Operating systems 1, Lecture 4

Sanda-Maria AVRAM, Ph. D.

Leave a feedback

Unix filters

Any command that reads a file from the standard input converts it in some way and displays it at the standard output.

sed	**uniq**
grep	**wc**
awk	**head**
sort	**tail**

sed (stream editor = noninteractive text editor)

Process line-by-line text files using a temporary buffer and display the temporary buffer at the standard output (unless the -n (--quiet, --silent) option is used).

sed [-n] [-e *script*] [-f *script_file*] [*file_list*]

```
script:

        condition   instruction
```

sed. Conditions

no condition	true for all lines in the file
n	true for the n-th line (lines are numbered cumulatively in the file list)
$	true condition for the last line in the file
/regular expression/	a true condition for lines containing at least one substring that matches the regular expression
expr1, expr2	true for lines between the line that matches expr1 and the line that matches expr2

sed. Conditions. Examples

```
sed 1,10 instruction file1
# executes instructions on lines from 1 to 10

sed 10,$ instruction file1 file2
# executes instruction on lines from 10 to the end of the file
# obtained by conclating file1 with file2
```

sed. Instructions

p	displays the temporary buffer at the standard output
d	delete the temporary buffer
i\\<ENTER>	has as a parameter a text (given on the following lines in the script file) which it displays at the standard output before the processed line
a\\<ENTER>	analogue to **i\** but displays the text after processing each line
y/str1/str2/	(where str1 and str2 have equal lengths) performs a translation by replacing the characters in the input files found in str1 with the corresponding characters in str2

sed. Instructions. Examples

```
sed p file1 # displays each line twice
sed -n p file1 # displays each line once

# script_file:
#                 i\
#                 ttttt\
#                 11111
sed -f script_file file1

echo "DECEMBER JANUARY" | sed y/ABC/123/    # ==> DE3EM2ER J1NU1RY
```

sed. Instructions (cont.)

s/regular_expression/str/[flags]

replaces the first occurrence of a string that matches the regular expression with str

flags:

nothing	replaces only the first occurrence
n	(**n** is a number between 1 and 512) replaces the **n**th occurrence
g	replaces all occurrences in the lines
p	displays the buffer on the output if there has been a change in that line

sed. Instructions (cont.) Examples

```
echo Sunday|sed s/day/night/    # ==> Sunnight

# if one wants to replace /usr/local/bin with /common/bin
sed 's/\/usr\/local\/bin/\/common\/bin/' old > new
sed 's_/usr/local/bin_/common/bin_' old > new
sed 's|/usr/local/bin|/common/bin|' old > new
sed 's:/usr/local/bin:/common/bin:' old > new

sed /YES/s/yesyes/noway/g old # replace only in lines containing YES

sed -n '/baz/s/foo/bar/p' old    # displays only lines that have changed, due to
flag p and substitution is only for lines containing baz
```

grep

Searches for a specific character string in a file or multiple files and displays the outcome to the standard output. The name comes from the English expression „global/regular expression/print".

grep [-chilnqsvw] [[-e] *regular_expression* | -f *script_file*] [*file_list*]

grep. Options

-c	(*count*) displays only the number of lines that match the regular expression
-h	(*hide*) does not display the file name
-i	(*ignore case*) does not make the difference between upper and lower case letters
-l	only displays the file names that contain the string that is searched
-n	displays the lines that match the regular expression preceded by the line number relative to the beginning of each file
-q, -s	displays nothing, to determine whether or not there was at least one match
-v	displays lines that do not contain the given string
-w	displays the lines where the string you are looking for is an entire word
-e	is used if we want the regular expression to begin with "-"

grep. Options. Examples

```
» grep "something" file1    # displays the lines in which something appears
there would be something ... or something else
» grep -c "something" file1    # (count) only the number of lines is displayed
1
# If the search is done in several files, the file name of the displayed line is
specified:
» grep "something" file1 f2
file1:there would be something ... or something else
» grep -h "something" file1 f2    # no file name is displayed
there would be something ... or something else
» grep -l "something" file1 f2 f3    # only the name of the file is displayed
file1
```

Extended regular expression (ERE)

^	the beginning of the line (if ^ is the first character in the regular expression)
$	the end of the line (if $ is the last character in the regular expression)
.	any character
[list]	any character in list
[c1-c2]	any character between c1 and c2 in lexicographic order
[^list]	the negation of [list]
*****	repeats the previous regular expression as many times as possible
+	as *, but repeats once or more
?	as *, but repeats once or zero times

Extended regular expression (ERE) (cont.)

{n}	(where **n** is a number between 0 and 255) repeat the previous expression of exactly **n** times
{n,}	repeats the previous regular expression at least **n** times
{n,m}	repeats the previous regular expression of at least **n** times and at most **m** times
(regular_exp)	group several characters into an expression
\\n	replaces a string with the **n**th regular expression found in the brackets **( )**
regexp1 | regexp2	matches either regexp1 or regexp2

Extended regular expression (ERE) (cont.)

\	escape, changes the meaning of the character following it, between normal and special
\\<	beginning of word
\\>	end of word

sed. Examples with regular expressions

```
echo abcd123ef | sed -E 's/([a-z]*).*/\1/' # displays only the first group of letters
# interchange the first two words:
echo abcd efg | sed -E 's/([a-z]*) ([a-z]*)/\2 \1/'
# eliminates the space between words:
echo abcd efg | sed -E 's/([a-z]*) ([a-z]*)/\1\2 /'

# white space = space, TAB
sed -E 's/^[ \t]*//g' f1 # removes all white spaces at the beginning of the lines
sed -E 's/[ \t]*$//g' f1 # eliminates all white spaces at the end of the lines

# removes all white spaces at the beginning and end of the lines:
sed -E 's/^[ \t]*//g;s/[ \t]*$//g' f1
```

grep. Examples with regular expressions

```
grep ^Ab file1 file2 # displays the lines in file1 and file2 starting with Ab
grep nd$ file1 file2 # displays the lines in file1 and file2 ending with nd
grep -E '^(.*) (.*) \1$' file1 # displays all lines that begin and end with the
same word, separator being space, and contain more than two words

grep -E 'o{3}' old    # displays lines containing "ooo"
grep -E 'o{2,}' old    # displays lines containing "oo", or "ooo", ...
grep -E 'fo*' old    # displays lines containing "f", or "fo", or "foo", ...
grep -E '^fo*' old    # shows lines starting with "f", or "fo", ...
# displays lines ending with $var:
grep -E ''$var'$' old
grep -E "$var$" old
```

awk

*Processes text files by selecting those lines that satisfy the conditions imposed by a list of templates (regular expressions). Its name comes from its three designers and implementers: A. **A**ho, P. **W**ieinberger and B. **K**erninghan.*

awk [-f *script_file*] [-F*c*] [*script*] [-v *variable*=*value*...] [*file_list*]

awk. Script

script describes the filtering actions by lines of the form:

```
        condition { instructions }
```

*The **awk** utility handles the input files one line at a time and executes **instructions** when the **condition** is **true**. If **condition** is missing, then **instructions** are run for all lines in the files.*

awk. Condition

*- is a logical expression built with C **operators**: ||, &&, !, (). **Operands** can be arithmetic expressions, relational expressions, constants and variables. **Variables** must not be declared, they are automatically initialized, their type deducted from the context. For **strings** there is the concatenation operator (space) as well as some string functions. **Arrays** can be used, whose indices can be numerical values or strings.*

Predefined conditions:

BEGIN	it is true before the first line of the first file
END	it is true after the last line of the last file

awk. Instructions

- **variable=expression**
- instructions **if**, **for**, **while** as in C
- **;** is instructions separator
- the continuation of a line is done with the character \ on the last position in the line
- **for (i in array) instruction** <- is a repetitive structure in which i takes as values the array's index values and executes instruction for each value of i
- through the **expressions_list [>file_name]** is displayed at the standard output (or in the file specified by file_name) the value of the expressions_list separated by **OFS**, and with **ORS** at the end of the line.

awk. Predefined variables

NF	(*Number of Fields*) the number of words/fields in the current line
NR	(*Number of Records*) the number of the current processed line (the countdown starts at 1); line 1 is the first line of the first file
FNR	(*File Number of Records*) restarts from 1 at the beginning of each file
FS	(*Field Separator*) word/field separator
FILENAME	the name of the current file being processed
OFS	field separator at output (default is space)
ORS	record separator at output (default is new line)
ARGV	string of command line parameters
ARGC	the number of command line parameters

awk. Accessing fields

$0	the whole line that is being processed
$1	the first word/field on the current line being processed
$2	the second ...
	...
$NF	The last word/field on the current line

awk. Predefined functions

length(str)	the length of str; **length <=> length($0)**
substr(s,p,n)	substring of s which begins at position p and has the length n
index(s1,s2)	returns the position to which s2 appears in s1, or 0 otherwise
sprintf(format, arg1, ...)	returns the string that *printf* would print in C as a result
split(s,a,c)	where s is string, a is an array and c is a character. Splits the string s into fields, considering the character c as separator; if c is missing then the default separator **FS** is used. The obtained strings are given as values to the elements of the array a.

awk. Examples

```
awk '{print $1}' f1 # displays the first word on each line in the f1 file
awk -v v=Ion '{print "Hello ", v}' f1 # displays Hello Ion as many lines has f1

# displays all system users who do not have ... "password":
awk -F: '$2==""  { print $5}' /etc/passwd

# display the number of lines for each processed file (script only):
{F[FILENAME]++}
END {for (f in F) print f, ":", F[f]}
```

awk. Examples (cont.)

```
# display the number of characters for each processed file (script only):
{F[FILENAME]+=length($0)}
END {for (f in F) print f, ":", F[f]}

# display the lines that coincide from a file:
$0==v { if(NR>1) print $0 }
{ v=$0 }

# display all the words in a file and their number of occurrence:
{for(i=1; i<=NF; i++) X[$i]++}
END {for (c in X) print "the word ", c, " appears ", X[c], " times!" }
```

sort

sorts lexicographically the lines of a text file

sort [-cmudMnr] [-o *output*] [*file_list*]

sort. Options

-c	(*check*) checks whether the file is sorted or not
-m	(*merge*) interlaces the input files
-u	(*unique*) removes duplicate output lines
-d	(*dictionary*) compares only letters, numbers and space
-M	(*month*) compares the names of months; ex: “JAN” < “FEB”
-n	(*numeric*) compares the numerical lines
-r	(*reverse*) sorts in reverse order

sort. Examples (cont.)

```
# lists the contents of the current directory ordered lexicographically by group:
ls -l|sort -k8

# lists the contents of the current directory in numeric and descending order, by
size:
ls -l|sort -rnk5
```

uniq

*- report or omit repeated lines; **! does not detect repeated lines unless they are adjacent (sort the input first)***

uniq [-cdiu] [*input_file* [*output_file*]]

-c	(*count*) prefix lines by the number of occurrences;
-d	(*duplicate*) only print duplicate lines;
-i	(*ignore-case*) ignore differences between upper and lowercase when comparing;
-u	(*unique*) only print unique lines.

uniq. Examples

```
# when there is a file with duplicate lines that are not adjacent:
sort authors.txt | uniq

# indicates how many times a line has occurred:
uniq -c authors.txt

# show only repeated line; show only lines that do not repeat:
uniq -d authors.txt; uniq -u authors.txt

# show the lines of a file, sorted by the frequency with which they occur:
sort f | uniq -c | sort -n
```

wc

(word count) - count the characters, lines, or words in the input files

wc [-clw] [*file_list*]

-c	(*chars*) returns the number of characters/bytes
-l	(*lines*) returns the number of lines
-w	(*words*) returns the number of words

wc. Examples

```
# returns the number of lines, words, characters and the name of the file:
» wc myfile.txt
5 13 57 myfile.txt
» ls -1 | wc -l # the number of entries in the current directory
13
» wc -l /etc/passwd # the number of lines in /etc/passwd
65 /etc/passwd
» wc -w MyStory.txt # the number of words in /etc/passwd
185 MyStory.txt
# the number of processes currently running on your Linux system:
» ps -e | wc -l
118
```

head, tail

*- list **the first** lines, and **the last** lines in a file, respectively*

head [-n [-]*number_of_lines* | -c [-]*number_of_bytes*] [*file_list*]

tail [-n [+]*number_of_lines* | -c [+]*number_of_bytes*] [*file_list*]

head, tail. Examples

```
head /etc/passwd # displays only the first 10 lines
head -5 /var/log/yum.log # same as head -n5 /var/log/yum.log
head -c45 /var/log/yum.log # displays only the first 45 characters
echo "abcd123"|head -c-4 # displays everything except the last 4 chars

tail access.log # displays only the last 10 lines
cat /usr/share/dict/words | head -360 | tail -15 # get lines from 345 to 360
tail -n +2 a.txt # all lines in the a.txt file except for the first line
tail -c5 access.log
```

External process management commands

- **ps** - displays information about active processes in the system
- **kill** - emit an interrupt type signal to a process

- **tee** - obtaining a standard output witness file for a given command
- **nice** - changing the priority of the command
- **nohup** - executing a command with disconnect immunity (i.e., CTRL-d)

ps

- displays information about active processes in the system

ps [-efl] [-t *terminals*] [-U *user_list*] [-u *user_list*]

-e	(similar to -A) all active processes in the system (generic Unix/Linux format)
-f	display detailed information about active processes in the system
-l	long display format
-t	only processes launched from specific terminals
-U	selects processes with **real** owners from user_list
-u	selects processes with **actual** owners from user_list

ps. Output fields of the command

F	(*flags*) flags specified in <sys/proc.h> (for example **P_PPWAIT=10** means that the parent is waiting for the child to finish)
S or STAT	process state: **R** *runable/ready*, **S** *sleep*, **I** *idle (sleep > 20s)*, **T** *stopped*, **Z** *zombie*
UID	the owner ID
PID	the process ID
PPID	the parent process ID
C	the number of children of the process
PRI	the priority of the process

ps. Output fields of the command (cont.)

TTY	the terminal where the process was launched
TIME	the time it was served by the CPU
NICE or NI	if the priority was modified by the **nice** command
ADDR	the memory address of the process
SZ	process size
START or STIME	(*start*) the starting time of the process
CMD	the external form of the process launch command

ps. Examples

```
# display every active process in generic (Unix/Linux) format:
ps -e # similar to ps -A
# display all processes in BSD format:
ps au
ps aux
# to perform a full-format listing, add the -f or -F flag:
ps -ef
# view every process running with root user privileges (real & effective ID):
ps -U root -u root
# to select processes by the terminal (i.e., tty):
ps -t pst/0
ps -ft tty1
```

kill

- emits an interrupt type signal to a process specified by its Process Identifier (PID)

kill [*-signal*] PID

Some of the most used signals:

1	HUP (hang up)	**9**	KILL (non-ignorable kill)
2	INT (interrupt)	**14**	ALRM (alarm clock)
3	QUIT (quit)	**15**	TERM (software termination)
6	ABRT (abort)		

kill. Examples

```
# kills the process with PID 2145:
kill -9 2145
kill -KILL 2145
kill -s KILL 2145
kill -SIGKILL 2145
kill -s SIGKILL 2145
```

Supervision application

```
#!/bin/sh
DIR=${2-${HOME}}
t=${1-60}
x=`ls -l $DIR`
while true
do
    sleep $t
    y=`ls -l $DIR`
    if [ "$x" != "$y" ]
    then
        echo "The $DIR directory has been changed!"
        exit
    fi
    x=$y
done
```

The End