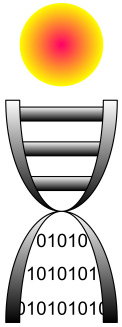
Processing Text with sed

Dr. Marjan Trutschl
marjan.trutschl@lsus.edu



LABi: Laboratory for Advanced Biomedical Informatics
Louisiana State University, Shreveport, LA 71115
labi.cs.lsus.edu .:. info@labi.cs.lsus.edu

Topics

- Getting an installing sed
- Methods of invoking sed
- Selecting lines to operate on
- Performing substitutions with sed
- Advanced sed invocation
- Advanced addressing
- Common one-line sed scripts

Introducing Sed

- It is a non-interactive stream editor designed by the late Lee E. McMahon in 1973 or 1974
- Sed is shell independent i.e. it works with whatever shell you want to use it with.
- Instead of altering a file interactively by moving the cursor on the screen (as with a word processor), the user sends a script of editing instructions to sed, plus the name of the file to edit (or the text to be edited may come as output from a pipe).

Introducing Sed

- In this sense, sed works like a filter -- deleting, inserting and changing characters, words, and lines of text. Its range of activity goes from small, simple changes to very complex ones.
- Most people use sed for its substitution features.
- Sed is often used as a find-and-replace tool.

Sed Versions

- It comes standard with nearly every Unix that exists, including Linux and Mac OS X.
- It is an essential shell command and generally does not need to be installed.
- There are free versions, shareware versions and commercial versions.
- The most common version is arguably GNU sed. For current version, type: `sed --version`
- The GNU sed has a number of extensions that the POSIX sed does not have. GNU vs. POSIX (LTR)

Sed Versions

- BSD implementations offer extensions that support the Japanese language.
- ssed (super sed) has more features than GNU sed and is based on the GNU sed code-base.
- Sed is generally found at /bin/sed or /usr/bin/sed

Sed Versions

- To find out what version you have on your sytem, type the following command

```
$ sed --version <Enter>
GNU sed version 4.1.4
Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying
  conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE,
to the extent permitted by law.
```

Installing Sed

- If you don't have a version of sed installed, get a version targeted for your OS
- Mac OS X comes with a BSD version of sed, but you can easily install the GNU version through fink (http://fink.sourceforge.net)
- On Debian GNU/Linux you can install sed as root by typing

```
apt-get install sed
```

Bootstrap Installation

- If you are building sed on a system that has no preexisting version of sed, you need to follow a bootstrap procedure outlined in README.boot
- NOTE: If you have the BSD version, you can directly jump to configuring and installing sed
- To bootstrap the building of sed, you run the shell script bootstrap.sh

```
$ sh ./bootstrap.sh
```

Bootstrap Installation

- If you get errors and the bootstrap version of sed fails to build, you will need to edit the config.h header file that was created for your system.
- Once the build is completed, copy the sed binary that was built in the sed directory to somewhere in your *$PATH*.

Configuring and Installing sed

- Please go thought text and commands in text book on page 193

```
$ sh ./configure
$ make
$ su
```

How sed Works

- Sed reads its input from stdin (Unix shorthand for "standard input," i.e., the console) or from files (or both), and sends the results to stdout ("standard output," normally the console or screen)
- You can also redirect this output to a file.
- Sed doesn't typically modify an original input file, instead you send the contents of your file through a pipe to be processed by sed.

Invoking Sed

- Before you can get started with any of the examples, you will need some data to work with.
- The /etc/passwd file, contains some useful data to parse with sed. Everyone may have a slightly different file, so results may vary.
- Sed can be invoked by sending data through a pipe to it. Try the following command at the prompt

```
$ cat /etc/passwd | sed <Enter>
```

Invoking Sed

- This command dumps the contents of the /etc/passwd to sed through the pipe into sed's pattern space.
- The "pattern space" is the internal work buffer that sed uses to do its work.
- Sed expects to always do something with its pattern space, and if you don't tell it what to do, it considers that an invocation error.
- The previous command was incorrectly invoked so we got sed's command usage as an output.

NOTE

- Create a file ‘sedfile’ for all examples in the book. So instead of using the file /etc/passwd, use sedfile. The following is the content of the sedfile.

```
1 -- This is line 1
2 -- This is line 2
3 -- This is line 3
4 -- This is line 4
5 -- This is line 5
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
10 -- This is line 10
```

Editing Commands

- Sed expects an editing command, i.e. what you want sed to do to the data in the pattern space.
- Try the following at the prompt, which deletes all lines with sed

```
$ cat sedfile | sed 'd' <Enter>
```

- It didn't print anything at all…this is because sed read the first line from sedfile into its pattern buffer. It then performed the *delete line* editing command on the contents of its pattern buffer and then printed out the pattern buffer.

Editing Commands

- Because the command deleted the line, the pattern buffer was empty and so nothing got printed out.
- Sed then repeats the process until it reaches the end of the file.
- NOTE: The original sedfile file is not altered.

Invoking sed with the –e flag

- Try the command at the prompt

```
$ sed -e 'd' sedfile <Enter>
$
```

- It does the same as the previous command
- Invoking sed in this manner explicitly defines the editing command as a sed script to be executed on the input file sedfile.
- The script is simply a one-character editing command in this case.

Redirection

- Output from sed is to the standard output, but you can redirect it to an output file.
- You will have to use shell's I/O redirection capabilities to send standard output to a file. Use the operator '>' to redirect your output as in:

```
$ sed -e 'd' sedfile > /tmp/new sedfile <Enter>
$ sed -e 'd' sedfile > /tmp/new\ sedfile <Enter>
$ sed -e 'd' sedfile > "/tmp/new sedfile" <Enter>
```

- Because this command deletes all lines, this results in an empty output file.

The –n, --quiet and --silent Flags

- By default, sed prints out the pattern space at the end of processing its editing commands and then repeats that process.
- The –n flag disables this automatic printing so that sed will instead print lines only when it is explicitly told to do so with the ‘p’ command.
- The ‘p’ command simply means to print the pattern space.

The –n, --quiet and --silent Flags

- The ‘p’ flag is generally used only in conjunction with the ‘-n’ flag; otherwise you will end up printing the pattern space twice.
- Try the following commands at the prompt

```
$ cat sedfile | sed ‘p’ | head -10 <Enter>
$ cat sedfile | sed –n ‘p’ | head -10 <Enter>
```

- The first command will print the pattern space twice i.e. you will have duplicate lines, while the second command will print it just once.
- See the output on the next slide.

The –n, --quiet and --silent Flags

```
$ cat sedfile | sed 'p' | head -10
1 -- This is line 1
1 -- This is line 1
2 -- This is line 2
2 -- This is line 2
3 -- This is line 3
3 -- This is line 3
4 -- This is line 4
4 -- This is line 4
5 -- This is line 5
5 -- This is line 5
```

The –n, --quiet and --silent Flags

```
$ cat sedfile | sed -n 'p' | head -10
1 -- This is line 1
2 -- This is line 2
3 -- This is line 3
4 -- This is line 4
5 -- This is line 5
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
10 -- This is line 10$
```

Sed Errors

- It is easy to incorrectly specify your sed editing commands, as the syntax requires attention to detail.
- If you miss one character, you can produce vastly different results than expected, or find yourself faced with a rather cryptic error message
- Sed is not friendly with its error messages

Selecting Lines to Operate On

- Sed understands something called ‘addresses’.
- Addresses are either particular locations in a file or a range where a particular editing command should be applied.
- When sed encounters no addresses, it performs its operations on every line in the file.
- The following command adds a basic address to the sed command

```
$ cat sedfile | sed ‘1d’ | more <Enter>
```

Selecting Lines to Operate On

```
$ cat sedfile | sed ‘1d’ <Enter>
2 -- This is line 2
3 -- This is line 3
4 -- This is line 4
5 -- This is line 5
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
```

Selecting Lines to Operate On

- The number 1 before the delete edit command tells sed to perform the editting command on the first line of the file.
- So, in the above example, sed will delete the first line of the file and print the rest.
- ‘more’ will just print the output, one page at a time

Address Ranges

- If you want to remove more than one line, you can use address ranges, as in:

```
$ cat sedfile | sed '1,5d' <Enter>
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
10 -- This is line 10
```

- In the above examples sed, deletes the lines 1 through 5 (1 and 5 inclusive)

Address Ranges

- If you use reverse address ranges as in:

```
$ cat sedfile | sed '10,4d' <Enter>
```

- In the above example because you've told sed to start deleting from line 10, it reads in line 10, deletes it and then looks for the range 9-4 to delete.
- However, it won't see those lines after it has reached line 10.
- Sed does not back up in its processing to look for those lines, so line 10 is deleted, but nothing else.

Address Ranges

- If you forget to complete your address range, you will receive an error message from sed, which may not be very helpful e.g.

```
$ cat /etc/passwd | sed '1,d‘ <Enter>
sed: -e expression #1, char 3: unexpected `,'
```

Address Ranges

- If you want to match line 4 and the five lines following line 4, you append a plus sign before the second address number

```
$ cat sedfile | sed '4,+5d' <Enter>
```

- This will match line 4 in the file, delete that line, continue to delete the next five lines, and then cease its deletion and print the rest.
- Output on next page

Address Ranges

```
$ cat sedfile | sed '4,+5d‘ <Enter>
1 -- This is line 1
2 -- This is line 2
3 -- This is line 3
10 -- This is line 10
```

Address Negation

- You can negate an address match by appending an exclamation mark at the end of any address specification.
- So, only those lines that do not match the address match will be matched.
- The following 2 slides list 2 examples with their respective outputs

Example: Address Negation

```
$ cat /etc/passwd | sed '1,5!d‘ <Enter>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
```

- Here sed matches everything except the first five lines and performs the deletion.

Example: Address Negation

```
$ cat /etc/passwd | sed '1,10!d'
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
games:x:12:100:Games account:/var/games:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
```

- Here sed matches everything except the first ten lines and performs the deletion.

Address Steps

- This allows you to do things such as selecting every odd line, 3rd line, 5th line etc.
- You use a tilde (~) to specify address steps.

```
$ cat sedfile | sed '1~3d'
```

- This deletes the first line, steps over the next two lines, and then deletes the fourth line. Sed continues this pattern till it reaches the end of the file.
- Output on next slide.

Address Steps

```
$ cat sedfile | sed '1~3d'
2 -- This is line 2
3 -- This is line 3
5 -- This is line 5
6 -- This is line 6
8 -- This is line 8
9 -- This is line 9
```

Address Steps

```
$ cat sedfile | sed '1~1d' <Enter>
```

- This tells sed to delete all lines.

```
$ cat sedfile | sed '2~2d' <Enter>
```

- This tells sed to delete the second line, step over the next line, delete the next line and repeat until end of file is reached.

Substitution

- The substitution command, denoted by s, will substitute any string that you specify with any other string that you specify.
- To substitute one string with another, you need to tell sed, where the 1st string ends and where the 2nd begins.
- This is done with the forward slash (/) character.
- The command substitutes only the first occurrence of the string on every line.

Example: Substitution

```
$ cat sedfile | sed 's/1/one/‘ <Enter>
one -- This is line 1
2 -- This is line 2
3 -- This is line 3
4 -- This is line 4
5 -- This is line 5
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
one0 -- This is line 10
```

Example: Substitution

- If you want to substitute every occurrence of the string on every line, do a global substitution. Add the letter ‘g’ to the end of the command.

```
$ cat sedfile | sed 's/1/one/g‘ <Enter>
one -- This is line one
2 -- This is line 2
3 -- This is line 3
4 -- This is line 4
5 -- This is line 5
6 -- This is line 6
7 -- This is line 7
8 -- This is line 8
9 -- This is line 9
one0 -- This is line one0
```

Substitution Flags

Flag	Meaning
g	Replace all matches, not just the first match
NUMBER	Replace only the NUMBERth match
p	If substitution was made, print pattern space
w FILENAME	If substitution was made, write result to FILENAME. GNU sed additionally allows writing to /dev/stderr and /dev/stdout
I or i	Match in a case-insensitive manner
M or m	In addition to the normal behavior of the special regular expression characters ^ and $, this flag causes ^ to match the empty string after a newline and $ to match the empty string before a newline.

Example: Substitution

- If you specify any number as a flag (NUMBER flag), this tells sed to act on the instance of the string that matched that number.

```
$ cat /etc/passwd | sed 's/root/toor/3' | head -2 <Enter>
root:x:0:0:root:/toor:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
```

- The /etc/passwd file has 3 instances of ‘root’ in the first line, so the above command replaces only the 3rd match on

Using an Alternative String Separator

- If you have to do a substitution on a string that includes the forward slash character, specify a different separator by providing the designated character after the ‘s’.

```
$ cat /etc/passwd | sed 's:/root:/toor:' | head -2 <Enter>
root:x:0:0:root:/toor:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
```

- In the above example we have changed the home directory of the root user from ‘/root’ to ‘/toor’. To do this we used a colon (:) as a separator.

Using an Alternative String Separator

- If you find yourself in a situation where you do need to use the string separator, you can do so by escaping the character.

```
$ cat /etc/passwd | sed 's/\/root/\/toor/' | head -2 <Enter>
root:x:0:0:root:/toor:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash

$ cat /etc/passwd | sed 's/:root/:absolutely power corrupts/g'
  | head -2 <Enter>
root:x:0:0:absolutely power corrupts:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
```

Replacing with Empty Spaces

- Use an empty substitution string to delete the root string from the /etc/passwd file entirely.

```
$ cat /etc/passwd | sed 's/root//g' | head -2 <Enter>
:x:0:0::/:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
```

- The ‘s/root//g’ tells sed to replace all instances of root with the empty replacement string that follows the separator.

Address Substitution

- It is possible to perform substitution only on specific lines or on a specific range of lines if you specify an address or an address range to the command.

```
$ cat /etc/passwd | sed '10s/sh/quiet/g' | head -10 <Enter>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
games:x:12:100:Games account:/var/games:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/baquiet
```

Address Substitution

- In the previous example, notice the string ‘baquiet’ on line 10. i.e. ‘bash’ is now ‘baquiet’ after the substitution.
- To do an address range, try the following

```
$ cat /etc/passwd | sed '1,5s/sh/quiet/g' | head -6 <Enter>
root:x:0:0:root:/root:/bin/baquiet
bin:x:1:1:bin:/bin:/bin/baquiet
daemon:x:2:2:Daemon:/sbin:/bin/baquiet
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/baquiet
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
```

Advanced sed Invocation

- You can specify multiple editing commands on the command line in three different ways.
- Create a file stream.txt containing the following text

```
Imagine a quaint bubbling stream of cool mountain water filled
with rainbow trout and elephants drinking iced tea.
```

- The first way, to specify multiple editing commands, on the command line, is to separate them by a semicolon.

```
$ cat stream.txt | sed 's/trout/catfish/; s/ and elephants//‘
  <Enter>
Imagine a quaint bubbling stream of cool mountain water filled
with rainbow catfish drinking iced tea.
```

Advanced sed Invocation

- The second way to specify multiple editing commands, on the command line is to specify multiple –e arguments.
- The following example (with output) will show how.

```
$ cat stream.txt | sed -e 's/trout/catfish/' -e 's/ and
  elephants//‘ <Enter>
Imagine a quaint bubbling stream of cool mountain water
filled with rainbow catfish drinking iced tea.
```

Advanced sed Invocation

- The third way to specify multiple editing commands, on the command line is to use the multiline capability of the bash shell
- The following examples (with output) will show how.

```
$ cat stream.txt | sed ‘ <Enter>
> s/trout/catfish/ <Enter>
> s/ and elephants//‘ <Enter>
Imagine a quaint bubbling stream of cool mountain water
filled with rainbow catfish drinking iced tea.
```

Advanced sed Invocation

- In the previous command, bash knows when you have not terminated a single quote and prompts you for more input until you enter the completing single quote.

Sed scripts

- Create a text file called water.sed containing the following text/commands

```
s/trout/catfish/
s/ and elephants//
```

- The file contains 2 editing commands.
- To specify the file containing the editing commands you want sed to perform, pass the ‘-f’ flag followed by the filename containing the editing commands.
- The example on the next slide shows how.

Sed scripts

- We execute the commands using the stream.txt file

```
$ sed -f water.sed stream.txt
Imagine a quaint bubbling stream of cool mountain water
filled with rainbow catfish drinking iced tea.
```

The Comment Command

- To add a comment in a sed script, simply precede the line with the pound (#) character.
- The comment continues till the next newline.
- There are 2 caveats with the comment command
  - Comments are not portable to non-POSIX versions of sed
  - If the first two characters of your script are #*n*, the –n (no auto-print) option is automatically enabled. In this case, either use ‘N’ or place a space between the # and ‘n’

The Insert, Append and Change Commands

- Insert (i) outputs the text immediately, before the next command
- Append (a) outputs the text immediately afterward.
- In the next example, we convert a text file to an HTML file. Using ‘i’ and ‘a’ you can create a simple sed file that will add the opening and closing tags to any text file.

Example: Insert, Append Commands

- Save the following in a file named txt2html.sed

```
#!/bin/sed -f

1 i\
<html>\
<head><title>Converted with sed</title></head>\
<body bgcolor="#ffffff">\
<pre>\

$ a\
</pre>\
</body>\
</html>
```

Example: Output

- Take stream.txt and run it through the sed script.

```
$ cat stream.txt | sed -f txt2html.sed <Enter>
<html>
<head><title>Converted with sed</title></head>
<body bgcolor="#ffffff">
<pre>

Imagine a quaint bubbling stream of cool mountain water
filled with rainbow trout and elephants drinking iced
tea.
</pre>
</body>
</html>
```

Example: Output Explanation

- Sed inserted, starting at line 1, the four opening HTML tags that indicate that the file is HTML.
- Then it printed the text file
- At the end of the file (denoted by the $), sed appended the closing HTML tags

The Change Command

- The change (c) command replace the current line in the pattern space with the text that you specify.
- The difference between the substitute (s) and change (c) commands is that ‘s’ works on a character-by-character basis, whereas the ‘c’ changes the entire line.
- In the example, change ‘*water.sed*’ to the following and name the file ‘*noelephants.sed*’

```
s/trout/catfish/
/ and elephants/ c\Although you can substitute trout with
catfish, there is no substitute for elephants, so we
cannot offer this item
```

Example: The Change Command

- Take stream.txt and run it through noelephants.sed

```
$ cat stream.txt | sed -f noelephants.sed
Imagine a quaint bubbling stream of cool mountain water filled
Although you can substitute trout with catfish, there is no substitute for elephants,
   <CONT>so we cannot offer this item.
```

Regular Expression Addresses

- When addresses are specified as in the previous examples, the editing command affects only the line(s) that you explicitly denoted in the address.
- Since it is not possible for you to know where in your file you want to perform sed operations, sed allows you to use regular expressions (regexps) to make your addressing much more powerful and useful.

Regular Expression Addresses

```
$ cat /etc/sysconfig/syslog
## Path:                System/Logging
## Description:         System logging
## Type:                list(0,1,2,3,4,5,6,7)
## Default:             1
## Config:              "“
## ServiceRestart:      syslog
#
# Default loglevel for klogd
#
KERNEL_LOGLEVEL=1

## Type:                string
## Default:             ""
## Config:              ""
## ServiceRestart:      syslog
#
# if not empty: parameters for syslogd
# for example SYSLOGD_PARAMS="-r -s my.dom.ain"
#
SYSLOGD_PARAMS=""
<CUT… Partial file>
```

Regular Expression Addresses

```
mtrutschl@sun> cat /etc/sysconfig/syslog | sed  '/^#/d'
KERNEL_LOGLEVEL=1

SYSLOGD_PARAMS=""

KLOGD_PARAMS="-x"

SYSLOG_DAEMON="syslog-ng"

SYSLOG_NG_CREATE_CONFIG="yes"

SYSLOG_NG_PARAMS=""
SYSLOGD_ADDITIONAL_SOCKET_NAMED="/var/lib/named/dev/log"

SYSLOGD_ADDITIONAL_SOCKET_NTP="/var/lib/ntp/dev/log"
mtrutschl@sun>
```

- The regular expression /^#/ removes all the comments.
- The blank lines are also printed.

Inverted Regular Expression Match

```
$ cat /etc/sysconfig/syslog | sed -n '/^#/p‘ <Enter>
```

- The command prints to your screen all the comments in the syslog file and nothing else

Regular Expression

- The regular expression table lists four special characters that are very useful in regular expressions.

Character	Description
^	Matches the beginning of lines
$	Matches the end of lines
.	Matches any single character
*	Matches zero or more occurrences of the previous character.

Regular Expressions

```
$ cat foo.txt | sed '/^$/d‘ <Enter>
#  foo.txt      Demo file
#
#  For more information see…
#  First some standard logfiles.  Log by facility.
auth,authpriv.*                  /var/log/auth.log
```

- ^ matches the beginning of the line
- $ matches the end of lines
- The combination matches and deletes all lines that have nothing between the beginning and end of the line.

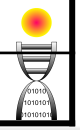
Regular Expressions

```
$ cat foo.txt | sed -n '/^[abc]/p'
auth,authpriv.*                 /var/log/auth.log
```

- This combines the regular expression ^ with the regular expression [abc] to print only those lines that begin with any one of those characters.
- The [] brackets denote a range of characters.
- [g-t] would get you all lowercase characters between g and t.
- [3-25] would get you all numbers between 3 and 25.

Character Class Keywords

Character Class Keyword	Description
[[:alnum:]]	Alphanumeric [a-z A-Z 0-9]
[[:alpha:]]	Alphabetic [a-z A-Z]
[[:blank:]]	Blank characters (space or tabs)
[[:cntrl:]]	Control characters
[[:digit:]]	Numbers [0-9]
[[:graph:]]	Any visible characters (excludes whitespaces)
[[:lower:]]	Lowercase letters [a-z]
[[:print:]]	Printable characters (noncontrol characters)
[[:punct:]]	Punctuation characters
[[:space:]]	Whitespace
[[:upper:]]	Uppercase letters [A-Z]
[[:xdigit:]]	Hex digits [0-9 a-f A-F]



Character Class Keywords

```
$ cat syslog.conf | sed -n '/^[[:alpha:]]/p' <Enter>
auth,authpriv.*                 /var/log/auth.log
```

- This command prints only those lines in the syslog.conf file that start with a letter of the alphabet.

Character Class Keywords

```
$ cat foo.txt | sed '/^[[:alpha:]]/d‘ <Enter>
#  foo.txt      Sample file…
#
#  For more information see…

#
#  First some standard logfiles.  Log by facility.
#
```

- This command deletes all lines in the foo.txt file that start with a letter of the alphabet

Combining Line Addresses with regexps

```
$ cat foo.txt | sed '1,/^$/d‘ <Enter>
# First some standard logfiles.  Log by facility.

auth,authpriv.*                 /var/log/auth.log
```

- This command starts deleting from the first line in the file and continues to delete up to the first line that is blank.

Advanced Substitution

```
$ cat stream.txt | sed 's/trout/catfish/g'
Imagine a quaint bubbling stream of cool mountain water filled
with rainbow catfish and elephants drinking iced tea.
```

- To do regular expression substitutions, you simply map a regular expression onto the literal string.

```
sed 's/^$/<p>/g'
```

- The first part of the substitution looks for blank lines and replaces them with the HTML <p> paragraph marker.

Referencing Matched regexps with &

- The sed metacharacter ‘&’ represents the contents of the pattern that was matched.
- If you have the file phonenums.txt with the following data

```
5555551212
5555551213
5555551214
6665551215
6665551216
7775551217
```

Regular Expression Address Ranges

- You want to surround the area code with parentheses, for easier reading.
- To do this you can use the ampersand (&) replacement character.

```
$ sed -e 's/^[0-9][0-9][0-9]/(&)/g' phonenums.txt <ENT>
(555)5551212
(555)5551213
(555)5551214
(666)5551215
(666)5551216
(777)5551217
```

Regular Expression Address Ranges

- Another way of getting the same output as the previous slide…

```
$ sed -e 's/^[[:digit:]] [[:digit:]] [[:digit:]]/(&)/g'
  phonenums.txt <Enter>
(555)5551212
(555)5551213
(555)5551214
(666)5551215
(666)5551216
(777)5551217
```

Regular Expression Address Ranges

- What does this statement do?

```
$ sed -e 's/[0-9][0-9][0-9][0-9][0-9][0-9]/&-/g'
  phonenums.txt | sed -e 's/^[0-9][0-9][0-9]/(&)/g‘
```

Note that this is one long line.

Back References

- To do back references, you have to first define a region and then refer back to that region.
- To define a region, you insert backslashed parentheses around each region of interest.
- The first region that you surround with backslashes is then referenced by \1, the second \2 and so on.
- Using phonenums.txt, create a file nums.txt, so that the area code is in parentheses and there is a hyphen after the 6th digit in the phone number.

Back References

```
$ sed -e 's/^[0-9]\{3\}/(&)/g' -e 's/)[0-9]\{3\}/&-/g'
phonenums.txt > nums.txt <Enter>
          'or'
$ sed -e 's/^[[:digit:]]\{3\}/(&)/g' -e 's/)[[:digit:]]
\{3\}/&-/g' phonenums.txt > nums1.txt <Enter>
$ cat nums.txt <Enter>
(555)555-1212
(555)555-1213
(555)555-1214
(666)555-1215
(666)555-1216
(777)555-1217
```

Back References

- First define the three regions in the left side of the sed command. i.e. the area code, the second set of numbers up to the dash, and the rest of the numbers.
- To select the area code, define a regular expression that includes parentheses. `/.*)/`
- This matches any number up to a ‘)’.
- If you want to reference this match later, you need to enclose the expression in escaped parentheses.

Back References

```
/\(.*)\)/
```

- This matches any number of characters up to a ‘)’.
- If you want to reference this match later, you need to enclose the expression in escaped parentheses.
- Now, you want to match the second set of numbers, terminated by a hyphen,

```
/\(.*-\)/
```

- The third set is specified by matching any character repeating up to the end of the line.

Back References

```
/\(.*$\)/
```

- Now we can put it all together in a search and then use the references in the replacement right side.

```
$ cat nums.txt | sed 's/\(.*\)\(.*-\)\(.*$\)/Area
  code: \1 Second: \2 Third: \3/‘ <Enter>
Area code: (555) Second: 555- Third: 1212
Area code: (555) Second: 555- Third: 1213
Area code: (555) Second: 555- Third: 1214
Area code: (666) Second: 555- Third: 1215
Area code: (666) Second: 555- Third: 1216
Area code: (777) Second: 555- Third: 1217
```

Hold Space

- The “hold space” is a temporary space to put things while you do other things, or look for the other lines.
- Lines in the hold space cannot be operated on, you can only put things in the hold space and take things out from it. (a.k.a. “accumulator”)
- Any work you want to do on lines has to be done in the pattern space.

Hold Space

- The most common use of the hold space is to make a duplicate of the current line while you change the original in the pattern space.

Command	Description of Command's Function
H or h	Overwrite (h) or append (H) the hold space with the contents of the pattern buffer into the hold buffer.
G or g	Overwrite (g) or append (G) the pattern space with the contents of hold space.
x	Exchange the pattern space and the hold space, note that this command is not useful by itself.

Using Hold Space

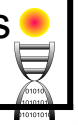
```
mtrutschl@sun> cat /etc/passwd | sed -ne '2G' -e 'p'
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash

daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
<CUT>
```

Common Sed Commands

Editing Commands	Description of Command's Function
#	Comment. If first two characters of a sed script are #n, then the –n (no auto-print) option is forced
{ COMMANDS }	A group of COMMANDS may be enclosed in curly braces to be executed together. This is useful when you have a group of commands that you want executed on an address match.
D[address][,address2]d	Deletes line(s) from pattern space
n	If auto-print was not disabled (-n), print the pattern space, and then replace the pattern space with the next line of input. If there is no more input, sed exits.

Less Common Sed Commands

Command	Usage
:label	Label a line to reference later for transfer of control via b and t commands.
a[address][,address2]a\text	Append text after each line matched by address or address range
b[address][,address2]]b[label]	Branch (transfer control unconditionally) to :label.
c[address][,address2]]\text	Delete the line(s) matching address and then output the lines of text that follow this command in place of the last line.
D[address][,address2]]D	Delete first part of multiline pattern (created by N command) space up to newline.
g	Replace the contents of the pattern space with the contents of the hold space



Less Common Sed Commands

Command	Usage
G	Add a newline to the end of the pattern space and then append the contents of the hold space to that of the pattern space
h	Replace the contents of the hold space with the contents of the pattern space
H	Add a newline to the end of the hold space and then append the contents of the pattern space to the end of the pattern space.
i[address][,address2]\text	Immediately output the lines of the text that follow this command; the final line ends with an unprinted “\”.
1N	Print the pattern space using N lines as the word-wrap length. Nonprintable characters and the \ character are printed in C-style escaped form. Long lines are split with a trailing “\” to indicate the split; the end of each line is marked with “$”.

Less Common Sed Commands

Command	Usage
P	Add a newline to the pattern space and then append the next line of input into the pattern space. If there is no more input, sed exits
r[address][,address2]FILENAME	Read in a line of FILENAME and insert it into the output stream at the end of a cycle. If file name cannot be read, or end-of-line is reached, no line is appended. Special file /dev/stdin can be provided to read a line from the standard input.
w[address][,address2]FILENAME	Write to FILENAME the pattern space. The special file names /dev/stderr and /dev/stdout are available to GNU sed. The file is created before the first line input line is read. All the w commands that refer to the same FILENAME are output without closing and reopening the file
x	Exchange the contents of the hold and pattern space.

GNU sed-Specific sed Extensions

Editing Command	Description of Command's Function
e[COMMAND]	Without parameters, executes the command found in pattern space, replacing pattern space with its output. With the parameter COMMAND, interprets COMMAND and sends output of command to output stream
LN	Fills and joins lines in pattern space to produce output lines of N characters (at most). This command will be removed in future releases.
Q[EXIT-CODE]	Same as common q command, except that it does not print the pattern space. It provides the ability to retuen an EXIT-CODE
R FILENAME	Reads in a line of FILENAME, and inserts it into the output stream at the end of a cycle. If file name cannot be read or eof is reached, no line is appended. Special file /dev/stdin can be provided to read a line from standard input.
T LABEL	Branch to LABEL if there have been no successful substitution(s) since last input line was read or branch taken. If LABEL is omitted, the next cycle is started.

GNU sed-Specific sed Extensions

Editing Command	Description of Command's Function
v VERSION	This command fails if GNU sed extensions are not supported. You can specify the VERSION of GNU sed required; default is 4.0, as this is the version that first supports this command
W FILENAME	Write to FILENAME the pattern space up to the first newline. See standard w command regarding the handles.

In-class lab (time permitting)

- Page 227, exercises 1, 2, and 3

Questions/Comments?

Presentation slides by:

Dr. Marjan Trutschl and Vanessa Pacheco

Contact information:

marjan.trutschl@lsus.edu