# Be in shell

## Text files,awk

Pavel Fibich

pavel.fibich@prf.jcu.cz
dep. Botany - Na Zlaté stoce 1

03-2016

Přírodovědecká
fakulta
Faculty
of Science

# text files

text files

# Text files manipulations

To get file from the web, you can easily use `wget` or `curl`

```
$ wget http://botanika.prf.jcu.cz/fibich/ideff.csv
...
```

We often work with csv or somehow delimited files. It is easy to exact columns from them.

```
# to get first two lines from 2. and 4. column (-d defines delimiter)
$ cut -d',' -f2,4 ideff.csv | head -n 2  # cut can N-,N-M,-M
sp,RYO
Plantago,1.11
```

We can also easily paste files together

```
$ cut -d',' -f4,1,3 ideff.csv > file1
$ cut -d',' -f4,1,5 ideff.csv > file2
$ paste -d'|' file1 file2  # paste files together
exp,IDKirw,RYO|exp,RYO,Sel
GE1,3.93,1.11|GE1,1.11,0.2
GE1,2.98,0.67|GE1,0.67,0.32
...
```

# Text files manipulations

To continue in pasting, we can also `join` files, its options define on which field to match files together

```
$ join -1 3 -2 2 -t, file1 file2
RYO,exp,IDKirw,exp,Sel
1.11,GE1,3.93,GE1,0.2
0.67,GE1,2.98,GE1,0.32
...
```

Beside printing unique lines in the file (by `uniq` command), we can also print common lines for two files (by `comm` command).

## Exercise

For `ideff.csv` (look at the 1st lesson for help)

- get unique names from the first column
- sort reversarily by 3rd column
- append line numbers (check man of `cat`)

# `tr` as *translate*

`tr` works on characters and can

- delete (`-d`)

```
$ printf "%s\n" "12 Files can be found in 54
    directories" | tr -d '1-9'
Files can be found in  directories # remove numbers
$ printf "%s\n" "My BIG files are ..." | tr -d '[:
    upper:]'
y  files are ... # remove upper cases
```

- substitute

```
$ printf "%s\n" "My BIG files are ..." | tr 'MB' 'mb'
my bIG files are ...
$ cat ideff.csv | tr '[:upper:]' '[:lower:]'
exp,sp,idkirw,ryo,sel,compl,net,oi
ge1,plantago,3.93,1.11,0.2,0.8,1,1.06
...
```

# Regular expressions

Regular expression (regex) describes a set of possible input strings; are built-in `vi`, `emacs`, `sed`, `awk`, `perl`, `python`, ... various syntax!

- if string is substring of given string or text file
- . (dot) matches any character (use backslash for regular dot)
- [] is for character class, eg. `[abc]` maches any of character abc, `[Bb]ye` matches bey and Bye, we can use ranges `[1-9]`, `[a-e]`, `[1-9a-e]`
- negation by caret `[^ eo]`
- named classes `[a-zA-Z]` for `[[:alpha:]]`, `[a-zA-Z0-9]` for `[[:alphanum:]]`, `[45a-z]` for `[45[:lower:]]`
- anchors match beginning ^ or end $
- `*` means zero or more repetitions (`ya*y` matches yay, yaaaay, ...), `{n}` n occurences, `{n,}` n or more, `{n,m}` n but max m (.0, same as .*, a2,3 matches aaa and aaaa)
- brackets: abc* matches ab, abc, abcc, .. (ab)2,3 matches abab, abababab

# Searching for lines

`grep` is a global regular expression print, eg.

```
$ grep SE ideff.csv  # print lines with SE string
SE,ss,20327.8,1.15,2157.34,1721.18,3878.52,-0.29
...
$ grep "^G" ideff.csv  # print lines starting with G, (caret)
GE1,Plantago,3.93,1.11,0.2,0.8,1,1.06
$ grep "2$" ideff.csv  # lines ending with 2
GE2,Holcus,3.773,2.52,0.97,0.265790,1.231985,-0.053142
$ grep -v Holcus ideff.csv  # print lines do not have Holcus
exp,sp,IDKirw,RYO,Sel,Compl,Net,OI
...
$ grep "Holcus\|Briza" ideff.csv  # lines having Holcus or Briza
GE2,Holcus,3.773,2.52,0.97,0.265790,1.231985,-0.053142
```

## Exercise

Write command that find all occurences of the string Holcus (case insensitive) in all .csv files in the current directory.

# `sed` as *stream editor* - substitute

or Stream oriented non-interactive text EDitor. `sed` is filter, do not modify original file, but std. output. Sed is fast and concise.
*Answer for*: How to substitute 'Prunella' for 'allheal' or numbers in the file? How to delete first line?

```
$ cat ideff.csv | grep Prunella
GE1,Prunella,2.98,0.67,0.32,0.52,0.84,0.99
GE2,Prunella,1.526,0.62,0.83,0.295264,1.125914,-0.028994
$ sed "s/Prunella/allheal/g" ideff.csv | grep allheal
GE1,allheal,2.98,0.67,0.32,0.52,0.84,0.99
GE2,allheal,1.526,0.62,0.83,0.295264,1.125914,-0.028994
$ sed "s/[[:digit:]]/_/g" ideff.csv | tail
GE_,Holcus,_.___,_.__,_.__,_._____,_._____,-_._____
```

Pattern have often 3 parts:

- command: substitute `s` (`5 s`,`5!s`, `5,10s`), append `a`, insert `i`, delete `d` (`1d`, `1~2d`), `p` print (`1,5p`), ...
- what/for what to change
- range on line `g` - all occurances on line; without it only the first

# `sed` as *stream editor* - print,append

How to delete first line or print exact lines?

```
$ cat ideff.csv | sed '1d'
GE1,Plantago,3.93,1.11,0.2,0.8,1,1.06
...
$ sed '/^$/d' ideff.csv  # delete blank lines, ^ beginning, $ end of line
...
$ sed -n '5,6p' ideff.csv  # print 5-6. line, -n print only lines with p
GE1,Agrostis,4.55,1.13,0.13,0.66,0.78,0.99
GE2,Holcus,3.773,2.52,0.97,0.265790,1.231985,-0.053142
# delete lines in matched range
$ sed '/Agrostis/,/Lychnis/d' ideff.csv
...
```

Append line before the matched lines

```
$ sed "/Plantago/i NEWONE:" ideff.csv
exp,sp,IDKirw,RYO,Sel,Compl,Net,OI
NEWONE:
GE1,Plantago,3.93,1.11,0.2,0.8,1,1.06
...
```

# `sed` as *stream editor* - complex

To combine more command use `-e`

```
$ sed -e '1,2s/[1-9]/x/g' -e '1d' ideff.csv
GEx,Plantago,x.xx,x.xx,0.x,0.x,x,x.0x
GE1,Prunella,2.98,0.67,0.32,0.52,0.84,0.99
```

More complex matching can be done by specifying patterns in () and later used by backslach and order of ()

```
# print non digit start of line before , matched by () pasted by backslash 1
$ sed 's/\([^1-9]*\),\(.*\)/\1/' ideff.csv
exp,sp,IDKirw,RYO,Sel,Compl,Net
GE1,Plantago
...
# switch 1. and 2. column divided by ,
$ sed 's/\(.*\),\(.*\)/\2,\1/' ideff.csv
OI,exp,sp,IDKirw,RYO,Sel,Compl,Net
1.06,GE1,Plantago,3.93,1.11,0.2,0.8,1
```

*Drawbacks*: do not remember text from one line to another, no facilities to manipulate numbers, cumbersome syntax

# Exercise

Run and get `ideff.csv` (beginning of this lesson)

```
$ mkdir textfiles; cd textfiles; touch fil{k..o}
$ touch fil{0..17}.log; touch {20..25}; touch a.x{a..e}
```

## Exercise

- print all lines with negative numbers from `ideff.csv`
- print all files having: (1) name from 4 characters (2) number together with alphabetic character in the name
- create new `ideffP.csv`, removing - mark from the original file
- print all files in the current folder with .* suffixes in upper style
- create new `ideffL.csv` by changing the first column of `ideff.csv` in lower style
- print all files with . in name and change . for `DOT` (use backslash)
- write command that will find `text` in all files in the current folder except of files that end with `.log`

# Homework 2

Download `http://botanika.prf.jcu.cz/fibich/bash/home2.tar.gz`, unpack (e.g. `tar -xvf home2.tar.gz`). Write a script that if you run it inside `home1` folder, will print (instead of `...` there are values or lines)

```
$ createSum.sh  # print folders and number of subfolders
SE 6
GE1 6
GE2 15
$ createSum.sh -c  # compound files
SE,plpru,-0.9117911229,-0.4025109039,...,-0.4606028115,0
SE,plach,0.3984090416,0.0980536463,...,-0.4888029837,1
...
GE1,ssws,1.4582010773,0.4898979486,...,1.1983342482,1
...
```

# Homework 2 - details

```
$ createSum.sh -c -p :  # compound files and specify separator
SE:plpru:-0.9117911229:-0.4025109039:...:-0.4606028115:0
SE:plach:0.3984090416:0.0980536463:...:-0.4888029837:1
...
GE1:ssws:1.4582010773:0.4898979486:...:1.1983342482:1
...
$ createSum.sh -h  # will print short help
```

# Homework 2 - details

## Homework1

- Script will create text on std. output based on files in folders in the current folder.
- It takes 1st folder as value for 1st column, 2nd folder (inside 1st folder) as the second column and values inside 1st/2nd/data.csv are as following columns.
- User can specify 3 options: `-h` for short help, `-c` to compound files, `-p` takes argument that is used as separator of columns (eventhrough there are , in data.csv). Use `getopts`!
- Homework is not team work, do it yourself.
- Send me the solution (script name according your surname: createSum.SURNAME) by email before 18:00 10 April
- Shorter script better script!

awk

# awk

AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

```
$ awk -F, '{print $2,$1}' ideff.csv # print 2. and 1. column
sp exp
Plantago GE1
..
$ echo $PATH | awk -F: '{print toupper($1)}' # -F is field sep.
/USR/LOCAL/BIN
$ awk -F, 'BEGIN{print "HI"} {print $2,$1}' ideff.csv
HI
sp exp
..
```

Basic structure of the script is in ”

```
BEGIN{ print "START" } # what is run before 1. line
 { print } # what is run for each line of the input
END { print "STOP" } # what is run after the last line
```

# awk over sed

awk pattern action language like sed (but convenient number processing, conventional way of accesing fields, flexible printing, built-in arithmetics and string functions). No variable declaration.

```
$ awk 'BEGIN{sum=0} {sum++} END{print sum}' ideff.csv
15
$ awk '{print $0}' ideff.csv # the same as cat
...
$ awk -F, '{print $3*$4}' ideff.csv # multiply 3. and 4. column
...
```

Few built-in variables

- FS (OFS) field separator (passed by -F option) and output separator
- NF number of fields in the line (`print $(NF-1)` to print pre-last column)
- NR line number (`print NR,$NF` print last column with line numbers)
- FILENAME, ARGC, ARGV ...

# awk - selection

```
awk 'searchPattern {commands}'
```

Easy conditional selection by comparison, computation or by string

```
$ awk -F, '$3 > 3 {print $3}' ideff.csv # 3. column bigger than 3
$ awk -F, 'log($3) > 3 {print $3}' ideff.csv # log of 3. column
$ awk -F, '$1 == "SE" {print $3}' ideff.csv # column matching
$ awk -F, '/ll/ {print $3}' ideff.csv # string matching
$ awk -F, '/1$/ {print $0}' ideff.csv # lines end with 1
...
```

To combine conditions, use && or ||

```
$ awk -F, '$3 > 3 && $2 > 1 {print $3}' ideff.csv
...
```

Line can by selected by number of fields

```
$ awk -F, 'NF >3 {print $3}' ideff.csv # more than columns
...
```

# awk - built-in

awk contains a number of built-in functions

- string - length (length of string), substr(s,m,n) substring of s from m-th position at most n characters, split(s,a,d) place elements of s delimited by d into array a, sub, toupper, tolower, printf print formating in c style
- arithmetics - sin, cos, atan, int, exp, log, rand, sqrt, ...
- special - system (executes a linux command, system("clear")), exit (stop and go to END)

To get environmental variable, ENVIRON["VARNAME"] is used.

```
$ awk 'BEGIN {print ENVIRON["PATH"]}' ideff.csv
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

# awk - built-in

Also control of the flow by if-else, while and for loops.

```
... END { if (n>0) print n} ...
... {while (i <= $3) { printf("%f",$1*$2+i); i = i +1} }
```

Arrays subscripts can have any value (also associative arrays), elements are not declared.

```
$ awk -F, '{line[NR]=$0} END { for(i=NR;i>0;i=i-1) print(
    line[i])}' ideff.csv
...  # print file clockwisely
# for loop for associative array
for (v in array) { print array[v] }
```

Own functions can be defined too

```
$ awk -F, 'function foo() { a = exp($4); printf("%f\n",a)
     } { foo() }' ideff.csv
...
```

# awk script

We can have awk script

```
$ cat awk.a
#! /usr/bin/awk -f
{ print $1}
$ chmod u+x awk.a
$ ./awk.a ideff.csv
```

Awk can be easily combined with other commands

```
$ ls -l | awk '{print $5}'  # list of file sizes
```

First version released 1977 by Aho-Weinberg-Kernighan, actual verion is often called as new awk. Beside `awk` there is GNU awk, often run by `gawk` that is also ported to many OSs.

# awk - exercise

Download

`http://botanika.prf.jcu.cz/fibich/bash/inteff.csv`

## Exercise on awk

- get only number of lines from the command `wc -l inteff.csv`
- print second column from `inteff.csv` having ”pru” inside
- use `inteff.csv` to print file in form

```
exp=GE1,mix=plpru,int=5.74,Sel=0.35
exp=GE1,mix=plach,int=3.5,Sel=0.29
```

- print number and length of each line in given file
- print every third line of the file, later get value that determines which line to print from environmental variables
- print sum, max and mean of 3. and 5. column of `inteff.csv`
- count frequencies of values in the first column of `inteff.csv`
- write a command to print the columns in a text file in reverse order