Linux notes

line continuation , multiline command. Put ‘\’ at the end of each line.

conf.d -> dump directory appends to config file

unset <varname> - removes linux exported variable

` - executes as command anything between backticks

source – executes scripts in current shell, shortcut is (.)

linux module ~ windows device driver

modprobe – install modules

pgreg – process grep

NOTE – move away from ifconfig and port commands, Use IP toolset in linux

IP A – check network

IP route – check routes

netns – namespace

TCP port 22 is SSH

file command – safe to see if it’s safe to cat, tells what type of file it is.

fdisk – list partitions

start of command + tab tab shows all the commands available

md5sum – check sum

watch –n # <command>, repeats command at an interval

!! – repeats last command, you can also add parameters to it. For example if last command was ls, you could do !! –lt…

check out screen command

for n in {10..14}; do ssh [openstack@192.168.2.${n}](mailto:openstack@192.168.2.$%7bn%7d)

“sudo scp [openstack@192.168.2.14:/etc/swift/\*.gz](mailto:openstack@192.168.2.14:/etc/swift/*.gz) /etc/swift”; done

**awk**

cat test.txt | awk '/toby/ {n++} {print > "test"n".out"}'

So it reads in a file and breaks the file up on a keyword:

This is the contents of test.txt

toby

this is a test

robbie

toby

to try out something

robbie

toby

to see if it works

Robbie

The command above breaks the file at toby and creates a file for each break:

**SED**

remove all double-spaces from text:                     s/  \*/ /g

ls \*[0-9] | sed 's/\(.\*pmt\)\(.\*\)/mv \-i \1\2 \1q\2/g' | sh

This would take a file for example:  tobypmt12345

This is the output (adds the letter q to the middle of the filename):

mv -i tobypmt12345 tobypmtq12345

=========================================================================

**ls -d to\_afnt\* | sed "s/\(to\_afnt\)\(.\*\)\$/mv -i '&' 'afnt9\2'/"**  <-- Sed substitution breaks down input into groups using **()**, you can then refer to those groups in the replace portion using **\1 , \2, \3...\9**.  & refers to the whole input string.  This example creates the following move statements:

mv -i 'to\_afnt.cfg' 'afnt9.cfg'  
mv -i 'to\_afnt.sc' 'afnt9.sc'

You can execute these statements by piping to sh.

=========================================================================

I have been cutting and pasting a lot of C programs from my mindleaders class to run them.  Vi always does the fun inserting tabs like crazy when I paste the file.  After getting tired of manually fixing it I decided to try and find a better way.

Here are 2 sed commands I have found helpful: (Use these from within VI, same format applies to line command.)

**%s/^[    ]\*//g   - Sed does not recognize '\t' inside the [ ] is actually [<space><tab>]; This will strip all spaces and tabs from the beginning of every line in the file.**

**1,5s/^[    ]\*//g - If you don't want to do the whole file use this version.  This will do the same thing as above but only to lines 1-5.**

=========================================================================

sed '/src-collapsed/,/))/d;/dest-collapsed/,/))/d;/collapsed-rules/,/))/d'  x\_adt\_afnt\_to\_omni.tsc | less

grep -e "\- ~" -e '\- "' x\* | grep -v "(copy" | less

=========================================================================

Sed if /else

IF/ELSE TESTING IN SED

----------

This message was originally posted on the seders mailing list in 1998.

It has been lightly edited for general readers. In answer to the

question:

> Are there any standard/short-cuts for

>

>     if (test) then action1 else action2

Yes, there are several ways of expressing IF/ELSE logic. For example:

  # -------------------------------------------

  # one-line actions, for if (test) then action1, else action2

  # -------------------------------------------

  /test/s/$/ action1/;     # if /test/ is found, append action1 to EOL

  /test/!s/$/ action2/;    # if /test/ not found, append action2

  /test2/d;                # if /test2/ is found, delete line. The

                           # implied ELSE is to print the line.

  /test3/!y/ABCDE/abcde/;  # if /test3/ is missing, lowercase A-E.

                           # The implied ELSE is leave A-E alone.

  # -------------------------------------------

  # multi-line actions showing IF/ELSE usage

  # -------------------------------------------

  /test4/{                 # if /test4/ is found, ...

     s/$/aaa/;             # ... perform these actions

     s/[0-9]/number/;

     s/test5/YYY/;         # Boolean /test4/ && /test5/

  }

  /test4/!{                # if /test4/ is missing, ...

     s/^/bbb/;             # ... perform these instead

     s/[a-f]/letter/;

     s/test6/ZZZ/;         # Boolean /test4/! && /test6/

  }

  /test7/b next            # if /test7/ is found, skip the next cmds

    s/$/new tail/;         # else: 1) add a new ending to each line

    /^/a\                  #       2) and append new line after each

    APPENDED WORDS AFTER EACH LINE

    /test8/d;              #       3) and delete each line with /test8/

  : next

  # Next routine will fail under GNU sed 2.05, due to a bug

  s/test9/&/6;             # if /test9/ appears 6 times or more,

  t next2                  # ... jump to label :next2 for commands

  cmd1;cmd2;cmd3;          # else, do these 3 commands

  b next3                  # the ELSE stops here

  : next2                  # the next 3 commands are executed only

  cmd4;cmd5;cmd6;          # ... if /test9/ was found 6 times

  : next3                  # this corresponds to ENDIF

  /test9/ { /test10/ {     # Boolean IF /test9/ && /test10/ are true,

    cmd1; cmd2; cmd3;      # ... do these 3 commands

    b next4

    }                      # ELSEIF /test9/ && /test10/! are true,

    cmd4; cmd5; cmd6;      # ... do cmd4, cmd5, and cmd6

    b next4

  }                        # ELSE,

  cmd7; cmd8; cmd9;        # ... do cmd7, cmd8, and cmd9

  : next4                  # This corresponds to ENDIF

Here you go.....

sort -t\| +1 -2 +0 -1  ---> sort command (fields separated by "|") +1 is the field and -2 is the position.

-------------------------------------------------------------------------

HANDY ONE-LINERS FOR SED (Unix stream editor)               Oct. 29, 1997

compiled by Eric Pement <[epement@jpusa.chi.il.us](mailto:epement@jpusa.chi.il.us)>             version 4.3

Latest version of this file is always at <[http://www.wollery.demon.co.uk](http://www.wollery.demon.co.uk/)>

FILE SPACING:

# double space a file

sed G

# triple space a file

sed 'G;G'

# undo double-spacing (assumes even-numbered lines are always blank)

sed 'n;d'

NUMBERING:

# number each line of a file (simple left alignment). Using a tab (see

# note on '\t' at end of file) instead of space will preserve margins.

sed = filename | sed 'N;s/\n/\t/'

# number each line of a file (number on left, right-aligned)

sed = filename | sed 'N; s/^/     /; s/ \*\(.\{6,\}\)\n/\1  /'

# number each line of file, but only print numbers if line is not blank

sed '/./=' filename | sed '/./N; s/\n/ /'

# count lines (emulates "wc -l")

sed -n '$='

TEXT CONVERSION AND SUBSTITUTION:

# IN UNIX ENVIRONMENT: convert DOS newlines (CR/LF) to Unix format

sed 's/.$//'

# IN DOS ENVIRONMENT: convert Unix newlines (LF) to DOS format

sed 's/$//'                          # method 1

sed -n p                             # method 2

# delete leading whitespace (spaces, tabs) from front of each line

# aligns all text flush left

sed 's/^[ \t]\*//'                    # see note on '\t' at end of file

# delete trailing whitespace (spaces, tabs) from end of each line

sed 's/[ \t]\*$//'                    # see note on '\t' at end of file

# delete BOTH leading and trailing whitespace from each line

sed 's/^[ \t]\*//;s/[ \t]\*$//'

q

# insert 5 blank spaces at beginning of each line (make page offset)

sed 's/^/     /'

# align all text flush right on a 79-column width

sed -e :a -e 's/^.\{1,78\}$/ &/;ta'  # set at 78 plus 1 space

# center all text in the middle of 79-column width. In method 1,

# spaces at the beginning of the line are significant, and trailing

# spaces are appended at the end of the line. In method 2, spaces at

# the beginning of the line are discarded in centering the line, and

# no trailing spaces appear at the end of lines.

sed  -e :a -e 's/^.\{1,77\}$/ & /;ta'                     # method 1

sed  -e :a -e 's/^.\{1,77\}$/ &/;ta' -e 's/\( \*\)\1/\1/'  # method 2

# substitute (find & replace) "foo" with "bar" on each line

sed 's/foo/bar/'             # replaces only 1st instance in a line

sed 's/foo/bar/4'            # replaces only 4th instance in a line

sed 's/foo/bar/g'            # replaces ALL instances in a line

# substitute "foo" with "bar" ONLY for lines which contain "baz"

sed '/baz/s/foo/bar/g'

# substitute "foo" with "bar" EXCEPT for lines which contain "baz"

sed '/baz/!s/foo/bar/g'

# reverse order of lines (emulates "tac")

sed '1!G;h;$!d'

# reverse each character on the line (emulates "rev")

sed '/\n/!G;s/\(.\)\(.\*\n\)/&\2\1/;//D;s/.//'

# join pairs of lines side-by-side (like "paste")

sed 'N;s/\n/ /'

SELECTIVE PRINTING OF CERTAIN LINES:

# print first 10 lines of file (emulates behavior of "head")

sed 10q

# print first line of file (emulates "head -1")

sed q

# print last 10 lines of file (emulates "tail")

sed -e :a -e '$q;N;11,$D;ba'

# print last line of file (emulates "tail -1")

sed '$!d'

# print only lines which match regular expression (emulates "grep")

Now add the word DRAFT to the end of each line in the file doc.

  $ cat doc

  These instructions are for using

  version 7.1 of our software.

  $ sed 's/$/ DRAFT/g' doc > doc.draft

sed -n '/regexp/p'           # method 1

sed '/regexp/!d'             # method 2

**# print only lines which do NOT match regexp (emulates "grep -v")**

**sed -n '/regexp/!p'          # method 1, corresponds to above**

**sed '/regexp/d'              # method 2, simpler syntax**

# print 1 line of context before and after regexp, with line number

# indicating where the regexp occurred (similar to "grep -A1 -B1")

sed -n -e '/regexp/{=;x;1!p;g;$!N;p;D;}' -e h

**# grep for AAA and BBB and CCC (in any order)**

**sed '/AAA/!d; /BBB/!d; /CCC/!d'**

**# grep for AAA or BBB or CCC (emulates "egrep")**

**sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d**

# print only lines of 65 characters or longer

sed -n '/^.\{65\}/p'

# print only lines of less than 65 characters

sed -n '/^.\{65\}/!p'        # method 1, corresponds to above

sed '/^.\{65\}/d'            # method 2, simpler syntax

# print section of file from regular expression to end of file

sed -n '/regexp/,$p'

# print section of file based on line numbers (lines 8-12, inclusive)

sed -n '8,12p'               # method 1

sed '8,12!d'                 # method 2

# print line number 52

sed -n '52p'                 # method 1

sed '52!d'                   # method 2

sed '52q;d'                  # method 3, efficient on large files

# print section of file between two regular expressions (inclusive)

sed -n '/Iowa/,/Montana/p'             # case sensitive

SELECTIVE DELETION OF CERTAIN LINES:

# print all of file EXCEPT section between 2 regular expressions

sed '/Iowa/,/Montana/d'

# delete duplicate lines from a sorted file (emulates "uniq"). First

# line in a set of duplicate lines is kept, the rest are deleted

sed '$!N; /^\(.\*\)\n\1$/!P; D'

# delete ALL blank lines from a file (same as "grep '.' ")

sed '/^$/d'

# delete all CONSECUTIVE blank lines from file except the first; also

# deletes all blank lines from top and end of file (emulates "cat -s")

sed '/./,/^$/!d'          # method 1, allows 0 blanks at top, 1 at EOF

sed '/^$/N;/\n$/D'        # method 2, allows 1 blank at top, 0 at EOF

# delete all CONSECUTIVE blank lines from file except the first 2:

sed '/^$/N;/\n$/N;//D'

# delete all leading blank lines at top of file

sed '/./,$!d'

# delete all trailing blank lines at end of file

sed -e :a -e '/^\n\*$/N;/\n$/ba'

SPECIAL APPLICATIONS:

# remove nroff overstrikes (char, backspace) from man pages

sed "s/.`echo **[Error! Hyperlink reference not valid.](file:///\\\\b%60\\g)**"    # double quotes required for Unix environment

sed 's/.\x08//g'           # hex expression for GNU sed (octal is "\010")

# get Usenet/e-mail message header

sed '/^$/q'                # deletes everything after first blank line

# get Usenet/e-mail message body

sed '1,/^$/d'              # deletes everything up to first blank line

# get Subject header, but remove initial "Subject: " portion

sed '/^Subject: \*/!d; s///;q'

# get return address header

sed '/^Reply-To:/q; /^From:/h; /./d;g;q'

# parse out the address proper. Pulls out the e-mail address by itself

# from the 1-line return address header (see preceding script)

sed 's/ \*(.\*)//; s/>.\*//; s/.\*[:<] \*//'

# add a leading angle bracket and space to each line (quote a message)

sed 's/^/> /

# delete leading angle bracket & space from each line (unquote a message)

sed 's/^> //'

# remove most HTML tags (accommodates multiple-line tags)

sed -e :a -e 's/<[^<]\*>/ /g;/</{N;s/\n/ /;ba;}'

# extract multi-part uuencoded binaries, removing extraneous header

# info, so that only the uuencoded portion remains. Files passed to

# sed must be passed in the proper order. Version 1 can be entered

# from the command line; version 2 can be made into an executable

# Unix shell script. (Modified from a script by Rahul Dhesi.)

sed '/^end/,/^begin/d' file1 file2 ... fileX | uudecode   # vers. 1

sed '/^end/,/^begin/d' $\* | uudecode                      # vers. 2

# zip up each .TXT file individually, deleting the source file and

# setting the name of each .ZIP file to the basename of the .TXT file

# (under DOS: the "dir /b" switch returns bare filenames in all caps).

echo @echo off >zipup.bat

dir /b \*.txt | sed "s/^\(.\*\)\.TXT/pkzip -mo \1 \1.TXT/" >>zipup.bat

TYPICAL USE: Sed takes one or more editing commands and applies all of

them, in sequence, to each line of input. After all the commands have

been applied to the first input line, that line is output and a second

input line is taken for processing, and the cycle repeats. The

preceding examples assume that input comes from the standard input

device (i.e, the console, normally this will be piped input). One or

more filenames can be appended to the command line if the input does

not come from stdin. Output is sent to stdout (the screen). Thus:

cat filename | sed '10q'        # uses piped input

sed '10q' filename              # same effect, avoids a useless "cat"

sed '10q' filename > newfile    # redirects output to disk

For additional syntax instructions, including the way to apply editing

commands from a disk file instead of the command line, consult "sed &

awk, 2nd Edition," by Dale Dougherty and Arnold Robbins (O'Reilly,

1997; [http://www.ora.com](http://www.ora.com/)), "UNIX Text Processing," by Dale Dougherty

and Tim O'Reilly (Hayden Books, 1987) or the tutorials by Mike Arst

distributed in U-SEDIT2.ZIP (many sites). To fully exploit the power

of sed, one must understand "regular expressions." For this, see

"Mastering Regular Expressions" by Jeffrey Friedl (O'Reilly, 1997).

The manual ("man") pages on Unix systems may be helpful (try "man

sed", "man regexp", or the subsection on regular expressions in "man

ed"), but man pages are notoriously difficult. They are not written to

teach sed use or regexps to first-time users, but as a reference text

for those already acquainted with these tools.

QUOTING SYNTAX: The preceding examples use single quotes ('...')

instead of double quotes ("...") to enclose editing commands, since

sed is typically used on a Unix platform. Single quotes prevent the

Unix shell from intrepreting the dollar sign ($) and backquotes

(`...`), which are expanded by the shell if they are enclosed in

double quotes. Users of the "csh" shell and derivatives will also need

to quote the exclamation mark (!) with the backslash (i.e., \!) to

properly run the examples listed above, even within single quotes.

Versions of sed written for DOS invariably require double quotes

("...") instead of single quotes to enclose editing commands.

USE OF '\t' IN SED SCRIPTS: For clarity in documentation, we have used

the expression '\t' to indicate a tab character (0x09) in the scripts.

However, most versions of sed do not recognize the '\t' abbreviation,

so when typing these scripts from the command line, you should press

the TAB key instead. '\t' is supported as a regular expression

metacharacter in awk, perl, and in a few implementations of sed.

VERSIONS OF SED: Versions of sed do differ, and some slight syntax

variation is to be expected. In particular, most do not support the

use of labels (:name) or branch instructions (b,t) within editing

commands, except at the end of those commands. We have used the syntax

which will be portable to most users of sed, even though the popular

GNU versions of sed allow a more succinct syntax. When the reader sees

a fairly long command such as this:

   sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d

it is heartening to know that GNU sed will let you reduce it to:

   sed '/AAA/b;/BBB/b;/CCC/b;d'

In addition, remember that while many versions of sed accept a command

like "/one/ s/RE1/RE2/", some do NOT allow "/one/! s/RE1/RE2/", which

contains space before the 's'. Omit the space when typing the command.

OPTIMIZING FOR SPEED: If execution speed needs to be increased (due to

large input files or slow processors or hard disks), substitution will

be executed more quickly if the "find" expression is specified before

giving the "s/.../.../" instruction. Thus:

   sed 's/foo/bar/g' filename         # standard replace command

   sed '/foo/ s/foo/bar/g' filename   # executes more quickly

   sed '/foo/ s//bar/g' filename      # shorthand sed syntax

On line selection or deletion in which you only need to output lines

from the first part of the file, a "quit" command (q) in the script

will drastically reduce processing time for large files. Thus:

   sed -n '45,50p' filename           # print line nos. 45-50 of a file

   sed -n '51q;45,50p' filename       # same, but executes much faster

awk cool

tr '\r' '\n' < ji\_affinity,Feb.27:12AM  | awk -F"|" '$1 ~ /MSH/ {print $1 "|" $7 "|" $9}; $1 ~ /PID/ {print $1 "|" $6 "|" $4 "|" $19}' | sed 'N;s/\n/|/' | less

-----Original Message-----

**From:** Fulper Sean

**Sent:** Thursday, May 06, 2004 4:25 PM

**To:** Matherly Toby

**Subject:** RE: a VERY cool AWK command

Ah, good point...this will actually d(elete) the messages that begin with "MSH|"...(a neat twist)...

-Sean

-----Original Message-----

**From:** Matherly Toby

**Sent:** Thursday, May 06, 2004 4:14 PM

**To:** Fulper Sean

**Subject:** RE: a VERY cool AWK command

I thought it was in there.  Here is the sed version:

sed '/^MSH|/d' ew\_i\_afnt.dat

-----Original Message-----

**From:** Fulper Sean

**Sent:** Thursday, May 06, 2004 4:08 PM

**To:** Matherly Toby; Kellems Lee

**Subject:** a VERY cool AWK command

**Importance:** High

awk '$0 !~ /^MSH|/' ew\_i\_afnt.dat

will display ALL messages that do NOT start with "MSH|"!!!!

============================================================================================

sed '/Host/s/$/,/' stupid | sed '/Port/s/$/ @/' | tr '\n' '\0' | tr '@' '\n'

============================================================================================

# then deletes all spaces at the head of each line

#

sed 's/^ \*//g'   |\

#

# then changes all multiple spaces to single spaces

#

sed 's/  \*/ /g' |\

#

# then removes everything between "{" and "}" (generally editorial matter)

#

sed 's/\{.\*\}//g' |\

#

# then removes everything between "[" and "]" (generally editorial matter)

#

sed 's/\[.\*\]//g' |\

#

# then removes everything between "|" and "|"

#

sed 's/\|.\*\|//g' |\

#

# then removes most punctuation as well as "0", "Z", and "X"

#

sed 's/[\.\,\:\;\"\!\?\-\\_\(\)\&\{\}\/0zZxX]\*//g' |\

#

# then removes apostrophes

#

sed "s/\'//g" |\

#

# then translates numerals into letters

#

sed 'y/123456789/abcdefghi/' |\

#

# then we go through again and remove extra spaces

#

sed 's/^ \*//g'   |\

sed 's/  \*/ /g' |\

#