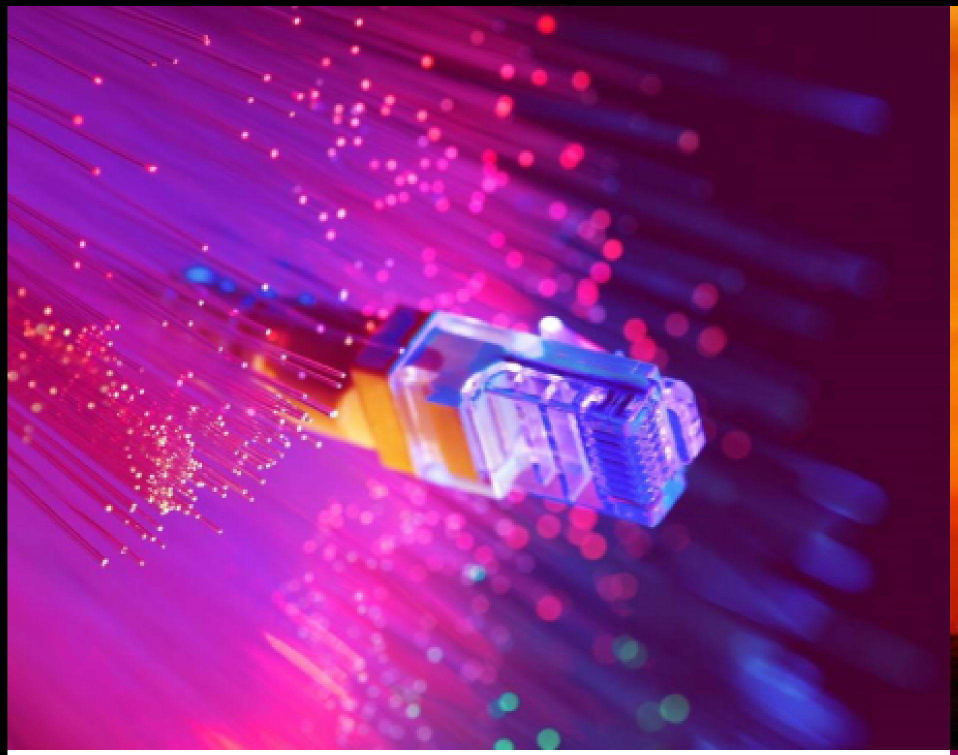

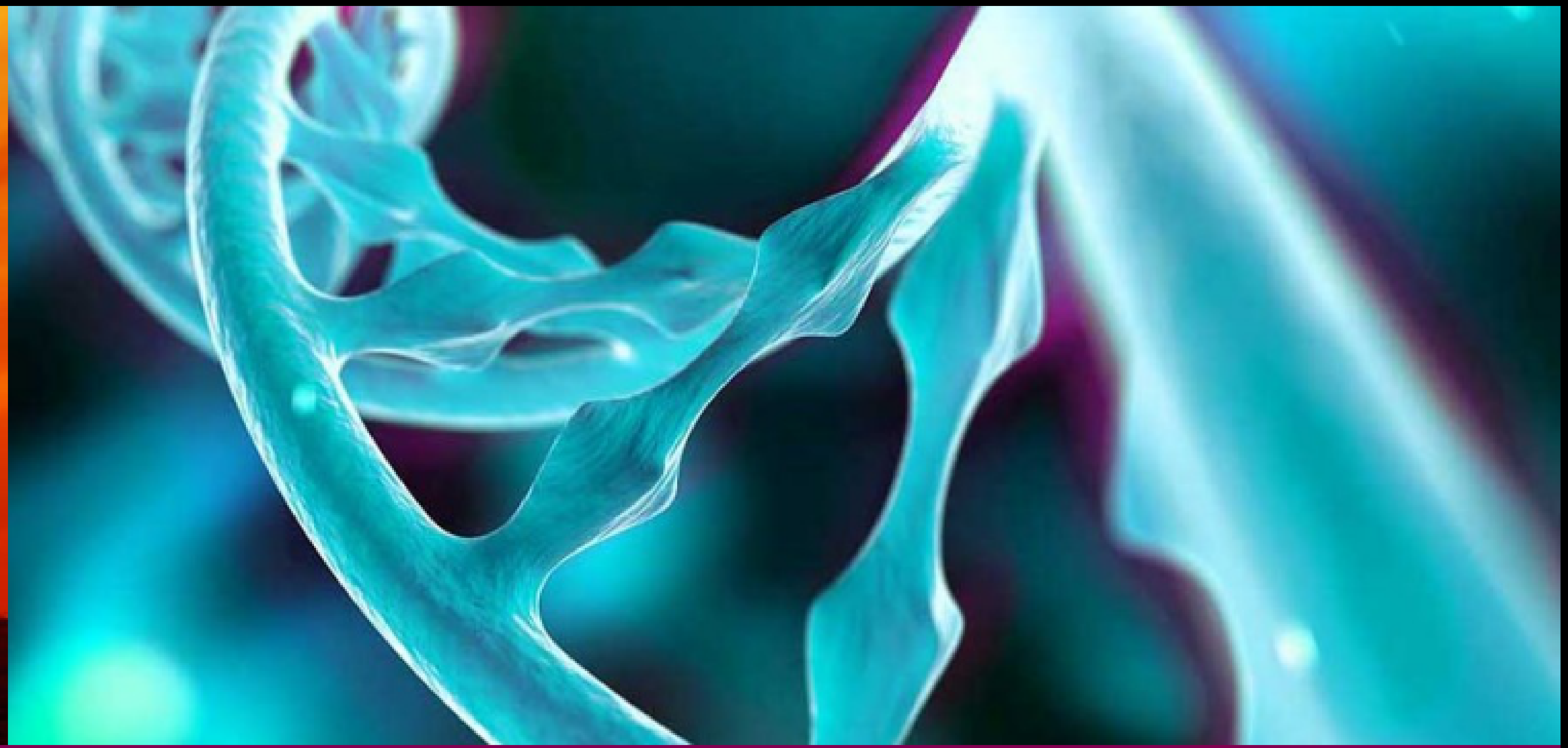

AWK - automated text editing

CSC Training, 2019-12

CSC – Finnish expertise in ICT for research, education and public administration

awk - text processing

- Developed at Bell Labs in 1977 by Aho (not Esko, but Alfred Vainö!), Weinberger, Kernighan
- A versatile scripting language which resembles C (surprise! - Kernighan & Ritchie)
- Powerful with spread-sheet type / tabulated data
- Typical usage perhaps in one-liners with matching / reordering / formatting / calculating fields from the existing tables of data
- awk command scripting is also available

awk - command line

- To print a certain column (`$2` here refers to 2nd column in row – will be explained later) of a file, type the following to the terminal:

```
$ awk '{print $2}' /etc/mime.types
```

- By default you assume that the file is separated by blank spaces.
- You can redirect the output (using the > symbol) to store the result into a new file:

```
$ awk '{print $2}' /etc/mime.types > suffixes.txt
```

- You can also use it within a pipe (feeding it with stdout):

```
$ cat /etc/mime.types | awk '{print $2}'
```

awk pattern matching

- awk commands allow to test the input against regular expression (enclosed in `/ /`):

```
awk '/regexp/ { action }' file
```

- An exclamation mark inverts match:

```
awk '!/regexp/ { action }' file
```

- For example, we want to print all relevant lines in `/etc/mime.types`, i.e., exclude all comment-lines that start with `#`:

```
$ awk '!/#/' /etc/mime.types | less
```

- compare with `less /etc/mime.types`

awk scripts

- You can save your awk directives in a text file (a.k.a. script).
- Why should I?
  - Sometimes one-liners get too long.
  - You want to be able to easily reproduce your awk-command.
  - Useful if you need to declare user defined functions through command scripts.
  - Not mandatory, but useful to give suffix .awk.
- Triggered by option -f:

```
$ awk -f myscript.awk inputfile.txt > outputfile.txt
```

awk scripts (cntd.)

- Mostly in scripts (can also be used in command line), we need pre- and postprocessing steps.
    - Actions taken *before* and *after* the text file is parsed, i.e., not tested against the input.
- This is achieved by optional `BEGIN { }` and `END { }` sections.
    - `BEGIN` is often used to initialize variables before the first input line has been read in.
    - `END` is usually used to print some summary information after input has been finished.

awk scripts (cntd.)

- Let’s write a script to display all nologin accounts in the system. Use your favourite text editor and create a new file called `nologin.awk`.
- Fill it with the following contents and save thereafter:

```
BEGIN {x=0}
/nologin/ {x=x+1; print x, " ...", $1}
END {print "------------------"; print "nologins=", x}
```

- Use `-f` option to launch the script:

```
$ awk -f nologin.awk /etc/passwd
```

Questions

How to get all users *with* login accounts are shown?

Who can produce a similar result with `grep`?

Field separator

- Field separator (`FS`), the same as -F option, can be used to indicate character(s) used to separate consecutive fields.
- Use `-F` followed by separator character(s) from command line, e.g.:

```
$ awk -F: -f nologin.awk /etc/passwd
```

- or add inside the script:

```
BEGIN { FS="[:,]" }
```

- Spot the difference in output:

```
33  ... colord              33  ... colord:x:117:124:colord
34  ... pulse               34  ... pulse:x:119:125:PulseAudio
35  ... geoclue             35  ... geoclue:x:121:127::/var/lib/geoclue:/usr/sbin/nologin
------------------          ------------------
nologins= 35                nologins= 35
```

Field separator (cntd.)

- For multiple choices of sepration characters, use regexp.
  - Your `FS` is either colon (`:`) or comma (`,`), try for instance (`NF` is number of columns – see next slide):

```
$ echo "0 1:2,3 4" | awk -F"[:,]" '{print "entries:" NF " last column: " $NF}'
```

  - spot the difference with not using regexp:

```
$ echo "0 1:2,3 4" | awk -F":," '{print "entries:" NF " last column:" $NF}'
```

  - or also including a blank:

```
$ echo "0 1:2,3 4" | awk -F"[:, ]" '{print "entries:" NF " last column:" $NF}'
```

Counters of columns, rows and records

- awk fields are accessed through variables `$1`, `$2`, ..., `$(NF-1)`, `$(NF)`.
  - `NF` (Number of Fields) is the number of fields on each line (# columns in row).

```
$ echo "0 1:2,3 4" | awk -F"[:, ]" '{print "entries:" NF " first:" $1 " last:" $NF}'
```

- `$0` refers to the whole input row.

```
awk -F":" '{printf "user: %s\n    whole line: %s\n", $1, $0}' /etc/passwd
```

- `printf` enables formatted printout - we will discuss in more details later.
- `NR` (Number of Records) is the number of input records (lines):

```
$ awk 'END {print NR}' /etc/passwd
```

- Much simpler still: `wc -l /etc/passwd`

Loops in awk

- Loops in awk are very much c-style:

```
for (countervar=initvalue; condition of validity; increment) {action}
```

- e.g., displaying single fields in row:

```
$ awk -F: '{for (i=1; i<=NF; i++) {print i, $i}; print " "}' /etc/passwd
```

- or to invert

```
$ awk -F: '{for (i=NF; i>=1; i--) {print i, $i}; print " "}' /etc/passwd
```

- or only odd lines

```
$ awk -F: '{for (i=1; i<=NF; i=i+2) {print i, $i}; print " "}' /etc/passwd
```

Output in awk

- Generic `print` just takes either strings or variables.

```
$ awk -F: '{print "string", $2, $NF, NF, NR}' /etc/passwd
```

- Alternatively, `printf` offers a wide range of C-style formatting capabilities, e.g.:

```
$ date | awk -F"[ :]" '{printf("Time=%2d hours and %2d minutes\n", $4, $5)}'
```

  - Remember not to forget to supply the newline `\n` in `printf`! The generic print already adds that for you automatically.

- Formats are: `%d` for integer, `%f` for floats, `%e` for scientific, `%s` for string
  - Length can be prescribed:

```
$ echo "1234.5678 910.16" | awk '{printf "%4.2f %1.3e \n", $1, $2}'
```

Variables in awk

- Already mentioned the awk internal ones: `NR,NF,$1,$2,...`
- User defined variables
  - Convention: use lowercase to define their names.
- Can be set inside script/command line:

```
awk 'BEGIN{myvar="Hello !"; a=1; b=2; print myvar, a, "+", b "=", a+b}'
```

  - Question: Why is everything inside `BEGIN` section?
- Or can be passed to awk from outside:

```
awk -F: -v n=1 '{print $n}' /etc/passwd
```

(try same with `n=2,3,...`)

Variables in awk (cntd.)

- We can use arrays in awk:

```
awk 'BEGIN{t[1,1]=1; t[1,2]=2; i=1; print t[1,2], t[i,i], t[i,1]}'
```

- awk arrays are in fact associative arrays.
  - **the index** into an array **does not have to be an integer number**.
  - it can be a string:

```
awk 'BEGIN{car["sweden"]="volvo"; car["russia"]="lada"; car["usa"]="pontiac"; //
for (i in car) {print i, ":", car[i]}}'
```

- NB: `//` at the end tells bash to continue the line - you may type that in one row.

Built-in functions

- Numerical functions: `int`, `exp`, `log`, `sin`, `cos`, `sqrt`.

```
$ for ((x=1; x<=180; x++)); { echo $x; } > angles.dat
$ awk '{print $1, cos($1*3.1415927/180.0)}' < angles.dat | tee cosine.dat
```

- String functions: `tolower`, `toupper`, `sprintf`, `match`, ….

```
$ awk '{print toupper($0)}' /etc/group
```

For more details, see e.g. gawk manual pages

- Bit manipulation functions: `and`, `or`, `xor`, …

```
$ awk 'BEGIN{printf "and(1,0)=%x or(1,0)=%x \n", and(1,0), or(1,0)}'
```

Control statements

- `if-else` statement (save into `sign.awk`):

```
{
  printf "cos(%f)=%2.2f, ", $1, $2
  if ($2 > 0) {print " positive"}\\
  else {print "negative"}
}
```

```
$ awk -f sign.awk cosine.dat
```

- also as ladder

```
{
   printf "cos(%f)=%2.2f, ", $1, $2
   if ($2 > 0) {print " positive"}
   else if (sqrt($2*$2) < 0.000001) {print "zero"}\\
   else {print "negative"}
}
```

Control statements (cntd.)

- logical operators: and &&, or ||.

```
# write awk script sign_product.awk
BEGIN {print "enter 2 numbers separated by space (end with CTRL+D)"}
{
    if (($1 == 0) || ($2 == 0)) {
        sign="zero"
    }
    else if ( (($1 < 0) && ($2 > 0)) || (($1 > 0) && ($2 < 0)) ) {
        sign="negative"
    } else {
        sign="positive"
    }
    printf "product of %f x %f is %s\n", $1, $2, sign
}
```

```
$ awk -f sign_product.awk
```

Further resources

- Like always, man-pages:

```
$ man awk
$ info awk
```

- awk web-manual by GNU https://www.gnu.org/software/gawk/manual
- The Internet, e.g.: https://stackoverflow.com