44 - Text formating with awk

Syntax:
awk -F*FieldSeperator* */SearchPatern/* {*command*} *File*
z.B. awk '/ftp/ {print $0}' /etc/services

Exercises:

awk:

```
less  /etc/passwd

awk -F: '{print $1}' /etc/passwd

awk -F: '{print $1; print $3; print $5}' /etc/passwd

awk -F: '{print $1, $3, $5}' /etc/passwd

awk -F: '{print $1, "-", $3, "-", $5}' /etc/passwd

awk -F: '{printf ("%-10s %-10d %20s\n", $1, $3, $5)}' /
        etc/passwd

awk -F: '{if ($3 >= 500) printf ("%-10s %-10d %20s\n", $1, $3,
        $5)}' /etc/passwd

awk -F: '{if ($3 >= 500 && $3 < 1000) printf ("%-10s %-10d %
        20s\n", $1, $3, $5)}' /etc/passwd

awk -F: '/michel/ {printf ("%-10s %-10d %20s\n", $1, $3, $5)}'
        /etc/passwd

awk '/ftp/ {print $0}' /etc/services
```

same as

```
grep ftp /etc/services
```

Combination Examples: (ifconfig, grep, egrep, **awk)**

Display all local IP Addresses:

```
ifconfig | awk -F: '/inet addr:/{print $2}' | awk '{print $1}'
```

or

```
ifconfig | grep "inet addr:" | awk -F: '{print $2}' \
| awk '{print $1}'
```

Display main local IP address (`/root/linux_course/my_scripts/IP`):

```
ifconfig | grep -1 "eth0" | grep "inet addr:" | \
awk -F: '{print $2}' | awk '{print $1}'
```

'awk' and 'nawk'

Pattern Scanning and Processing Language

```
$awk [ options ] [ 'program' ] [ parameters ] [ files ]
$nawk [ options ] [ 'program' ] [ files ]
```

Description:

The **awk/nawk** command performs actions for lines in *files* that match *patterns* specified in *program*. Each input line is made up of fields separated by whitespace.

Options:

-f*file*	get *patterns* from *file* instead of *program*
-F*c*	separate fields with character *c* (default whitespace)
-v *variable=value*	assign *value* to variable (**nawk** only)
parameters	parameters have the format *variable=expression*
files	read standard input if *files* is **-** or no *files* are specified

Program Format:

Patterns in program can be associated with a statement to perform if an input line matches the pattern. The format is:

```
pattern { statement }
```

A missing pattern always matches, and a missing statement prints the current input line.

Patterns:

BEGIN	match before first input line
END	match after last input line
pattern1, *pattern2*, ..., *patternn*	match if *pattern1*, *pattern2*, or *patternn* match current input line
pattern1 && *pattern2*	match if *pattern1* and *pattern2* match current input line
pattern1 || *pattern2*	match if *pattern1* or *pattern2* match current input line
!*pattern*	match if *pattern* does not match current input line
/regular-expression/	match if *regular-expression* matches current input line
relational-expression	match if *relational-expression* evaluates to true

Flow Control Statements:

break	exit from a **for** or **while** loop
continue	execute next **for** or **while** loop
delete *variable*[*expression*]	delete element *expression* from array *variable*
do *statement* **while** (*expression*)	execute *statement* while *expression* is true
exit	skip remaining input
for (*expression1*; *expression2*; *expression3*) *statement*	execute *statement* while *expression2* is true; loop is usually initialized with *expression1* and incremented with *expression3*
for (*variable* **in** *array*) *statement*	execute *statement*, setting *variable* to successive elements in *array*
if (*expression*) *statement1* [**else** *statement2*]	execute *statement1* if *expression* is true, otherwise execute *statement2*
next	skip rest of the input line
return[*expression*]	return value of *expression*
system(*command*)	execute *command* and return status
while (*expression*) *statement*	execute *statement* while *expression* is true

Input/Output Statements:

close(*file*)	close *file*
getline	set **$0** to next input record; set **NF**, **NR**, **FNR**
getline<*file*	set **$0** to next input from *file*; set **NF**
getline *var*	set *var* to next input record; set **NR**, **FNR**
getline *variable*<*file*	set *variable* to next input record from *file*
command | **getline**	pipe output of *command* into **getline**
print	print current input record
print *expression*	print *expression*; multiple expressions must be separated with a ","
print *expression*>*file*	print *expression* to *file*; multiple expressions must be separated with a ","
printf *format expression*	print *expression* according to C-like *format*. Multiple expressions must be separated with a ",". Output can also be appended to *file* using >> or piped to a command using '**|**'.
printf *format expression*>*file*	print *expression* to *file* according to C-like *format*. Multiple expressions must be separated with a ",". Output can also be appended to *file* using >> or piped to a command using '**|**'.

Functions:

atan2(*x*,*y*)	arctangent of *x/y* in radians
cos(*expr*)	cosine of *expr*
exp(*expr*)	exponential of *expr*
gsub(*regular-expression*, *string1*, *string2*)	substitute *string1* for all instances of *regular-expression* in *string2*. If *string2* is not specified, use the current record **$0**.
index(*string1*, *string2*)	return the position of *string1* in *string2*
int(*expr*)	integer value of *expr*
length(*string*)	return the length of *string*
log(*expr*)	natural logarithm of *expr*
match(*string*, *regular-expression*)	return the position in *string* where *regular-expression* occurs. If not found, return **0**. **RSTART** is set to starting position, and **RLENGTH** is set to the length of *string*.
rand()	random number between 0 and 1
sin(*expr*)	sine of *expr* in radians
split(*string*, *array*)	split *string* into *array* using **$FS**
split(*string*, *array*, *fs*)	split *string* into *array* using *fs* as separator
sprintf(*format*, *expr*)	format *expr* according to the **printf** format
sqrt(*expr*)	square root of *expr*
srand()	new seed for rand (current time)
srand(*expr*)	set the seed for rand to *expr*
sub(*regular-expression*, *string1*, *string2*)	substitute *string1* for the first instance of *regular-expression* in *string2*. If *string2* not specified, use the current record **$0**.
substr(*string*, *x*)	return the suffix of *string* starting at position *x*
substr(*string*, *x*, *n*)	return *n* character substring of string starting at position *x*
function *name*(*args*,...) {*statements*}	define a function *name*
func *name*(*args*,...) {*statements*} *name* (*expr*, *expr*, . . .)	

Operators:

`=, +=, -=, *=, /=, %=, ^=`	assignment operators
`?:`	conditional expression
`||, &&, !`	logical OR, logical AND, logical NOT
`~, !~`	regular expression match/do not match
`<, <=, >, >=, !=, ==`	relational operators
`+, -`	add, subtract
`*, /, %`	multiple, divide, modulo
`+, -`	unary plus, unary minus
`^`	exponentiation
`++, --`	increment, decrement

Variables:

$ARGC	number of command-line arguments
$ARGV	array of command-line arguments
$FILENAME	current input file
$FNR	record number in current input file
$FS	input field separator (default blank and tab)
$NF	number of fields in the current record
$NR	number of current record
$OFMT	output format for numbers (default **%g**)
$OFS	output field separator (default blank)
$ORS	output record separator (default newline)
$RLENGTH	length of string matched by **match**()
$RS	contains the input record separator (default newline)
$RSTART	index of first character matched by **match**()
$SUBSEP	subscript separator (default **\034**)
$0	current input record
$*n*	*nth* input field of current record