BASH

Input and Output

Reading and printing

```
read var
read -a arr # read as array, splitted to multiple elements based on the space

echo hi # print to stdout/screen

echo hi > file # redirect stdout to file
echo error 2> errorlog # redirect stderr to file
```

Conditional

```
test expression
    e.g: test -e file
[ exprn ]
    e.g: [ -e file ]
[[ exprn ]]
    e.g: [[ $ver == 5.*]]
(( exprn ))
    e.g: (( $v ** 2 > 10 ))
command
    e.g: wc -l file
pipeline
    e.g: who|grep “joy” > /dev/null
! for negation
    e.g: ! [ $a = $b ] # note there is a space after !
```

Conditional Expressions

Unary file comparisons (test)

```
-e file Check if file exists
-d file Check if file exists and is a directory
-f file Check if file exists and is a file
-r file Check if file exists and is readable
-s file Check if file exists and is not empty
-w file Check if file exists and is writable
-x file Check if file exists and is executable
-O file Check if file exists and is owned by current user
-G file Check if file exists and default group is same as that of current user
```

String comparison (test)

expression	description
`$str1 = $str2`	Check if str1 is same as str2
`$str1 != $str2`	Check if str1 is not same as str2
`$str1 < $str2`	Check if str1 is less than str2
`$str1 > $str2`	Check if str1 is greater than str2
`-n $str2`	Check if str1 has length greater than zero
`-z $str2`	Check if str1 has length of zero

Arithmetic comparison

expression	description
`$n1 -eq $n2`	Check if n1 is equal to n2
`$n1 -ge $n2`	Check if n1 is greater than or equal to n2
`$n1 -gt $n2`	Check if n1 is greater than n2
`$n1 -le $n2`	Check if n1 is less than or equal to n2
`$n1 -lt $n2`	Check if n1 is less than n2
`$n1 -ne $n2`	Check if n1 is not equal to n2

Conditional execution

if-elif-else

```
if condition; then
    commands
elif condition; then
    commands
else
    commands
fi
```

Case statement

```
case $var in
    op1)
        commandset1;;
    op2 | op3)
        commandset2;;
    op4 | op5 | op6)
        commandset3;;
    *)
        commandset4;;
esac
```

Loop

for do loop

```
for var in list; do
    commands
done

for (( i = 0; i < 10; i++ ));    do
    echo $i
done
```

while loop

```
while condition; do
    commands
done
```

until do loop

```
until condition; do
    commands
done
```

Select loop

```
echo select a middle one
select i in {1..10}; do
    case $i in
        1 | 2 | 3)
            echo you picked a small one;;
        8 | 9 | 10)
            echo you picked a big one;;
        4 | 5 | 6 | 7)
            echo you picked the right one
        break;;
    esac
done
echo selection completed with $i
```

Functions

Definition

```
myfunc() {
    commands
}

function myfunc() {
    commands
}
```

Call

```
myfunc
```

Reading a file line by line

```
while read line; do
    command1
done < file
```

Grep

Options

Option	Description
-E, --extended-regexp	PATTERNS are extended regular expressions
-F, --fixed-strings	PATTERNS are strings
-G, --basic-regexp	PATTERNS are basic regular expressions
-P, --perl-regexp	PATTERNS are Perl regular expressions
-e, --regexp=PATTERNS	use PATTERNS for matching
-f, --file=FILE	take PATTERNS from FILE
-i, --ignore-case	ignore case distinctions in patterns and data

-v, --invert-match	select non-matching lines
-m, --max-count=NUM	stop after NUM selected lines
-n, --line-number	print line number with output lines
-H, --with-filename	print file name with output lines
-o, --only-matching	show only nonempty parts of lines that match
-r, --recursive	like --directories=recurse
-L, --files-without-match	print only names of FILEs with no selected lines
-l, --files-with-matches	print only names of FILEs with selected lines
-c, --count	print only a count of selected lines per FILE

Regex

Special characters (BRE & ERE)

	Description
.	Any single character except null or newline
*	Zero or more of the preceding character / expression
[ ]	Any of the enclosed characters; hyphen (-) indicates character range
^	Anchor for beginning of line or negation of enclosed characters
$	Anchor for end of line
\	Escape special characters

Special characters (BRE)

	Description
{n,m}	Range of occurances of preceding pattern at least n and utmost m times
( )	Grouping of regular expressions

Special characters (ERE)

	Description
{n,m}	Range of occurances of preceding pattern at least n and utmost m times
( )	Grouping of regular expressions
+	One or more of preceding character / expression
?	Zero or one of preceding character / expression
\	

Character classes

	description
[[:print:]]	Printable
[[:blank:]]	Space / Tab
[[:alnum:]]	Alphanumeric
[[:space:]]	Whitespace
[[:alpha:]]	Alphabetic
[[:punct:]]	Punctuation
[[:lower:]]	Lower case
[[:xdigit:]]	Hexadecimal
[[:upper:]]	Upper case

[[:graph:]]	Non-space
[[:digit:]]	Decimal digits
[[:cntrl:]]	Control characters

Backreferences

```
\1 through \9
\n matches whatever was matched by nth earlier paranthesized subexpression
A line with two occurances of hello will be matched using: \(hello\).*\1
```

sort command

```
NAME
 sort - sort lines of text files

SYNOPSIS
 sort [OPTION]... [FILE]...
 sort [OPTION]... --files0-from=F

DESCRIPTION
 Write sorted concatenation of all FILE(s) to standard output.
   With no FILE, or when FILE is -, read standard input.

   Mandatory arguments to long options are mandatory for short options too.  Ordering options:

   -b, --ignore-leading-blanks
          ignore leading blanks

   -d, --dictionary-order
          consider only blanks and alphanumeric characters
```

```
-f, --ignore-case
       fold lower case to upper case characters

-g, --general-numeric-sort
       compare according to general numerical value

-n, --numeric-sort
           compare according to string numerical value

 -r, --reverse
           reverse the result of comparisons

 --batch-size=NMERGE
           merge at most NMERGE inputs at once; for more use temp files

 -c, --check, --check=diagnose-first
           check for sorted input; do not sort

 -C, --check=quiet, --check=silent
           like -c, but do not report first bad line

 -u, --unique
           with -c, check for strict ordering; without -c, output only the first of an equal run
```

uniq Command

```
NAME
 uniq - report or omit repeated lines

SYNOPSIS
 uniq [OPTION]... [INPUT [OUTPUT]]

DESCRIPTION
 Filter adjacent matching lines from INPUT (or standard input), writing
 to OUTPUT (or standard output).

 With no options, matching lines are merged to the first occurrence.

   -c, --count
              prefix lines by the number of occurrences

   -d, --repeated
          only print duplicate lines, one for each group
```

```
   -D     print all duplicate lines

   --all-repeated[=METHOD]
          like  -D,  but  allow  separating  groups  with  an  empty line;
          METHOD={none(default),prepend,separate}

   -f, --skip-fields=N
          avoid comparing the first N fields

   -i, --ignore-case
          ignore differences in case when comparing

   -s, --skip-chars=N
          avoid comparing the first N characters

   -u, --unique
          only print unique lines

   -z, --zero-terminated
          line delimiter is NUL, not newline

Note: 'uniq' does not detect repeated lines unless they are adjacent.
 You may want to sort the input first, or use 'sort -u' without 'uniq'.
 Also, comparisons honor the rules specified by 'LC_COLLATE'.
```

xargs

xargs converts the stdin to arguments

```
Usage: xargs [OPTION]... COMMAND [INITIAL-ARGS]...
Run COMMAND with arguments INITIAL-ARGS and more arguments read from input.

Mandatory and optional arguments to long options are also
mandatory or optional for the corresponding short option.
  -0, --null                   items are separated by a null, not whitespace;
                                 disables quote and backslash processing and
                                 logical EOF processing
  -a, --arg-file=FILE          read arguments from FILE, not standard input
  -d, --delimiter=CHARACTER    items in input stream are separated by CHARACTER,
                                 not by whitespace; disables quote and backslash
                                 processing and logical EOF processing
  -E END                       set logical EOF string; if END occurs as a line
                                 of input, the rest of the input is ignored
                                 (ignored if -0 or -d was specified)
  -e, --eof[=END]              equivalent to -E END if END is specified;
                                 otherwise, there is no end-of-file string
```

```
-I R                          same as --replace=R
-i, --replace[=R]             replace R in INITIAL-ARGS with names read
                                from standard input, split at newlines;
                                if R is unspecified, assume {}
-L, --max-lines=MAX-LINES     use at most MAX-LINES non-blank input lines per
                                command line
-l[MAX-LINES]                 similar to -L but defaults to at most one non-
                                blank input line if MAX-LINES is not specified
-n, --max-args=MAX-ARGS       use at most MAX-ARGS arguments per command line
-o, --open-tty                Reopen stdin as /dev/tty in the child process
                                before executing the command; useful to run an
                                interactive application.
-P, --max-procs=MAX-PROCS     run at most MAX-PROCS processes at a time
-p, --interactive             prompt before running commands
    --process-slot-var=VAR    set environment variable VAR in child processes
-r, --no-run-if-empty         if there are no arguments, then do not run COMMAND;
                                if this option is not given, COMMAND will be
                                run at least once
-s, --max-chars=MAX-CHARS     limit length of command line to MAX-CHARS
    --show-limits             show limits on command-line length
-t, --verbose                 print commands before executing them
```

```
  -x, --exit                  exit if the size (see -s) is exceeded
      --help                  display this help and exit
      --version               output version information and exit

Please see also the documentation at https://www.gnu.org/software/findutils/.
You can report (and track progress on fixing) bugs in the "xargs"
program via the GNU findutils bug-reporting page at
https://savannah.gnu.org/bugs/?group=findutils or, if
you have no web access, by sending email to <bug-findutils@gnu.org>.
```