sed

ComS 252 — Iowa State University

Andrew Miner

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?
  - Open your favorite editor (`vim`, `nano`, `Emacs`, ...)
  - Make the changes "by hand"
    - Except you utilize editor features like "search and replace"

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?
  - Open your favorite editor (`vim`, `nano`, `Emacs`, ...)
  - Make the changes "by hand"
    - Except you utilize editor features like "search and replace"
- What if the file is 2 Terabytes?

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?
  - Open your favorite editor (`vim`, `nano`, `Emacs`, ...)
  - Make the changes "by hand"
    - Except you utilize editor features like "search and replace"
- What if the file is 2 Terabytes?
- What if you need to make the same edits to 100,000 files?

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?
  - Open your favorite editor (`vim`, `nano`, `Emacs`, ...)
  - Make the changes "by hand"
    - Except you utilize editor features like "search and replace"
- What if the file is 2 Terabytes?
- What if you need to make the same edits to 100,000 files?

- "Gee whiz, I wish there were a way to automate those edits"

Motivation

- Suppose you want to make a series of edits to a file, e.g.:
  - Replace some pattern of text with alternate text
  - Remove some pattern of text
- How do you do this for a single file?
  - Open your favorite editor (`vim`, `nano`, `Emacs`, ...)
  - Make the changes "by hand"
    - Except you utilize editor features like "search and replace"
- What if the file is 2 Terabytes?
- What if you need to make the same edits to 100,000 files?

- "Gee whiz, I wish there were a way to automate those edits"
  - That's what `sed` is for!

What is sed?

- Stream editor
- Technically — we have `sed` programs
- Idea:
  1. Read lines of input file(s), one at a time
  2. Make edits on each line
     - Runs the entire program on the current line
  3. Writes to standard output
- Similar to `awk`
  - But different features

Running sed programs

```
sed 'program' file1 file2 ...
```

- Pass the entire program as the first argument
  - Easiest to use single quotes
- The remaining arguments are input files
- No files given: reads from standard input

Running sed programs

`sed 'program' file1 file2 ...`

- Pass the entire program as the first argument
  - Easiest to use single quotes
- The remaining arguments are input files
- No files given: reads from standard input

`sed -f program file1 file2 ...`

- Use `-f` to read the program from a file
- For a sed script use first magic line:

```
#!/usr/bin/sed -f
# Check your sed path...
# Program statements here
```

Running sed programs

`sed 'program' file1 file2 ...`

- Pass the entire program as the first argument
  - Easiest to use single quotes
- The remaining arguments are input files
- No files given: reads from standard input

`sed -f program file1 file2 ...`

- Use `-f` to read the program from a file
- For a sed script use first magic line:

```
#!/usr/bin/sed -f
# Check your sed path...
# Program statements here
```

Same as awk

sed statements

Generic syntax

```
address(es) instruction instruction-arguments
```

- "address(es)": lines to which the instruction applies
  - If not specified, applies to all input lines
  - A few instructions do not allow addresses
  - A few instructions have restrictions on addresses
- "instruction" is usually a single letter
- "instruction-arguments" depend on the instruction
- sed is picky: no spaces after an instruction
  - But some sed implementations will not complain

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$ ls | sed -n
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
a.out    bar.txt  foo.txt
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$ ls | sed -n
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
a.out    bar.txt  foo.txt
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$ ls | sed -n
prompt$ ls | sed 'd'
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$ ls | sed -n
prompt$ ls | sed 'd'
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
prompt$ ls | sed
a.out
bar.txt
foo.txt
prompt$ ls | sed -n
prompt$ ls | sed 'd'
prompt$ ls | sed 'p'
```

Simple sed instructions

`p` : Print the current line

`d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
a.out
a.out
bar.txt
bar.txt
foo.txt
foo.txt
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
a.out
a.out
bar.txt
bar.txt
foo.txt
foo.txt
prompt$ ls | sed -n 'p'
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
foo.txt
foo.txt
prompt$ ls | sed -n 'p'
a.out
bar.txt
foo.txt
prompt$
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
foo.txt
foo.txt
prompt$ ls | sed -n 'p'
a.out
bar.txt
foo.txt
prompt$ ls | sed 'p;d'
```

Simple sed instructions

- `p` : Print the current line
- `d` : Delete the current line

Note:

- By default, `sed` prints every line after processing
- If we use `sed -n`, lines are not printed

```
bar.txt
foo.txt
prompt$ ls | sed 'p;d'
a.out
bar.txt
bar.txt
foo.txt
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed -n '1p'
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed -n '1p'
a.out
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed -n '1p'
a.out
prompt$ ls | sed -n '$p'
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed -n '1p'
a.out
prompt$ ls | sed -n '$p'
foo.txt
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
prompt$ ls
a.out    bar.txt  foo.txt
prompt$ ls | sed -n '1p'
a.out
prompt$ ls | sed -n '$p'
foo.txt
prompt$ ls | sed '2d'
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
a.out
prompt$ ls | sed -n '$p'
foo.txt
prompt$ ls | sed '2d'
a.out
foo.txt
prompt$
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
a.out
prompt$ ls | sed -n '$p'
foo.txt
prompt$ ls | sed '2d'
a.out
foo.txt
prompt$ ls | sed '/txt/d'
```

Simple addresses

A single address is of the form:

- Number
  - Apply instruction to this line number
- $
  - Special case of "number": last line of input
- /pattern/
  - Apply instruction to any line containing the pattern
  - The pattern can be a regular expression

```
foo.txt
prompt$ ls | sed '2d'
a.out
foo.txt
prompt$ ls | sed '/txt/d'
a.out
prompt$
```

Address ranges

We can apply an instruction to a range of lines using:

```
address,address
```

where each address is any of: number, $, /pattern/

Address ranges

We can apply an instruction to a range of lines using:

```
address,address
```

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'`

Address ranges

We can apply an instruction to a range of lines using:

```
address,address
```

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'` : print first 10 lines

Address ranges

We can apply an instruction to a range of lines using:

```
address,address
```

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'` : print first 10 lines
- `sed -n '15,/FOO/p'`

Address ranges

We can apply an instruction to a range of lines using:

```
address,address
```

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'` : print first 10 lines
- `sed -n '15,/FOO/p'`
  - Print lines from number 15 to a line containing FOO, inclusive

Address ranges

We can apply an instruction to a range of lines using:

`address,address`

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'` : print first 10 lines
- `sed -n '15,/FOO/p'`
  - Print lines from number 15 to a line containing FOO, inclusive
- `sed -n '/begin/,/end/p'`

Address ranges

We can apply an instruction to a range of lines using:

`address,address`

where each address is any of: number, $, /pattern/

Examples:

- `sed '11,$d'` : print first 10 lines
- `sed -n '15,/FOO/p'`
  - Print lines from number 15 to a line containing FOO, inclusive
- `sed -n '/begin/,/end/p'`
  - When a line contains "begin", start printing
  - When a line contains "end", print the line and stop printing

Inverting an address or range

Use “!” before the instruction to invert the addresses

Inverting an address or range

Use "!" before the instruction to invert the addresses

Examples:

- `sed '1,10!d'`

Inverting an address or range

Use "!" before the instruction to invert the addresses

Examples:

- `sed '1,10!d'`
  - Delete all lines except 1 through 10
  - (Same as `head`)

Inverting an address or range

Use "!" before the instruction to invert the addresses

Examples:

- `sed '1,10!d'`
  - Delete all lines except 1 through 10
  - (Same as `head`)
- `sed '/pattern/!d'`

Inverting an address or range

Use "!" before the instruction to invert the addresses

Examples:

- `sed '1,10!d'`
  - Delete all lines except 1 through 10
  - (Same as `head`)
- `sed '/pattern/!d'`
  - Delete all lines except lines containing "pattern"
  - (Same as `grep`)

Insert, append, and change

- `a` : append text
  - Can take a single address only
  - Adds specified text after the current line
- `i` : insert text
  - Can take a single address only
  - Adds specified text before the current line
- `c` : change text
  - Can take an address range as usual
  - Changes the current line to the specified text
- Note!
  - In the program, the specified text must appear on the next line
  - These instructions are annoying to use on the command line

Example: change line 3

ch3.sed

```
#!/usr/bin/sed -f

3c\
LINE 3!  WOO HOO!
```

```
prompt$
```

Example: change line 3

ch3.sed

```
#!/usr/bin/sed -f

3c\
LINE 3!  WOO HOO!
```

```
prompt$ ls
```

Example: change line 3

ch3.sed

```
#!/usr/bin/sed -f

3c\
LINE 3!  WOO HOO!
```

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$
```

Example: change line 3

ch3.sed

```
#!/usr/bin/sed -f

3c\
LINE 3!  WOO HOO!
```

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | ./ch3.sed
```

Example: change line 3

ch3.sed

```
#!/usr/bin/sed -f

3c\
LINE 3!  WOO HOO!
```

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | ./ch3.sed
a.out
bar.txt
LINE 3!  WOO HOO!
foo.txt
prompt$
```

Example: insert lines

```
prompt$
```

Example: insert lines

```
prompt$ ls
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$
```

Example: insert lines

```
prompt$ ls
a.out      bar.txt    ch3.sed*   foo.txt
prompt$ ls | sed '1i\
```

Example: insert lines

```
prompt$ ls
a.out      bar.txt    ch3.sed*   foo.txt
prompt$ ls | sed '1i\
>
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
> 
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
> Here is your listing:
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
> Here is your listing:
> 
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
> Here is your listing:
> '
```

Example: insert lines

```
prompt$ ls
a.out     bar.txt   ch3.sed*  foo.txt
prompt$ ls | sed '1i\
> Hello!\
> Here is your listing:
> '
Hello!
Here is your listing:
a.out
bar.txt
ch3.sed*
foo.txt
prompt$
```

y: transform characters

- Usage: `y/abcstr/xyzstr/`
- Does the same as `tr 'abcstr' 'xyzstr'`
- Transforms character in `abcstr`
  into the corresponding character in `xyzstr`

```
prompt$
```

y: transform characters

- Usage: `y/abcstr/xyzstr/`
- Does the same as `tr 'abcstr' 'xyzstr'`
- Transforms character in `abcstr`
  into the corresponding character in `xyzstr`

```
prompt$ ls | sed 'y/abcd/ABCD/'
```

y: transform characters

- Usage: `y/abcstr/xyzstr/`
- Does the same as `tr 'abcstr' 'xyzstr'`
- Transforms character in `abcstr`
  into the corresponding character in `xyzstr`

```
prompt$ ls | sed 'y/abcd/ABCD/'
A.out
BAr.txt
Ch3.seD*
foo.txt
prompt$
```

y: transform characters

- Usage: `y/abcstr/xyzstr/`
- Does the same as `tr 'abcstr' 'xyzstr'`
- Transforms character in `abcstr`
  into the corresponding character in `xyzstr`

```
prompt$ ls | sed 'y/abcd/ABCD/'
A.out
BAr.txt
Ch3.seD*
foo.txt
prompt$ ls | sed '2!y/abcd/ABCD/'
```

y: transform characters

- Usage: `y/abcstr/xyzstr/`
- Does the same as `tr 'abcstr' 'xyzstr'`
- Transforms character in `abcstr`
  into the corresponding character in `xyzstr`

```
prompt$ ls | sed 'y/abcd/ABCD/'
A.out
BAr.txt
Ch3.seD*
foo.txt
prompt$ ls | sed '2!y/abcd/ABCD/'
A.out
bar.txt
Ch3.seD*
foo.txt
prompt$
```

s: substitution

- Usage: `s/pattern/replacement/flags`
- "`pattern`" can be a regular expression
- "`replacement`" can have special characters
  - We will get to those in a minute
- Default: replaces only the first occurrence on each line
- "`flags`" modify the substitution:
  - *n* : replace only the n^{th} occurrence on the line
  - `g` : change globally (all occurrances on the line)
  - `p` : print the line after changing it

Substitution examples

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
ho ho ho ho
prompt$
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
ho ho ho ho
prompt$ ls | sed '4s/txt/text/'
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
ho ho ho ho
prompt$ ls | sed '4s/txt/text/'
a.out
bar.txt
ch3.sed*
foo.text
prompt$
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
ho ho ho ho
prompt$ ls | sed '4s/txt/text/'
a.out
bar.txt
ch3.sed*
foo.text
prompt$ ls | sed -n 's/bar/pub/p'
```

Substitution examples

```
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/'
ho hi hi hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/3'
hi hi ho hi
prompt$ echo 'hi hi hi hi' | sed 's/hi/ho/g'
ho ho ho ho
prompt$ ls | sed '4s/txt/text/'
a.out
bar.txt
ch3.sed*
foo.text
prompt$ ls | sed -n 's/bar/pub/p'
pub.txt
prompt$
```

Regular expression substitutions

- We can replace a regular expression pattern with fixed text
- But what about replacements that are not fixed text?
  - For example, can we replace text "`integer(`*n*`)`" with "`int` *n*" for any integer *n*?

Regular expression substitutions

- We can replace a regular expression pattern with fixed text
- But what about replacements that are not fixed text?
  - For example, can we replace text "`integer(`*n*`)`" with "`int` *n*" for any integer *n*?
- Use the following special characters in the replacement text

  `&` : replaced by the pattern text

  `\`*n* : the n^{th} substring in the pattern regular expression
  - *n* is a single digit
  - Use "`\(`" and "`\)`" in the pattern to specify substrings

  `\` : for escaping special characters

Advanced substitution examples

```
prompt$
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
  PID TTY          TIME CMD
xx pts/2    xx:xx:xx bash
xx pts/2    xx:xx:xx ps
xx pts/2    xx:xx:xx sed
prompt$
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
  PID TTY          TIME CMD
xx pts/2    xx:xx:xx bash
xx pts/2    xx:xx:xx ps
xx pts/2    xx:xx:xx sed
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/x&x/'
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
  PID TTY          TIME CMD
xx pts/2    xx:xx:xx bash
xx pts/2    xx:xx:xx ps
xx pts/2    xx:xx:xx sed
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/x&x/'
  PID TTY          TIME CMD
x30164x pts/2    00:00:00 bash
x30400x pts/2    00:00:00 ps
x30401x pts/2    00:00:00 sed
prompt$
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
  PID TTY          TIME CMD
xx pts/2    xx:xx:xx bash
xx pts/2    xx:xx:xx ps
xx pts/2    xx:xx:xx sed
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/x&x/'
  PID TTY          TIME CMD
x30164x pts/2    00:00:00 bash
x30400x pts/2    00:00:00 ps
x30401x pts/2    00:00:00 sed
prompt$ ps | sed 's/\([0-9]\)[0-9]*\([0-9]\)/\1...\2/'
```

Advanced substitution examples

```
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/xx/g'
  PID TTY          TIME CMD
xx pts/2    xx:xx:xx bash
xx pts/2    xx:xx:xx ps
xx pts/2    xx:xx:xx sed
prompt$ ps | sed 's/[0-9][0-9]*[0-9]/x&x/'
  PID TTY          TIME CMD
x30164x pts/2    00:00:00 bash
x30400x pts/2    00:00:00 ps
x30401x pts/2    00:00:00 sed
prompt$ ps | sed 's/\([0-9]\)[0-9]*\([0-9]\)/\1...\2/'
  PID TTY          TIME CMD
3...4 pts/2    00:00:00 bash
3...6 pts/2    00:00:00 ps
3...7 pts/2    00:00:00 sed
prompt$
```

Quiz

How do we replace text "`integer(`*n*`)`"
with "`int` *n*" for any (non-negative) integer *n*?

Quiz

How do we replace text "`integer(`*n*`)`"
with "`int` *n*" for any (non-negative) integer *n*?

```
sed 's/integer(\([0-9]*[0-9]\))/int \1/'
```

Quiz

How do we replace text "`integer(`*n*`)`"
with "`int` *n*" for any (non-negative) integer *n*?

```
sed 's/integer(\([0-9]*[0-9]\))/int \1/'
```

How do we replace text "`integer(`*n*`)`"
with "`even(`*n*`)`" for any (non-negative) even integer *n*?

Quiz

How do we replace text "`integer(`*n*`)`"
with "`int` *n*" for any (non-negative) integer *n*?

```
sed 's/integer(\([0-9]*[0-9]\))/int \1/'
```

How do we replace text "`integer(`*n*`)`"
with "`even(`*n*`)`" for any (non-negative) even integer *n*?

```
sed 's/integer(\([0-9]*[02468]\))/even(\1)/'
```

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1.

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1. On every line, replace " first" with "second"

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1. On every line, replace " first" with "second"
2.

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1. On every line, replace " first" with "second"
2. On lines containing "second", replace "These" with "Those"

Longer sed programs

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1. On every line, replace " first" with "second"
2. On lines containing "second", replace "These" with "Those"

Let's run this and see what happens...

Example: running 2.sed

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

```
prompt$
```

Example: running 2.sed

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

```
prompt$ cat file.txt
```

Example: running 2.sed

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

```
prompt$ cat file.txt
The first line of a small file.
These characters form the second line.
This is the third line.
prompt$
```

Example: running 2.sed

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

```
prompt$ cat file.txt
The first line of a small file.
These characters form the second line.
This is the third line.
prompt$ ./2.sed file.txt
```

Example: running 2.sed

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

```
prompt$ cat file.txt
The first line of a small file.
These characters form the second line.
This is the third line.
prompt$ ./2.sed file.txt
Thosecond line of a small file.
Those characters form the second line.
This is the third line.
prompt$
```

sed programming model — more precise

sed programs execute as follows.

0. If no more input lines, then terminate
1. Read the next input line into the pattern space
2. Execute sed statements, in order, on the pattern space
   - The /pattern/ address applies to the pattern space
   - Any text changes are made to the pattern space
3. Go to step (0)

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1.

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1. Delete any line containing text “foo”

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1. Delete any line containing text “foo”
2.

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1. Delete any line containing text “foo”
2. Replace “line” with “queue” in any line containing “bar”

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1. Delete any line containing text “foo”
2. Replace “line” with “queue” in any line containing “bar”

- What will happen if a line contains both “foo” and “bar”?

Delete revisited

Consider the following program:

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

What does each statement do?

1. Delete any line containing text "foo"
2. Replace "line" with "queue" in any line containing "bar"

- What will happen if a line contains both "foo" and "bar"?
  - The line will be deleted
- Statements are not executed on a deleted pattern space
- Effectively, control goes back to the top of the program

Delete example

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

```
prompt$
```

Delete example

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

```
prompt$ cat file2.txt
```

Delete example

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

```
prompt$ cat file2.txt
This line contains foo and should be deleted.
There was a long line at the bar.
Will sed barf on this line containing foo?
The ABC computer solved linear systems.
prompt$
```

Delete example

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

```
prompt$ cat file2.txt
This line contains foo and should be deleted.
There was a long line at the bar.
Will sed barf on this line containing foo?
The ABC computer solved linear systems.
prompt$ ./foobar.sed file2.txt
```

Delete example

foobar.sed

```
#!/usr/bin/sed -f
/foo/d
/bar/s/line/queue/
```

```
prompt$ cat file2.txt
This line contains foo and should be deleted.
There was a long line at the bar.
Will sed barf on this line containing foo?
The ABC computer solved linear systems.
prompt$ ./foobar.sed file2.txt
There was a long queue at the bar.
The ABC computer solved linear systems.
prompt$
```

Grouping statements

- Statements may be grouped with braces
  - The closing brace must be on its own line
- Statements within braces may have addresses
- The group may have addresses
- A group may appear within another group
- For example:

Grouping statements

- Statements may be grouped with braces
  - The closing brace must be on its own line
- Statements within braces may have addresses
- The group may have addresses
- A group may appear within another group
- For example:

```
s/ first/second/
/second/s/These/Those/
```

Grouping statements

- Statements may be grouped with braces
  - The closing brace must be on its own line
- Statements within braces may have addresses
- The group may have addresses
- A group may appear within another group
- For example:

```
1 {
  s/ first/second/
  /second/s/These/Those/
}
```

Grouping statements

- Statements may be grouped with braces
  - The closing brace must be on its own line
- Statements within braces may have addresses
- The group may have addresses
- A group may appear within another group
- For example:

```
/line/ {
  1 {
    s/ first/second/
    /second/s/These/Those/
  }
}
```

Quit Instruction

`q` : Quit

- Pattern space will still be printed
  - Unless we used `sed -n`
- No more input lines will be read

Quit Instruction

`q` : Quit

- Pattern space will still be printed
  - Unless we used `sed -n`
- No more input lines will be read

- Examples:
  - Better way to do "`head`" with `sed`:

```
sed '10q'
```

Quit Instruction

`q` : Quit

- Pattern space will still be printed
  - Unless we used `sed -n`
- No more input lines will be read

- Examples:
  - Better way to do "`head`" with `sed`:

    ```
    sed '10q'
    ```

  - Extract text until the first line containing "`</html>`":

    ```
    sed '/<\/html>/q'
    ```

Next Instruction

`n` : read the next line into the pattern space

- The pattern space will be printed, first
  - Unless we used `sed -n`
- Control does not go back to the start of the program

Example: convert my "shorthand" into HTML

- "My shorthand": special characters at the start of a line:
  - `hr` signifies a new page, convert to "<hr>"
  - `h1` following `hr` converts to "<h1>text</h1>"
- Example input:

```
hr
h1 This is the title
Random text here.
hr
No title here.
```

- Desired output:

```
<h1> This is the title</h1>
Random text here.
<hr>
No title here.
```

Example continued: the sed script

short2html.sed

```
#!/usr/bin/sed -f
/^hr/ {
  s/hr/<hr>/
  n
  /^h1/ {
    s/h1\(.*\)/<h1>\1<\/h1>/
  }
}
```

Motivating example

- Suppose we want to replace "Red Hat" with "Fedora"

Motivating example

- Suppose we want to replace "Red Hat" with "Fedora"
- Easy:

```
s/Red Hat/Fedora/g
```

Right?

Motivating example

- Suppose we want to replace "Red Hat" with "Fedora"
- Easy:

  `s/Red Hat/Fedora/g`

  Right?
- What about the following text:

```
Debian users should know about .deb files, and Red
Hat users should know about .rpm files.
```

Motivating example

- Suppose we want to replace "Red Hat" with "Fedora"
- Easy:

  `s/Red Hat/Fedora/g`

  Right?
- What about the following text:

```
Debian users should know about .deb files, and Red
Hat users should know about .rpm files.
```

- Can we use `n` and grouping like before?

Motivating example

- Suppose we want to replace "Red Hat" with "Fedora"
- Easy:

  `s/Red Hat/Fedora/g`

  Right?
- What about the following text:

```
Debian users should know about .deb files, and Red
Hat users should know about .rpm files.
```

- Can we use `n` and grouping like before?
  - `n` causes current line to be printed...
  - ...before we know if "Red" should be replaced
- To do this "right", we need to examine 2 lines of input

Instructions for a multi–line pattern space

- `sed` can store multiple lines in the pattern space
  - A newline character "`\n`" will be between the lines
  - Meta character "`^`" does not match after "`\n`"
  - Meta character "$" does not match before "`\n`"

Instructions for a multi–line pattern space

- sed can store multiple lines in the pattern space
  - A newline character "\n" will be between the lines
  - Meta character "^" does not match after "\n"
  - Meta character "$" does not match before "\n"

N: read next line, append it to pattern space

- A newline character will be added between lines, if necessary

D: delete first line of the pattern space and go to top of script

- Deletes up to and including the first newline character
- A new line is not read into the pattern space
- Processing continues on whatever remains in pattern space

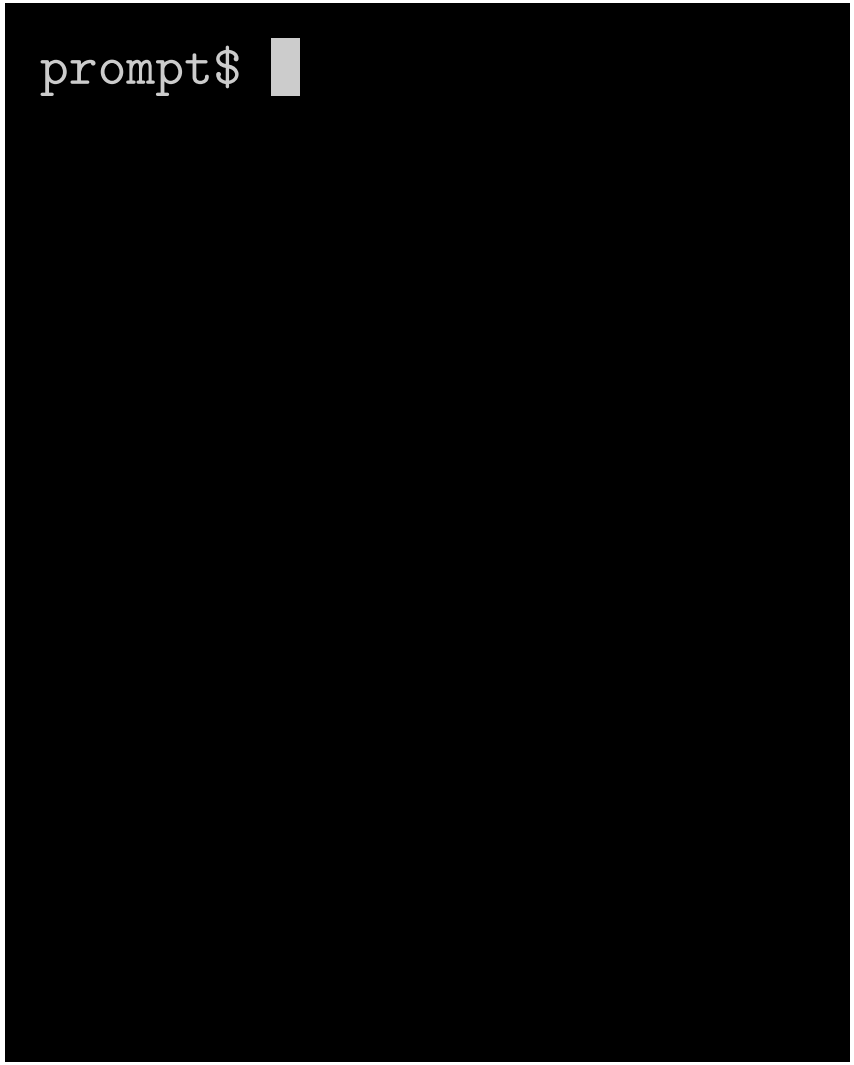
P: print first line of the pattern space

- Print up to the first newline character

Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```



Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```

```
prompt$ cat red.txt
```

Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```

```
prompt$ cat red.txt
My fun Red  Hat example.
The tricky case with Red
Hat split across lines.
Multiple Red Hats and Red
Hat split across lines.

Evil cases:
Red
Hat Red
Hat Red Hat Red
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$
```

Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```

```
prompt$ cat red.txt
My fun Red  Hat example.
The tricky case with Red
Hat split across lines.
Multiple Red Hats and Red
Hat split across lines.

Evil cases:
Red
Hat Red
Hat Red Hat Red
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$ ./rhr2.sed red.txt
```

Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```

```
prompt$ ./rhr2.sed red.txt
My fun Fedora example.
The tricky case with Fedora
split across lines.
Multiple Fedoras and Fedora
split across lines.

Evil cases:
Fedora
Red
Hat Fedora Fedora
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$
```

Red Hat replacement: attempt 2

rhr2.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
}
```

Problem:

"Red Hat" not changed in remaining pattern space after `N`

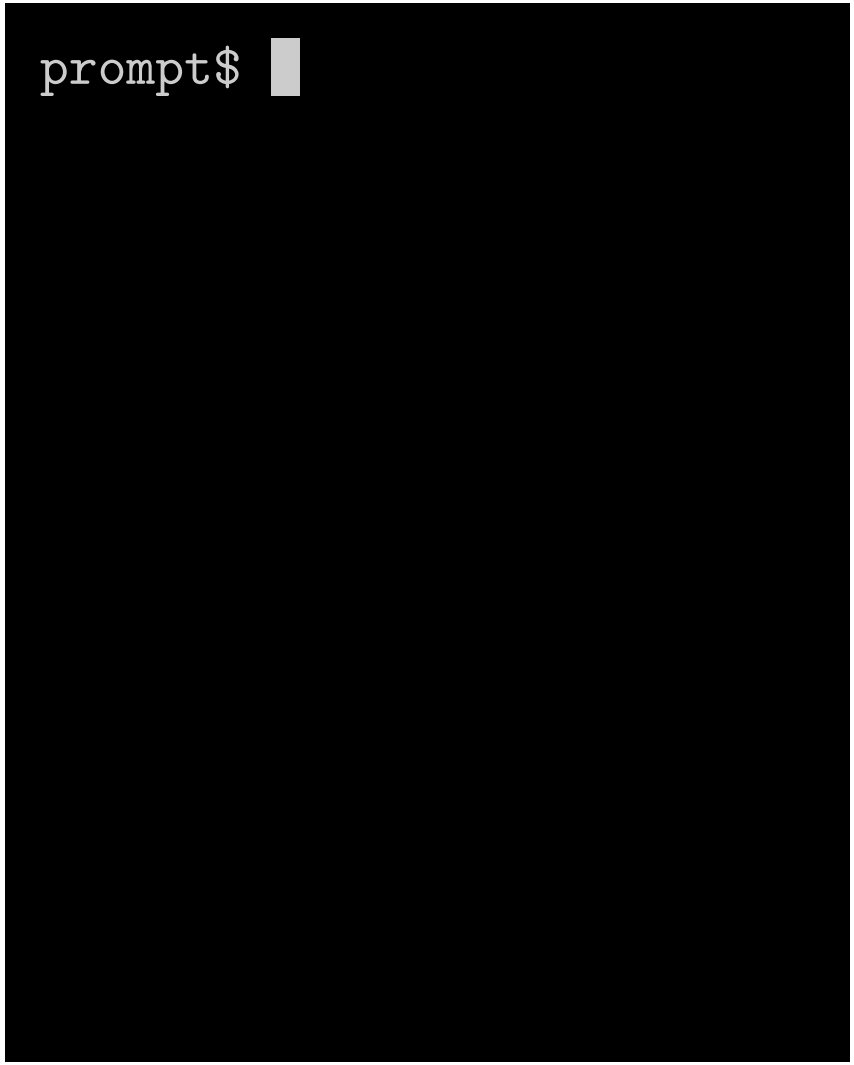
```
prompt$ ./rhr2.sed red.txt
My fun Fedora example.
The tricky case with Fedora
split across lines.
Multiple Fedoras and Fedora
split across lines.

Evil cases:
Fedora
Red
Hat Fedora Fedora
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$
```

Red Hat replacement: attempt 3

rhr3.sed

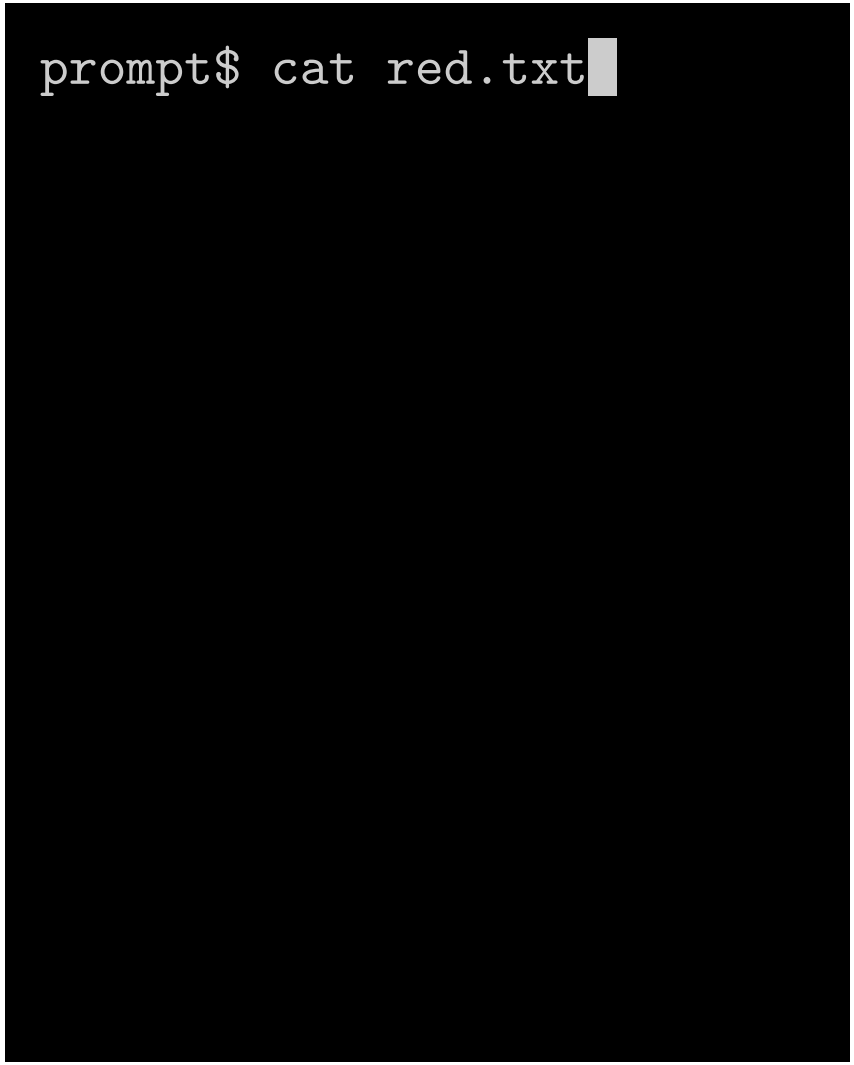
```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
  P
  D
}
```



Red Hat replacement: attempt 3

rhr3.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
  P
  D
}
```



Red Hat replacement: attempt 3

rhr3.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
  P
  D
}
```

```
prompt$ cat red.txt
My fun Red  Hat example.
The tricky case with Red
Hat split across lines.
Multiple Red Hats and Red
Hat split across lines.

Evil cases:
Red
Hat Red
Hat Red Hat Red
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$
```

Red Hat replacement: attempt 3

rhr3.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
  P
  D
}
```

```
prompt$ cat red.txt
My fun Red  Hat example.
The tricky case with Red
Hat split across lines.
Multiple Red Hats and Red
Hat split across lines.

Evil cases:
Red
Hat Red
Hat Red Hat Red
Hat Red Hat Red Hat Red
Hat Red
Corvette
prompt$ ./rhr3.sed red.txt
```

Red Hat replacement: attempt 3

rhr3.sed

```
#!/usr/bin/sed -f
s/Red *Hat/Fedora/g
/Red *$/ {
  N
  s/Red *\n *Hat */Fedora\
/
  P
  D
}
```

```
prompt$ ./rhr3.sed red.txt
My fun Fedora example.
The tricky case with Fedora
split across lines.
Multiple Fedoras and Fedora
split across lines.

Evil cases:
Fedora
Fedora
Fedora Fedora
Fedora Fedora Fedora
Red
Corvette
prompt$
```

The hold space

- We can save text into a buffer called the hold space
- The hold space is used only for temporary storage
- But we can use this space in clever ways
- The get instructions:
  - `g`: Copy the hold space into the pattern space.
  - `G`: Append the hold space to the pattern space.
- The hold instructions:
  - `h`: Copy the pattern space into the hold space.
  - `H`: Append the pattern space to the hold space.
- The exchange instruction:
  - `x`: Swap contents of the pattern space and the hold space.

sed instruction cheat sheet

- `a` : Append text
- `c` : Change text
- `d` : Delete the pattern space
- `g` : Copy hold space to pattern space
- `h` : Copy pattern space to hold space
- `i` : Insert text
- `n` : Read the next input line into the pattern space
- `p` : Print the pattern space
- `q` : Quit
- `s` : Substitute text
- `x` : Exchange pattern and hold spaces
- `y` : Transform characters

sed multi–line instruction cheat sheet

`D` : Delete the first line of the pattern space

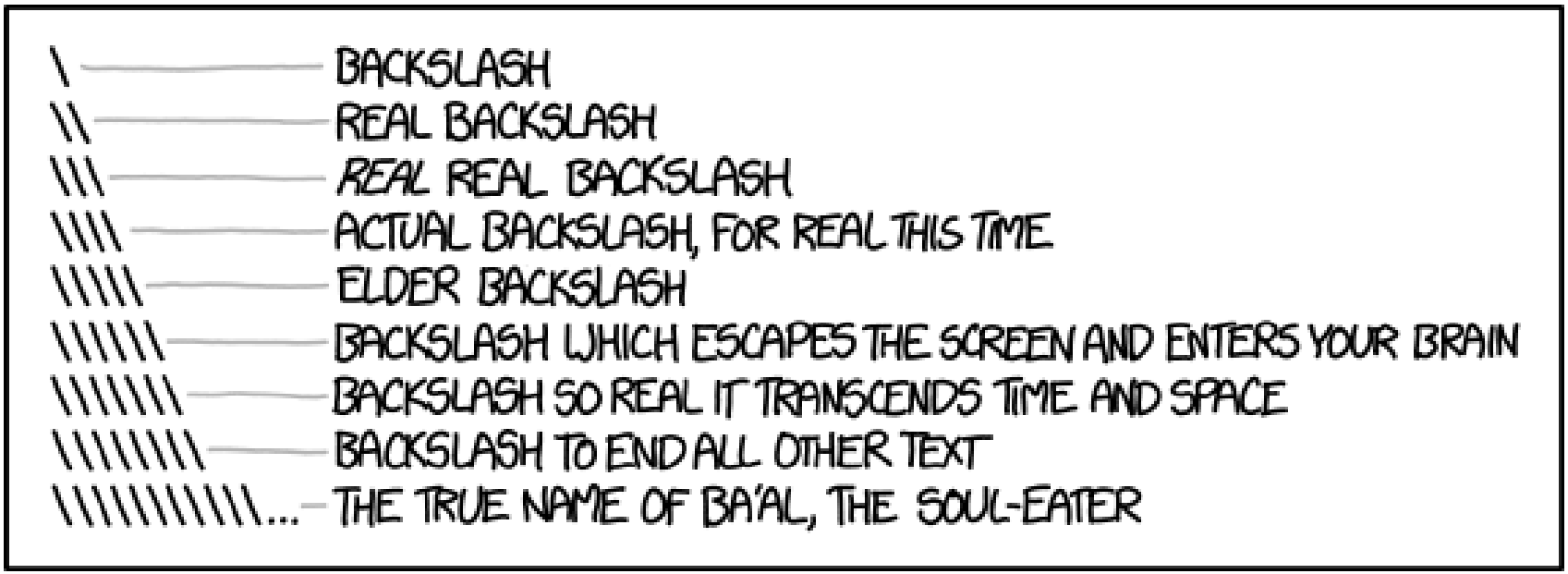
`G` : Append hold space to pattern space

`H` : Append pattern space to hold space

`N` : Append the next input line into the pattern space

`P` : Print the first line of the pattern space

A "close enough" xkcd comic: http://xkcd.com/1638/



End of lecture