num2 ]                  num1 <  num2
[ $num1 -le $num2 ]                  num1 <= num2
[ $num1 -gt $num2 ]                  num1 >  num2
[ $num1 -ge $num2 ]                  num1 >= num2
```

Strings

```
[[ $str1 = $str2 ]]            str1 equal to str2
[[ $str1 == $str2 ]]           str1 equal to str2
[[ $str1 != $str2 ]]       str1 not equal to str2
[[ -z $str ]]                     str size of zero
[[ -n $str ]]                  str non-zero size
[[ $str ]]                         str non-empty
```

Files

```
[ -r $file ]                             readable
[ -w $file ]                             writable
[ -x $file ]                      execute Access
[ -f $file ]                              ordinary
[ -s $file ]                       filesize > 0
[ -d $file ]                          directory
[ -e $file ]                        file exists
[ -N $file ]                  mod since last read
[ -O $file ]                   cur. user is owner
[ -G $file ]                 cur. user is in group
[ $file1 -nt $file2 ]      file1 newer than file2
[ $file1 -ot $file2 ]      file1 older than file2
[ $file1 -ef $file2 ]         file1 same as file2
```

Boolean Expressions

```
[[ expr1 && expr2 ]]                 expr1 and expr2
~~[ expr1 -a expr2 ]~~          (!rec.) expr1 and expr2
[[ expr1 || expr2 ]]                  expr1 or expr2
~~[ expr1 -o expr2 ]~~          (!rec.) expr1 or expr2
[[ expr1 != expr2 ]]           expr1 not equal expr2
 ! [ expr1 = expr2 ]           expr1 not equal expr2
[[ ! expr ]]                                not expr
```

Matching

```
[[ $var = c* ]]                     pattern matching
[[ $var =~ ^c ]]                      regex matching
```

Patterns & Wildcards

```
*                                    0 or more char
?                                         one char
[XxYyZz]                          1 char from list
[^AEIOU]                        any char not from list
[0-9]                          char from range of char
```

Arithmetic

```
x=0; y=`expr $x+1`                  (old) bourne math
x=5; let x++; let x+=1                (old) bash math
let "i=$i+1"                     let math? Use "" for
                                other lets. ($ not req)
(( e=5 )); j=$(( e++ ))               (new) bash math
i=$(( x + y + z ))                     (new) assign i
echo $(( l*w ))                           (new) echo
```

Note:

[ ] shell compat > [[ ]] does more
'let' shell compat > (( )) & $(( ))
" " is less shell compat, but does more
Trick if arthmetic evals as 0:
((x=$y-1)) || true (if on it's own line)

BASH SCRIPT CHEAT SHEET

BETA EDITION - WORK IN PROGRESS - USE AT YOUR OWN RISK

Process Control Characters

<CTRL-C>	kill job (SIGINT)
<CTRL-D>	EOF or exit shell
<CTRL-L>	clear screen
<CTRL-Z>	process suspend
fg 'procname'	undo suspend
<ALT-DEL>	del word before cursor

More

$PWD	current directory
$OLDPWD	“last” current dir
$PATH	path : delim.
$CDPATH	cd path : delim.
$REPLY	last read iff new read empty
$IFS	int field separator
$EDITOR	default editor
$UID	user UID
$EUID	effective UID
$COLUMNS	lines in term output
$FUNCNAME	name of cur. Function
$HOME	home dir of cur. User
$LINENO	line number in script
$SECONDS	seconds script running
$HOSTNAME	computer hostname

Prompt

$PS1	command line
$PS2	input
$PS3	select loop
$PS4	debug

Environmental

$HOME	home dir
$PWD	working directory
$SHELL	shell interpreter
$LOGNAME	user login name
$TERM	terminal
$USER	similar to logname
$LANGUAGE	for translation
$_	last command

Display Environmental & Shell Variables

printenv	list environmental var.
declare -p	(same as above)
env	(same as above)
(set -o posix ; set)	list shell, local & env. var.
declare -xp	(same as above)
–	–
set	env., local & shell vars + shell functs.

Arrays

a[0]=Value	decl. element
b=(“one” “two” “3”)	decl. array
${a[*]}	items in array
${!a[*]}	indexes in array
${#a[*]}	num of items in array
${#a[0]}	len of index 0 in array

NOTES: