Bash cheat-sheet [LATEST]

```
# Integer Base conversion

# converts hex 33 to decimal
echo 'ibase=16; 33' | bc

# converts hex FEF33D9v to octal and binary
echo 'ibase=16; obase=8; FEF33D9' | bc
echo 'ibase=16; obase=2; FEF33D9' | bc
#
var=$(echo 'ibase=16; FEF33D9' | bc)
echo $var     O/P: 267334617
```

```
# if / test

# numeric compare:      ( -lt < | -gt > | -ne != | -eq ==)
e.g. if ((x>=-100)) && ((1<y<=100)) && ((y!=0)); then ..

# string compare:       ( != | = | < | >)
e.g. if [[ $achar = 'Y' || $achar = 'y' ]];      then

# And OR:               ( &&  ||)

# shell math #Leading $ not required inside $(( ... ))
VAL=((VAL + 1)) <<< This does not work
VAL=$((VAL + 1))
(( VAL++ )) OR (( VAL += 1 )) OR (( VAL = VAL + 1 ))
```

```
# math in bash, (( )) works like a command and will have a return code.
# $(( )) works like variable substitution.

$ a="2"; b="3".
$ ((c=a*b+2))
$ echo $c   O/P: 8

# in variable substitution, the variable is substituted with its
value before the expression is evaluated.
$ c=$(( a*b + 2 ))
$ echo $c          O/P: 8
$ echo $((a>b))    O/P: 0
$ echo $((a<b))    O/P: 1

$ echo $((a==2))   O/P: 1 if $((a>b)); then echo "Good"; fi # does't work
# this works
((c = a*2 ))       O/P: 4

# this does not
(($c = a*2 ))

# 'x' is the value of "a+2"
x=$((a+2))

# 'smaller' is true if a is smaller than 10, and false otherwise
smaller=$((a<10)) # Note that smaller=1 also means it is true
if ((smaller=1)); then echo "Good"; fi  # works
```

```
# floating point math

# quick calculation. This works only when
one operand is float
$ echo "1.2*8" | bc    O/P: 9.6
$ echo "219/5" | bc    O/P: 43

# use a bash variable value
$ pi="3.141592"; r="5"
$ area=$(echo "$pi*$r*$r" | bc)
$ echo $area    O/P: 78.539800

# correct way to do it this way
x=219; y=5
var=$(bc<<< "scale=3; $x/$y")
echo $var  O/P: 43.800

s="5+50*3/20 + (19*2)/7"
var=$(bc<<< "scale=3; $s")
echo $var  O/P: 17.928.

# no rounding, see the tricks below
s="5+50*3/20 + (19*2)/7"
var=$(bc<<< "scale=5; $s")
echo $(round $(math "$s") 3)  O/P: 17.929

Ref next block for round() and math()
function
```

```
# round to a certain decimal
round() {
     printf "%.${2:-0}f" "$1"
 }

math () {
    echo "$*" | bc -l
}
a="3.141592*5*5"
echo "Pi, to five decimal places, is $(round $(math "4*a(1)") 5)"
O/P: Pi, to five decimal places, is 3.14159
```

```
# Typed variables
typeset|declare| Description
-------------------------------------
-a     | -a   | Normal indexed array
-A     | -A   | Associative array
-r     | -r   | Make the variable read only
-u     | -u   | Conv. on assignment to uppercase
-l     | -l   | Conv. on assignment to lowercase
e.g. typeset -u x
     x="abcd"
     echo $x   O/P: ABCD
```

```
# special variables
$SECONDS  # seconds since shell started
$RANDOM   # a random number
$LINENO   # Current line number of the script

# substring pattern extraction / substitution
${astrvar:offset:length} # length chars of $astrvar

# Remainder of $astrvar starting at offset, 1st char is 1
${astrvar:offset}
```

```
# pattern matching places

?(pattern)    -   Zero or one instances of pattern
*(pattern)    -   Zero or more instances of pattern
+(pattern)    -   One or more instances of pattern
@(pattern)    -   Exactly one instance of pattern
!(pattern)    -   Anything not matching pattern
~(E)pattern   -   pattern is an extended regular expression (egrep)
~(G)pattern   -   pattern is an basic regular expression (grep)
if [[ ${STRING} = A@(da|to)m ]] ⇐ Match Adam or Atom

# Watch the syntax , string comparison uses =

# matches abc21, abc91 or abc1
ls abc?(2|9)1
```

```
# more pattern extractions
${var#pattern}          -  Delete first match from left, return rest
${var##pattern}         -  Delete all matches from left, return rest
${var%pattern}          -  Delete last match from right, return rest
${var%%pattern}         -  Delete all matches from right, return rest
${var/pattern/string}   -  Replace longest match of first occurrence
${var//pattern/string}  -  Replace longest match of all occurrences
${var/#pattern/string}  -  Replace longest match from beginning
${var/%pattern/string}  -  Replace longest match from end
theaddr=192.168.1.25 ⇐ Assign an address (example assumes class C)
network=${theaddr%.*} ⇐ Delete dot and last octet
thehost=${theaddr##*.} ⇐ Delete all octets followed by dots
echo ${password//~(E)./X} ⇐ Substitute X for every character in $password
```

```
# variable substitution
```

Expression	VAR defined return	VAR undefined return	VAR undefined set VAR to
${VAR:-string}	$VAR	string	
${VAR:=string}	$VAR	string	string
${VAR:?string}	$VAR	string to stderr, exit	
${VAR:+string}	string	NULL	

```
# '#' at the start gets the number of characters
$ myvar="bipp bopp"
$ echo ${#myvar}   O/P: 9

# remove exactly three '0' from the beginning
$ val="000123"
$ echo ${val#000}  O/P: 123

# "*" means any number of any characters
# remove anything, then a zero, as little as possible
$ echo ${val#*0}   O/P: 00123

# remove anything, then a zero, as much as possible
$ echo ${val##*0}  O/P: 123

# "?" means any single character,remove 3 characters
$ echo ${val#???}  O/P: 123
```

```
# remove '.jpg' at the end using "`%.jpg`",
# add '.png' to the end instead, store in outname
$ filename="mypicture.jpg"
$ outname=${filename%.jpg}.png

# Let's say out="Submitted batch job 12345"
# remove everything from beginning up to the last
space, jobid will be 12345
$ jobid=${out##* }

# search for "input", replace with "output", save in
outfile
#  to replace all matches in the string, begin with
two slashes ("//") instead of one.
# use "*" and "?" to match any sequence and any one
character
$ inputfile="mydata_987_input_123.file"
$ outfile=${inputfile/input/output}
$ echo $outfile  O/P: mydata_987_output_123.file
```

```
# special params, ($* and $@ are same except below)
$#, $!, $$, $?, $*, $@
"$@" = $1 $2 $3
"$*" = "$1 $2 $3 .."

# arrays
typeset -A array  # array is associative
typeset -a array  # array is regular indexed from 0

[root@aap02 ~]# typeset -A array1
[root@aap02 ~]# array1[apple]=green
[root@aap02 ~]# array1[banana]=yellow
[root@aap02 ~]# array1[orange]=orange

[root@aap02 ~]# echo ${array1[@]}
green yellow orange

[root@aap02 ~]# echo ${!array1[@]}
apple banana orange

[root@aap02 ~]# unset array1[banana]
[root@aap02 ~]# echo ${!array1[@]}
apple orange
[root@aap02 ~]# echo ${array1[@]} O/P: green orange
```

```
# regular array
[root@aap02 ~]# arr=(alpha beta gamma delta)
[root@aap02 ~]# echo ${arr[*]}  O/P: alpha beta gamma delta

[root@aap02 ~]# echo ${arr[@]}  O/P:alpha beta gamma delta
[root@aap02 ~]# echo ${#arr[@]} O/P: 4

[root@aap02 ~]# echo ${!arr[@]} O/P: 0 1 2 3
[root@aap02 ~]# arr+=( theta )

[root@aap02 ~]# echo ${arr[@]} O/P: alpha beta gamma delta theta

[root@aap02 ~]# echo ${#arr[@]} O/P: 5

[root@aap02 ~]# echo ${!arr[@]} O/P: 0 1 2 3 4
[root@aap02 ~]# unset arr[2]
[root@aap02 ~]# echo ${arr[@]}  O/P: alpha beta delta theta
[root@aap02 ~]# echo ${!arr[@]} O/P: 0 1 3 4
```

```
# example array operations
$ arr=(a b)  # arr is (a b)
$ arr+=(c)   # add a value to the end: arr is (a b c)
$ arr=(start ${arr[@]} end)
$ echo ${arr[@]}  O/P: start a b c end

# example array operations
$ i=3
$ arr=(zero one two three four)
$ echo ${arr[$i]}   # simpler way, O/P: three
$ i=2
$ echo ${arr[${i}]} # formal way, using '${ }' O/P: two
```

```
# example array operations
$ arr=( )               # empty array
$ arr[4]="first"        # set three elements
$ arr[12]="second"
$ arr[7]="third"

$ echo ${arr[@]}  # a list of elements, O/P: first third
second

$ echo ${#arr[@]}   # three elements O/P: 3
$ echo ${!arr[@]}   # element indexes O/P: 4 7 12
$ echo ${arr[@]:1:2} # slicing element O/P: first second
```

```
# example array operations
slurmfiles=(./*slurm)  # watch the syntax
echo “slurm files: “ ${slurmfiles[@]}

# example array operations
$ cmdarr=( $(sbatch myjob.slurm) )
$ echo ${cmdarr[@]}
Submitted batch job 12345
$ echo ${cmdarr[3]}. O/P: 12345

# built-in “mapfile”, also called “readarray” reads input
line by line (or word by word) and fill it into an array.
mapfile takes input from stdin(you can't use pipes for
deep technical reasons). You can set what to split on
(default is newline), how many lines to read, whether to
skip lines at the start and more. Default separator is
‘\n’. ‘-t’ removes new line, ‘-d’ changes the separator.

# we get Macbeth line by line in "maclines"
$ mapfile -t <macbeth.txt maclines

# How many lines?
$ echo ${#maclines[@]}  O/P: 4828
# Seems right:
$ wc -l Macbeth.txt     O/P: 4828
```

```
# mapfile will split on every space. That is, with two
spaces between words, that will be two splits. If we
split the string “hi Bob” with two spaces between the
words, we’d get the elements “hi”, " " and “Bob”.
mapfile will treat the first space as the end of “hi”,
but the second space as the end of an empty element. If
you want words in a text into an array, it’s better to
use command capture to create an array instead.

# Let's try to get the words:
$ mapfile -d " " -t <macbeth.txt macwords
$ echo ${#macwords[@]}.    O/P: 25332

# Something is wrong:
$ wc -w <Macbeth.txt    O/P: 18101

# we have extra empty "words"
$ echo ${macwords[13]}

# Let's use command capture instead:
$ macwords=($(cat Macbeth.txt))
$ echo ${#macwords[@]}.  O/P: 18101
```

```
# conditionals
$ a=1
$ mystr="hello"

# numerical comparisons
$ test $a -eq 1     # true; a is equal to 1
$ test $a -lt 0     # false, a is not less than 0

# string tests
$ test "$mystr" == "hello"  # true.
$ test -z "$mystr"          # false – mystring is > 0

# file tests
$ test -e myfile.txt        # myfile.txt exists
$ test -f myfile.txt        # myfile.txt is a normal file
$ test -d myfile.txt        # myfile.txt is a directory

# examples, these two lines do the same thing
$ test $s == "hello"
$ [ $s == "hello" ]
if [ "1" -lt "2" ]; then echo "yes"; fi

# find all .fasta files in all subdirectories, copy them
to fastafiles/
for f in $(find . -name "*.fasta")
do
    cp $f fastafiles/
done

# use seq to generate a sequence
for i in $(seq 5)  # $(seq 1 2 100)  >> 1 3 5 .. 99
do
   echo $i
done
```

```
# seq is useful when you want the sequence to be variable
iter=100

# this doesn't work:
for i in {1..${iter}}

# this works
for i in $(seq ${iter})
# While is most often used when you want to do something
on each line of a text file. Here we read each line one
by one, then reverse each line:
while read -r line
do
    echo $line|rev
done <macbeth.txt

# another way..
$ cat macbeth.txt| while read -r line; do echo $line|rev;
done |less
OR
cat macbeth.txt| rev| less
OR
rev macbeth.txt | less
```

```
# trap signals

trap 'command;command' 1 2 3 15
trap 'rm tmp*; exit 1' 1 2 15
trap 'echo "CRTL + C"' INT
trap 'echo "CRTL + Z"' QUIT
trap 'echo ..' ERR
trap -ERR # to unset
```

```
# fake signal ERR and DEBUG

function error_handler () {

  printf "Failed.\n"
  echo "ERROR: Command failed. Exiting now." >&2
}

# register your error handling function for the ERR
trap error_handler ERR
set -e                # Tell shell to exit on failure

printf "Running my cmd..." my_cmd
printf "Done.\n"

trap -ERR # to unset
```

```
# waiting for background jobs to finish
my_command() {
  sleep 3
  echo "I am done $SECONDS"
}

other_commands() {
  sleep 5
  echo "I have finished my tasks $SECONDS"
}

my_command &
SECONDS=0
big_pid=$!
echo "one $!"
other_commands &
wait $big_pid && echo "Foo $SECONDS"
wait
echo "end of it all $SECONDS"

O/P:
one 760
foo 3
I have finished my tasks 5
end of it all 5
```

```
# misc

# life of a process
fork()  >> wait() >> exec() >> exit()

# uid /gid /euid /eguid /setuid /setgid
# umask 022: 777-022 = 755 (dir); 666-022 = 644 (file)

# regex
[^A-Z] # lines not containing chars A thru Z
\(..\) # /(love\)able  \1er = lover

# In vi
:s/\(square\) and \(fair\)/\2 and \1  => fair and square

x\{m\}  # repetition of char x, m times
x\{m,\} # at least m times
x\{m,n\} # at least m times and at most n times

# repition of one of the characters between a-z , by 9
times
grep '[a-z]\{9\}

# matches abc21, abc91 or abc1
ls abc?(2|9)1
```

```
# eval evaluates the command, performs all shell
substitutions and then executes the command

set a b c d
echo The last argument is \$#
The last argument is $4
eval echo The last argument is \$#
The last argument is d
```

```
# shift, doit.sh
#!/bin/bash

while [ $# -gt 0 ]
do
  echo $*
  shift
done
O/P:
doit.sh a b c d
a b c d
b c d
c d
d
```

```
Misc:
x="/a/b/c/d/file1.txt"
echo $(basename $x) O/P: file1.txt
echo $(dirname $x)  O/P: /a/b/c/d
```

```
# Capture more than one variable in a loop
my_cmd 2> /dev/null | while read A B C REST
do
  echo $C
done

OR

cat afile | while read A B C REST
do
  echo $C
done
```

```
# local variables, acript.sh
#!/bin/bash

function afunc {
  local var1
  echo in function: $0 $1 $2
  var1="in function"
  echo var1: $var1
}

var1="outside function"
echo var1: $var1
echo $0: $1 $2
afunc funcarg1 funcarg2
echo var1: $var1
echo $0: $1 $2

O/P:
./ascript.sh arg1 arg2
var1: outside function
./ascript.sh: arg1 arg2
in function: ./ascript.sh funcarg1 funcarg2
var1: in function
var1: outside function
./ascript.sh: arg1 arg2
++++++++++++++++++++++++++++++
# getopts examples
#!/bin/bash

while getopts xy options
do
  case $options in
    x) echo "you entered -x as an option" ;;
    y) echo "you entered -y as an option" ;;
  esac
done

O/P:
./getopt1.sh -x
you entered -x as an option
[root@aap02 ~]# ./getopt1.sh -y
you entered -y as an option
```

```
#!/bin/bash

while getopts :xy options
do
  case $options in
    x) echo "you entered -x as an option" ;;
    y) echo "you entered -y as an option" ;;
    \?) echo $OPTARG is not a valid option 1>&2;;
  esac
done

O/P:
./getopt2.sh -c
c is not a valid option


#!/bin/bash

while getopts :x: options
do
  case $options in
    x) echo $OPTARG is name of the argument ;;
    :) echo "enter argument after -x" >&2 ;;
    \?) echo $OPTARG is not a valid option 1>&2;;
  esac
  echo $OPTIND
done

O/P:
[root@aap02 ~]# ./getopt4.sh -x filex
filex is name of the argument
3
[root@aap02 ~]# ./getopt4.sh -x
enter argument after -x
2
[root@aap02 ~]# ./getopt4.sh -d
d is not a valid option
2
```

```
# cut || head/tail || tr || sort || uniq || paste crap

cut -d " " -f 2,3          # cuts 2nd & 3rd field d is delimeter
cut -b 1,2,3               # cuts 1st, 2nd, 3rd bytes
cut -b 1-  || cut -b -3    # Other examples
cut -c 2,3,5 || cut -c 2-5 # other examples

uniq <file name>  # display only uniq lines
uniq -ic          # display uniq lines and their count, ignore case
uniq -d           # display uniq lines only
uniq -D           # display duplicate lines as they appear

sort <fiile_name> # sorts file alphabetically
sort -k 3         # sorts fiile based on 3rd field
sort -n           # sorts numerically
sort -r           # sorts in reverse order
sort -k1n -k3rn file1 # sort on 1st col then in reverse on 3rd col, all numbers
sort -t $'\t' . .     # tab delimited text
sort -r -n -k 2 -t $'\t' file1 # -r reverse, -n numeric, -k on field 2, -t is
tab

# converts upper from file1 to lower and writes to file2
tr [:upper:] [:lower:] < file1 > file2 tr [a-z] [A-Z]
tr '()' '[]'     # replace all () with []
tr -d [a-z]      # delete all lowercase letters

# replace multi \n with single and write to file2 || replace space example
tr -s '\n' <file1 >file2 || tr -s ' '
```

```
paste -s    #all lines into one line
paste - - - # 3 consecutive lines
pasted together to make one line

paste file1 file2 # paste
corresponding lines of both files
side by side

paste -d '_' file1 file2 #u know

paste -d '%|' file1 file2 file1
# uses % to paste file1 & file2 then
resulting line use | to paste again
file1
```

```
# grep / sed / awk
grep '\<north' filename       # lines beginning with north
grep '\<north\>' filename     # lines containing the word north

# Find anything beginning with "H" then zero or more alphabetic characters (not numbers, spaces or punctuation)
grep "H[[:alpha:]]*" Hamlet.txt

# \(.\)\1 means any one character, followed by the same character
grep "\(.\)\1" Hamlet.txt

sed '1,3d' myfile                           # print all lines except 1 to 3
sed -n '/[Jj]ohn/p' myfile                  # print lines with John and john
sed '/Tom/!d' myfile                        # delete lines not containing Tom
sed '$d' myfile                             # delete last line
sed 's/west/north/g' myfile                 # substitute west with north
sed -n 's/^north/west/p'                    # lines beginning with north will be substituted by west and print
sed 's/\(Mar\)got/\1inne/p'                 # substitute Margot with Marinne
sed 's#3#8#g'                               # '#' use instead of '/'
sed -n '/west/,/east/p'                     # print lines between west and east
sed -n '5,/^northeast/p'                    # print lines between lineno 5 and lines starting with northeast
sed -e '1,3d' -e 's/north/west/'            # delete lines 1 to 3 and then substitute north with west
sed '1,3y/abcd../ABCD../'                   # replace lowercase with uppercase between lines 1 and 3
sed '5q'                                    # quit after printing 5 lines
sed '/Lewis/{s/Lewis/John/;q;}'             # pattern Lewis is matched, substitute with John, print and quit
sed -e '/north/h' -e '/$/G' myfile          # add lines to holding buffer to the line after the end of all lines
sed -e '/Patricia/h' -e '/Margaret/x' myfile   # find lines with Patricia, put it in holding buff,
                                            find lines with Margaret and replace whole line

sed -f <script_file> datafile
<script_file>
/western/,/southeast/ {      # in the range of western to southwest
   /^ *$/d                   # delete all blank lines
   /Susan/{h;d;}             # find lines with Susan, put it in holding buffer, then delete the lines from output
}
/Ann/g                       # find lines with Ann and replace that line with the line in holding buffer
s/TB \(Savage\)/Thomas \1/   # Replace "TB Savage" with "Thomas Savage"

awk '{printf 'The name is: %-15s ID is %8d", $1, $2}' filename            # formatted o/p
awk '$1 ~ /[Bb]ill/' filename                                             # matching a field
awk '$1 !~ /ly$/' filename                                                # field $1 should not have ly at the end
awk -F":" -f <script_file> filename                                       # using a script_file like sed
awk '$5 ~ /\.[7-9]+/' filename                                            # ..
awk '$3 * $4 > 4000' filename                                             # ..
awk '{max=($1 > $2)?$1:$2; print max}' filename                           #..
awk '$2=="CT"{$1="Connecticut"; print}' filename. | '$3 ~ /^Susan/ { print "Percent: $6 + .2"}' filename  # ..
awk '$2 > 5 && $2 <= 15' filename  |  '$6 > .9 {print $1}'                                               # ..
awk '/Marry/{count++} END{ print "Marry was ".count." times"}' filename                                   # ..
awk 'BEGIN {while("ls"|getline) print}'                                                                  # ..
awk 'BEGIN {while (getline<"/etc/passwd" > 0) lc++; print lc}'                                           # ..
awk -F":" '{id[NR]=$1} END{for(x=1; x<=NR; x++) print id[x]}' /etc/passwd                                # ..
## other examples
awk '{score=($2+$3+$4)/3;}
    {if (score >= 80) {print $0" : A"}
     else if (score >= 60  && score < 80) { print $0" : B"}
     else if (score >= 50  && score < 60) { print $0" : C"}
     else { print $0" : FAIL" }}'
Ref: https://stackoverflow.com/questions/26634978/how-to-use-readarray-in-bash-to-read-lines-from-a-file-into-a-2d-array
```

```
awk '{ if ( $1 ~ /^extended/ ) { next; }  # refer my perf.ksh code
       else if ( $1 ~ /^(Mon|Tue|Wed|Thu|Fri|Sat|Sun)/ ) {
currdate=$0; next; }
       }
       else if ( $11 ~ /:/ ) { next; }
       else if ( max[$11] < $8 ) {
            max[$11] = $8;
            tot[$11] += $8;
            count[$11]++;
            diskdate[$11] = currdate;
      }
     END {
            for (i in tot) {
            avg = tot[i] / count[i];
            print "AVG asvc time for ", i is:", avg, "ms"
      }
     }'
```

```
script_file:
BEGIN { print " .. " }
{ print $1 }
{ total += $7 }
/north/{ count++ }
END { print "..";
      Print "Count=" count
}

awk '{if ( $6 > 50 ) { count++; print $3 } \
     else { x+5; print $5 }}' filename

awk '{if ( .. ) A++
      else ( ..) B++
      else C++
     }
     END { print ".." }
```

Code Base:

```
# Parse a iostat -tkx 1 10 output  file to find IOPS, BDW, LAT for disk sda
#!/bin/bash

function parse_io() {
    awk '{  if ($1 ~ /sda/) {
              iops[sda]  = iops[sda] + $2 + $8;
               bdw[sda]  = bdw[sda] + $3 + $9;
               lat[sda]  = lat[sda] + $6 + $12;
               count++;
            }
            else {
               next;
            }
         }

         END {
            print "Average IO stat for sda for " count " times: ", iops[sda]/count, bdw[sda]/count,
lat[sda]/count
         }
        ' $1
}

## main starts here
###################

while getopts :f: options
do
    case $options in
        f) echo "Input file name is: $OPTARG"
           FILE=$OPTARG ;;
           #echo $FILE ;;
        :) echo "Input file name is mandatory" ;;
        \?) echo "$OPTARG is not a valid option" ;;
    esac
done

parse_io $FILE

./iostat-parse.sh -f iostat-out.txt
Input file name is: iostat-out.txt
Average IO stat for sda for 32 times:  3126.24 799095 1.3775
```

```
# Coprocess in ksh

#!/bin/ksh

####################
# script: calculator
####################

cat << EOF
*******************************
Welcome to my calculator program
*******************************
EOF

bc |&

while true
do
    print "Select a letter"
    cat << EOF
      a) +
      s) -
      m) *
EOF

    read op
    case $op in
      a) math="+";;
      s) math="-";;
      m) math="*";;
      *) echo "Bad operator"
         continue;;
    esac
```

```
# Conversion between different base in bash
ibase=10; obase=10              # set up defaults
usage() {
  echo "Usage: $(basename $0) -i base -o base value" 1>&2
  echo "  where base can be 2, 8, 10 or 16." 1>&2
  exit 1
}
while getopts "i:o:" value ; do
  case "$value" in
    i) ibase=$OPTARG
       (( ibase == 2 || ibase == 8 || ibase == 10 ||
          ibase == 16 )) || usage
       ;;
    o) obase=$OPTARG
       (( obase == 2 || obase == 8 || obase == 10 ||
          obase == 16 )) || usage
       ;;
    *) usage ;;
  esac
done
shift $(( OPTIND - 1 ))

echo Converting $1 from base-$ibase to base-$obase\:
echo "obase=$obase; ibase=$ibase; $1" | bc
exit 0

O/P:
$ bconvert.sh -i 16 33
Converting 33 from base-16 to base-10:
51

$ bconvert.sh -i 16 -o 2  33
Converting 33 from base-16 to base-2:
110011
```

```
    print -p scale=3
    print "Please enter two numbers: "
    read num1 num2
    print -p "$num1 $math $num2"
    read -p result
    print $result

    print -n "Continue? y/n? "
    read answer
    case $answer in
      [Nn]*) break;;
      *) continue;;
    esac
done

print "Good bye"
```

```
$ bconvert.sh -i 2 -o 16 110011
Converting 110011 from base-2 to base-16:
33
```

disk report from prometheus endpoint: disk_weekly_avg_alerts.sh

```
#!/bin/bash

PROM_URL="http://rh94-6:9090/api/v1/query_range"
OUTPUT_FILE="weekly_disk_avg.txt"

# Hosts monitored by node_exporter
HOSTS=("rh94-1" "rh94-2" "rh94-3" "rh94-4" "rh94-5" "msirtx4050")

# Query range: last 7 days
START=$(date -d "7 days ago" +%s)
END=$(date +%s)
STEP="3600s"  # 1-hour resolution

echo "Weekly Average Disk Utilization (%) with Alerts" > $OUTPUT_FILE
echo "Generated: $(date)" >> $OUTPUT_FILE
echo "-------------------------------------------------------------" >> $OUTPUT_FILE
echo "" >> $OUTPUT_FILE

for HOST in "${HOSTS[@]}"; do

    QUERY="100 * (1 -
node_filesystem_avail_bytes{instance=\"${HOST}:9100\",fstype!=\"tmpfs\",fstype!=\"overlay\",mountpoint=\"/\"} /
node_filesystem_size_bytes{instance=\"${HOST}:9100\",fstype!=\"tmpfs\",fstype!=\"overlay\",mountpoint=\"/\"})"

    # Query Prometheus
    RESPONSE=$(curl -sG $PROM_URL \
        --data-urlencode "query=$QUERY" \
        --data-urlencode "start=$START" \
        --data-urlencode "end=$END" \
        --data-urlencode "step=$STEP")

    # Extract numerical values
    VALUES=($(echo "$RESPONSE" | jq -r '.data.result[0].values[][1]'))

    SUM=0
    COUNT=0

    for VAL in "${VALUES[@]}"; do
        SUM=$(echo "$SUM + $VAL" | bc)
        COUNT=$((COUNT + 1))
    done

    if [ $COUNT -gt 0 ]; then
        AVG=$(echo "scale=2; $SUM / $COUNT" | bc)
    else
        AVG="N/A"
    fi

    # ALERT threshold
    ALERT_THRESHOLD=80.00
    ALERT_MSG=""

    if [ "$AVG" != "N/A" ]; then
        COMPARE=$(echo "$AVG > $ALERT_THRESHOLD" | bc)
        if [ "$COMPARE" -eq 1 ]; then
            ALERT_MSG=" *** ALERT: Disk usage > 80%! ***"
        fi
    fi
```

```
    echo "$HOST average disk usage: ${AVG}%${ALERT_MSG}" >> $OUTPUT_FILE
done

echo "" >> $OUTPUT_FILE
echo "Report saved to $OUTPUT_FILE"
```

cat weekly_disk_avg.txt

```
Weekly Average Disk Utilization (%) with Alerts
Generated: Sun Dec  7 08:55:30 PM EST 2025
------------------------------------------------------------
rh94-1 average disk usage: 18.32%
rh94-2 average disk usage: 17.51%
rh94-3 average disk usage: 17.03%
rh94-4 average disk usage: 17.04%
rh94-5 average disk usage: 16.99%
msirtx4050 average disk usage: 38.41%
```

```
# This stores the output of running "sbatch" in out

out=$(sbatch my_job.slurm)
# just capture the job ID - note the space at the end
jobid=${out#Submitted batch job }

# another way:remove everything from beginning up to the last
space
jobid=${out##* }


# some error checking to avoid bad things to happen if you
forget the parameter or misspell the directory

#!/bin/bash
#SBATCH --partition short
#SBATCH --time 0-1
#SBATCH --mem 10G
#SBATCH --cpus-per-task 8

# get our directory as a parameter

# First see if there is any input: is $# less than 1?
if [ "$#" -lt 1 ]
then
    echo "give a directory with video files"
    exit 1
fi

# Did we get a valid directory?
# if not (!) a directory (-d) then...
if [ ! -d "$1" ]
then
    echo "$1 is not a valid directory"
    exit 1
fi

# we're good, go to the directory and convert the
# images (numbered in sequence) to video
cd $1
ffmpeg -framerate 1/10 -i *%03d.png -c:v libx264 -r 30 -
pix_fmt yuv420p out.mp4
```

```
# code example, spinning pipe and floating point
math
cat my_cmd.sh
for i in $(seq 5)
do
  echo "Hello: $i"
  sleep 1
done

## cat aa.sh
typeset -i start_time=$SECONDS
typeset -i end_time=0
typeset -i total_time=0
typeset -i count=0
typeset -ir ITERATIONS=25
typeset -F 3 average_time
typeset -F ftemp # does not work for bash, correct
it!!

printf "."

while (( count < ITERATIONS ))
do
  case $(( count % 4 )) in
   0) printf "\b|" ;;
   1) printf "\b/" ;;
   2) printf "\b-" ;;
   3) printf "\b\\" ;;
  esac

  /root/my_cmd.sh > /dev/null 2>&1
  (( count++ ))
done

printf "\b"

total_time=$(( SECONDS - start_time ))
ftemp=$total_time
average_time=$(( ftemp / ITERATIONS ))

echo Iterations: $ITERATIONS
echo Total time: $total_time seconds
echo Average time: $average_time seconds

O/P:
[root@aap02 ~]# ./aa.sh
Iterations: 25
Total time: 125 seconds
Average time: seconds
```