Text processing

Adam Belloum
a.s.z.belloum@uva.nl

ES 2016/2017

Material Prepared by Eelco Schatborn

What if you are asked to …

- Remove duplicate lines from file(s)
- Count the frequency of words in a file(s)
- Extract message header from an email folder(s)
- Merge together files into a single, multi-column file(s)
- Extract users' login names and shells from the system passwd
- …

The best option is to get the job done with the minimum effort (re-using existing tools, not code from scratch)

most of time there are tools ready to use …

- Many great processing tools exist
- There is much overlap between tools
- Use the **right tool** for the **right job**. . .

"When the only tool you own is a hammer, every problem begins to resemble a nail."

(Abraham Maslow)

What tools?

- You will see them all used, so **know them all**
- **Be fluent** in using **a few**
- Big differences in implementations
- Always **test** your results **extensively**

Eelco Schatborn

In Unix/linux world …

- There are tools programs called "**Filters**" which can help in solving all the text processing problems stated in the previous slides

- A **Filter** is command-line:
  1. takes either std input or content of file(s)
  2. performs some processing the data
  3. And produces an output

Common Filters

- cut, paste
- tr, sort, uniq

- Ed (text editor)
- sed (streamline text editor)

- awk
- tail, head, cat, more …

- And many others see
  - http://tldp.org/LDP/abs/html/textproc.html

More Text processing tasks to do

1. Select **Column** of Characters
2. Select **Column** of Characters using Range
3. Select **Column** of Characters using Start/End Position
4. Select a Specific **Field** from a File
5. Select Multiple **Fields** from a File
6. Select **Fields** Only When a Line Contains the Delimiter
7. Select All **Fields** Except the Specified Fields
8. Change Output Delimiter for Display
9. Change Output Delimiter to Newline

cut

- A tool for extracting fields from files.
- It first appeared in AT&T System III UNIX in 1982.
- It may be simpler to use *cut* in a script than *awk*.

- Particularly important are:
  - -d (delimiter)
  - -f (field specifier).
  - -c (character positions)

http://tldp.org/LDP/abs/html/textproc.html

cut: vertical cutting

```
$ cat file
abcd:ABCD
efgh:EFGH
ijkl:IJKL
```

```
$ cut -d : -f 2 file
ABCD
EFGH
IJKL
```

```
$ cut -c 2-4,8 file
bcdC
fghG
jklK
```

-d (delimiter)
-f (field specifier).
-c (character positions)

cut: vertical cutting

- example

```
$ cat /etc/passwd
Root:x:0:0:root:/root:/bin/bash
Daemon:x:1:1:daemon:/usr/sbin:/bin/sh
Bin:x:2:2:bin:/bin:/bin/sh
Sys:x:3:3:sys:/dev:/bin/sh
…
```

Extract users' login names and shells from the system passwd

```
$ cut -d : -f 1,7 /etc/passwd
Root:/bin/bash
Daemon:/bin/sh
Bin:/bin/sh
Sys:/bin/sh
…
```

Show the names & login times of the currently logged-in users

```
$ who | cut -c 1-16,26-38
```

-d (delimiter)
-f (field specifier).
-c (character positions)

Solutions of Text processing tasks in slide 7

1. Select Column of Characters
2. Select Column of Characters using Range
3. Select Column of Characters using Start/End Position
4. Select a Specific Field from a File
5. Select Multiple Fields from a File
6. Select Fields Only When a Line Contains the Delimiter
7. Select All Fields Except the Specified Fields
8. Change Output Delimiter for Display

```
1. $ cut -c2   test.txt
2. $ cut -c1-3 test.txt
3. $ cut -c3-  test.txt
4. $ cut -c-8  test.txt
5. $ cut -d':' -f1 passwd
6. $ cut -d':' -f1,6 /etc/
7. $ cut -d':' --complement
   -s -f7
8. $ cut -d':'  -s -f1,6,7
   --output-delimiter='#'
```

http://www.thegeekstuff.com/2013/06/cut-command-examples/

More Text processing tasks to do

1. Join **all lines** in a file
2. Join **all lines** using the comma delimiter
3. Merge a file by pasting the data into 2 columns
4. Merge a file by pasting the data into 2 columns using a colon separator
5. Merge a file into 3 columns using 2 different delimiters
6. Show contents of 2 files side by side
7. Show contents of 2 files side by side with a comma separator

http://www.theunixschool.com/2012/07/10-examples-of-paste-command-usage-in.html

Paste

- Tool for **merging** together **different files** into a single, **multi-column** file.

- Example of usage:
  - In combination with cut, useful for creating system log files.

http://tldp.org/LDP/abs/html/textproc.html

paste: vertical 'catting'

```
$ cat names
Jan
Bob
Klaas
```

```
$ paste -d : names numbers
Jan:0123
Bob:7654
Klaas:3456
```

```
$ cat numbers
0123
7654
3456
```

$ man paste

..

-s Concatenate all of the lines of each separate input file in command line order…

-d list Use one or more of the provided characters to replace the newline characters instead of the default tab …

More examples

List the files in the current directory in three columns:

```
$ ls | paste - - -
```

Combine pairs of lines from a file into single lines:

```
$ paste -s -d '\t\n' myfile
```

Number the lines in a file, similar to nl:

```
$ sed = myfile | paste -s -d '\t\n' - -
```

Create a colon-separated list of directories named bin, suitable for use in the PATH environment variable:

```
$ find / -name bin -type d | paste -s -d :  -
```

$ man paste

..

-s Concatenate all of the lines of each separate input file in command line order…

-d list Use one or more of the provided characters to replace the newline characters instead of the default tab …

If `-' is specified for one or more of the input files, the standard input is used; standard input is read one line at a time, circularly, for each instance of `-'.

More Text processing tasks to do

1. Create a list of the words in file1, one per line, where a word is taken to be a maximal string of letters
2. Translate the contents of file1 to uppercase.
3. Remove all non-printable characters from file1.
4. Remove all "diacritical" marks from accented versions of the letter e.

tr: character translation filter

- The tr utility copies the **standard input** to the **standard output** with substitution or deletion of selected characters

- Must use **quoting** and/or **brackets**, as appropriate.
  - Quotes prevent the shell from **reinterpreting** the special characters in **tr** command sequences.
  - Brackets should be quoted to prevent expansion by the shell.

- Either
  - **`tr "A-Z" "*" < filename`** or **`tr A-Z \* < filename`**

    changes all the uppercase letters in filename to asterisks (writes to stdout).

http://tldp.org/LDP/abs/html/textproc.html

tr

Example 1

```
$ cat file
(((0 + 0) * 1) - 2)
$ tr '(' '{' < file| tr
  ')' '}'
$ tr '()' '{}' < file

{{{0 + 0} * 1} - 2}
```

Example 2

```
$ cat names
Jan Marie Bob klaas
$ tr [a-z] [A-Z] < names
JAN MARIE BOB KLAAS
```

$man tr

…

-C Complement the set of characters in string1, that is ``-C ab'' includes every character except for `a' and `b'.

-c Same as -C but complement the set of values in string1.

-d Delete characters in string1 from the input.

-s Squeeze multiple occurrences of the characters listed in the last operand (either string1 or string2) in the input into a single instance of the character This occurs after all deletion and translation is completed.

-u Guarantee that any output is unbuffered.

More examples

- Create a list of the words in file1, **one per line**,

  where a word is taken to be a maximal string of letters.

```
$ tr -cs "[:alpha:]" "\n" < file1
```

- Translate the contents of file1 to **upper-case**.

```
$ tr "[:lower:]" "[:upper:]" < file1
```

- **Strip out non-printable** characters from file1.

```
$ tr -cd "[:print:]" < file1
```

$man tr [CcSU] String 1 String2

…

-C Complement the set of characters in string1, that is ``-C ab'' includes every character except for `a' and `b'.

-c Same as -C but complement the set of values in string1.

-d Delete characters in string1 from the input.

-s Squeeze multiple occurrences of the character listed in the last operand (either string1 or stri in the input into a single instance of the chara This occurs after all deletion and translation is completed.

-u Guarantee that any output is unbuffered.

sort

- File sort utility, often used as a **filter** in a **pipe**.
- This command sorts a ***text stream*** or **file**
  - forwards or backwards,
  - or according to various keys or character positions.

- Using the -m option, it merges presorted input files.

http://tldp.org/LDP/abs/html/textproc.html

sort

```
$ cat names
Marie
Jan
Piet
Johanna
Dirk

$ tr [a-z] [A-Z] < names | sort -r
PIET
MARIE
JOHANNA
JAN
DIRK
```

$ man sort
…
-b, --ignore-leading-blanks
-d, --dictionary-order
-f, --ignore-case
-g, --general-numeric-sort
-i, --ignore-nonprinting
-M, --month-sort
-n, --numeric-sort

sort

```
$ cat numbers
345
2
11
067
8
```

```
$ sort -n numbers
2
8
11
067
345
```

```
$ sort numbers
067
11
2
345
8
```

$ man sort

…

-b, --ignore-leading-blanks

-d, --dictionary-order

-f, --ignore-case

-g, --general-numeric-sort

-i, --ignore-nonprinting

-M, --month-sort

-n, --numeric-sort

uniq

- This filter **removes duplicate** lines from a **sorted file**.
- It is often seen in a pipe coupled with sort.

```
$ cat words
aap
noot
mies
noot
noot
aap
wim
```

```
$ uniq words
aap
noot
mies
noot
aap
wim
```

```
$ sort words | uniq -c
2 aap
1 mies
3 noot
1 wim
```

```
Options:
   c count
   u shows uniques
   d show duplicates
   i ignore case
```

Unix Editors

- Question that might cross your mind right now:
  - "why a UNIX programmer should know these editing tools when more elegant editors like emacs exist."

1. Some of the commands from the early editors like ed can be used with later, more sophisticated editors
2. String searches use the same constructs in the majority of the UNIX editors.
3. The syntax you learn with ed is also used in other UNIX tools like grep and diff. Learning ed will enable you to use these other tools with less difficulty.
4. After learning the tools available in ed, other powerful file editing tools like sed are easy and natural to use to edit whole files.
5. If you don't have access to a screen editor you can still perform edits using line editors like ed and ex.
6. It is often easier to direct someone to use a line editor like ed, rather than have them start a screen editor like vi. .

http://snap.nlc.dcccd.edu/learn/nlc/ed.html

ed utility

- The ed utility is a **line-oriented** text editor.
- It is used to **create**, **display**, **modify** and otherwise **manipulate** text files.
- When invoked as red, the editor runs in ”restricted" mode,
  - can only :
    - edit file in the current directory
    - or execute shell command `!’ (interpreted as shell commands by ed) or contain a `/'.

ed: Line editor

```
$ cat phonenumbers
Jan Mobiel : 0612345678
Marie Thuis : 0201234567
Marie Mobiel : 0654321321
Koos : 0107654321
Truus : 03012345678
Truus Mobiel : 0687654321
```

```
$ ed phonenumbers
139
p
Truus Mobiel:0687654321
1
Jan Mobiel:0612345678
/Mari/
Marie Thuis:0201234567
```

ed: Line editor

```
g/Mob/p
Jan Mobiel : 0612345678
Marie Mobiel : 0654321321
Truus Mobiel : 0687654321
```

g/re/p: global regular expression print
(origin of the name grep)

```
/Tru/
Truus : 03012345678
s/Truus/Truus Thuis/p
Truus Thuis: 03012345678
q
```

Example create a file with ed

```
$ ed
a
bc
def
ghij
.
d
w myfile
5
q
```

```
$ cat myfile
a
bc
def
ghij
.
d
```

```
$ cat myfile
bc
def
```

An ed-script

```
$ cat edscr
    g/Jan/s/Jan/Johan/g
    g/Mobiel/s/Mobiel/Mob/g
    w

$ ed phonenumbers < edscr
    141
    234
    ?
$ cat phonenumbers
    Johan Mob : 0612345678
    Marie Thuis : 0201234567
    Marie Mob : 0654321321
    Koos : 0107654321
    Truus : 03012345678
    Truus Mob : 0687654321
```

```
$ cat phonenumbers
Jan Mobiel : 0612345678
Marie Thuis : 0201234567
Marie Mobiel : 0654321321
Koos : 0107654321
Truus : 03012345678
Truus Mobiel : 0687654321
```

sed: Stream editor

- Not interactive
- **Automat** editing of **one** or **more files**
- **Repeat** the **same** edits on **multiple files**
- sed will apply **commands** on every line of input

```
sed s/regular/complex/
ed g/regular/s/regular/complex/

$ cat words
aap noot mies noot
aap noot mies noot
$ sed s/noot/wim/ words
aap wim mies noot
aap wim mies noot
```

Basic sed operators

Operator	Name	Effect
`[address-range]/p`	print	Print [specified address range]
`[address-range]/d`	delete	Delete [specified address range]
`s/pattern1/pattern2/`	substitute	Substitute pattern2 for first instance of pattern1 in a line
`[address-range]/s/pattern1/pattern2/`	substitute	Substitute pattern2 for first instance of pattern1 in a line, over *address-range*
`[address-range]/y/pattern1/pattern2/`	transform	replace any character in pattern1 with the corresponding character in pattern2, over *address-range* (equivalent of **tr**)
`g`	global	Operate on *every* pattern match within each matched line of input

http://tldp.org/LDP/abs/html/x22444.html#AEN22474

Examples of sed

Notation	Effect
8d	
/^$/d	
1,/^$/d	
/Jones/p	
s/Windows/Linux/	
s/BSOD/stability/g	
s/ *$//	
s/00*/0/g	
/GUI/d	
s/GUI//g	

http://tldp.org/LDP/abs/html/x22444.html#AEN22474

sed [options] script filename

```
$ sed s/Unix/UNIX/ file > temp
$ mv temp > file
```

More than one occurrence of `unix' on a line

```
$ sed s/Unix/UNIX/g file > temp
```

Multiple substitutions:

```
$ sed -e s/Unix/UNIX/g -e /Windows/d file > temp
```

Many substitutions:

```
$ sed -f changes file > temp
$ cat changes
s/Unix/UNIX/g
/Windows/d
```

Suppress output:

```
$ sed -n -e s/Unix/UNIX/gp file > temp
```

awk: pattern-directed scanning and processing language

- *awk* is a full-featured text processing language with a syntax reminiscent of *C*.

- awk **breaks each line** of input passed to it into fields.
  - By default, a field is a string of consecutive characters delimited by whitespace (can be changed)

- awk **parses** and **operates** on each separate **field**.
  - This makes it ideal for handling structured text files
    - especially tables
    - data organized into consistent chunks, such as rows and columns.

awk: pattern-directed scanning and processing language

- A. Aho, P. Weinberger, and B. Kernighan.
- developed from grep, C, and sed syntax

Usage:

```
$ awk [options] script filename
```

Script:

```
pattern { action }
```

Pattern:

```
regular expressions, BEGIN, END
```

awk: examples

```
$ echo One Two | awk '{print $1}'
One

$ echo One Two | awk '{print $2}'
Two
```

But what is field #0 ($0)?

```
$ echo one two | awk '{print $0}’
one two # All the fields!
```

awk: examples

Prints field #3 of file $filename to **stdout**.

```
$ awk '{print $3}' $filename
```

Prints fields #1, #5, and #6 of file `$filename`.

```
$ awk '{print $1 $5 $6}' $filename
```

Prints **the entire file**!

```
awk '{print $0}' $filename
```

same effect as:

```
$ cat $filename
$ sed '' $filename
```

Forcing a log-off

```
#!/bin/bash
# Killing ppp to force a log-off.
# For dialup connection, of course.

# Script should be run as root user.

SERPORT=ttyS3
#  Depending on the hardware and even the kernel version,
#+ the modem port on your machine may be different --
#+ /dev/ttyS1 or /dev/ttyS2.


killppp="eval kill -9 `ps ax | awk '/ppp/ { print $1 }'`"
#                      -------- process ID of ppp -------

$killppp                     # This variable is now a command.


# The following operations must be done as root user.

chmod 666 /dev/$SERPORT      # Restore r+w permissions, or else what?
#  Since doing a SIGKILL on ppp changed the permissions on the serial port,
#+ we restore permissions to previous state.

rm /var/lock/LCK..$SERPORT   # Remove the serial port lock file. Why?

exit $?
```

http://tldp.org/LDP/abs/html/internal.html#EX44

awk: useful constructions & examples

- cat eg4.txt

```
The cow jumped over the moon
And the dish ran away with the spoon
```

- printf statements

```
awk '{for (j=1; j <= NF; j++) { \
printf("%d\t%s\n",j,$j);}}' eg4.txt
```

  - what if I want continuous numbering?

```
awk 'BEGIN {idx=0;} {for (j=1; j <= NF; j++) { \
printf("%d\t%s\n",idx,$j); idx++;}}' eg4.txt
```

- substrings
  - substr(<string>, <start>, <end>)
  -

```
awk '{for (j=1; j <= NF; j+=2) { \
printf("%s ",substr($j,1,3))}; print "";}' eg4.txt
```

The jum the
And dis awa the

1 The
2 cow
3 jumped
4 over
5 the
6 moon
1 And
2 the
...

Counting Letter Occurrences

- Look at the example
  - http://tldp.org/LDP/abs/html/awk.html#AWKREF
- Read it
- Try it
- Try to understand how awk is used in this bash script
- As suggested at the end of this bash script compare it with letter-count.sh

  http://tldp.org/LDP/abs/html/extmisc.html

Stripping comments from C program files

- Look at the example
  - http://tldp.org/LDP/abs/html/filearchiv.html#STRIPC
- The script show the usage of both sed and awk in shell programming

Suggested Tutorial

- Unix Shell Scripting Tutorial - Text Processing (Part 1) http://www.youtube.com/watch?v=aWxG8TqudTU
- Unix Shell Scripting Tutorial - Text Processing: Grep(Part 2) http://www.youtube.com/watch?v=VUoyeyFpuek
- Unix Shell Scripting Tutorial - Text Processing: Sort(Part 3) http://www.youtube.com/watch?v=VJ_bxbtxL4w
- …