ULI101

Week 9

sed

- Stream Editor
- Checks for address match, one line at a time, and performs instruction if address matched
- Prints *all lines* to standard output by default (suppressed by -n option)
- **Syntax**:

```
sed 'address instruction' filename
```

sed

- **Syntax:**

```
sed [-n] 'address instruction' filename
```

- **address**
  - can use a line number, to select a specific line (for example: 5)
  - can specify a range of line numbers (for example: 5,7)
  - can specify a regular expression to select all lines that match (e.g /^happy[0-9]/)
    - Note: when using regular expressions, you must delimit them with a forward-slash "/"
  - default address (if none is specified) will match every line

- **instruction**
  - p - print line(s) that match the address (usually used with -n option)
  - d - delete line(s) that match the address
  - q - quit processing at the first line that matches the address
  - s - substitute text to replace a matched regular expression, similar to vi substitution

sed - Example 1

- Unless you instruct it not to, sed sends all lines - selected or not - to standard output.
  - When you use the –n option on the command line, sed sends only those lines to stdout that you specify with the print "p" command

- **Example 1:** The following command line displays all lines in the readme file that contain the word *line* (all lowercase). In addition, because there is no –n option, sed displays all the lines of input. As a result, sed displays the lines that contain the word line twice.

```
$ sed '/line/ p' readme
Line one.
The second line.
The second line.
The third.
This is line four.
This is line four.
Five.
This is the sixth sentence.
This is line 7.
This is line 7.
Eight and last.
```

sed - Example 2

- **Example 2:** In this example, sed displays part of a file based on line numbers.
  - The Print instruction selects and displays lines 3 through 6.

```
$ sed -n '3,6 p' readme
The third.
This is line four.
Five.
This is the sixth sentence.
```

sed - Example 3

- **Example 3:** The next command line uses the Quit instruction to cause sed to display only the beginning of a file. In this case sed displays the first five lines of text just as a `head -5 lines` command would.
- Remember: sed prints all lines, beginning from the first line, by default. In this example, sed will terminate when the address (in this case, line 5) is matched.

```
$ sed '5 q' readme
Line one.
The second line.
The third.
This is line four.
Five.
```

sed - Example 4

- **Example 4:** This example uses a regular expression as the pattern.
  - The regular expression in the following instruction (^.) matches one character at the beginning of every line that is not empty.
  - The replacement string (between the second and third slashes) contains a backslash escape sequence that represents a TABcharacter (\t) followed by an ampersand (&).
    - The ampersand (&) takes on the value of what the regular expression matched.

```
$ sed 's/^./\t&/' readme
Line one.
The second line.
The third.
...
```

- This type of substitution is useful for indenting a file to create a left margin

sed - Example 5

- **Example 5:** This example uses a regular expression as the pattern again.
  - The regular expression in the following instruction ([0-9][0-9][0-9]$) **matches three digits** at the **end of a line**.
  - The instruction (q) instructs sed to stop processing lines once the regular expression is matched

```
$ sed '/[0-9][0-9][0-9]$/ q' myfile
sfun 11
cool 12
Super 12a
Happy112
```

•The command will process the file, one-line at a time, beginning at the top, and (by default) outputs each line to standard output. Once the regular expression matches, it will display the matched line, and stop processing the file any further.

sed – More Examples

- **Syntax:**

```
sed [-n] 'address instruction' filename
```

```
1.  plym    fury     77      73      2500
2.  chevy   nova     79      60      3000
3.  ford    mustang  65      45      17000
4.  volvo   gl       78      102     9850
5.  ford    ltd      83      15      10500
6.  Chevy   nova     80      50      3500
7.  fiat    600      65      115     450
8.  honda   accord   81      30      6000
9.  ford    thundbd  84      10      17000
10. toyota  tercel   82      180     750
11. chevy   impala   65      85      1550
12. ford    bronco   83      25      9525
```

`sed -n '3,6 p' cars` - display only lines 3 through 6
`sed '5 d' cars` - display all lines except the 5th
`sed '5,8 d' cars` - display all lines except the 5th through 8th
`sed '5 q' cars` - display first 5 lines then quit, same as `head -5 cars`
`sed -n '/chevy/ p' cars` - display only matching lines, same as `grep 'chevy' cars`
`sed '/chevy/ d' cars` - delete all matching lines, same as `grep -v 'chevy' cars`
`sed '/chevy/ q' cars` - display to first line matching regular expression
`sed 's/[0-9]/*/' cars` - substitute first digit on each line with an asterisk
`sed 's/[0-9]/*/g' cars` - substitute every digit on each line with an asterisk
`sed '5,8 s/[0-9]/*/' cars` - substitute only on lines 5 to 8
`sed 's/[0-9][0-9]*/*** & ***/' cars` - surround first number on each line with asterisks

Note that in the last 4 examples above, "*" has no special meaning between the 2nd and 3rd "/"

awk

- **Syntax:**
  - `awk 'pattern {action}' filename`

- The pattern selects lines from the input. The awk utility performs the action on all lines that match the pattern.
- The braces surrounding the action enable awk to differentiate it from the pattern.
- If no pattern is specified, awk selects all lines in the input.
- If no action is specified, awk copies the selected lines to standard output

awk - Patterns

- **Syntax:**
  - `awk 'pattern {action}' filename`

Patterns:

- You can use a regular expression, enclosed within slashes, as a pattern.
- The ~ operator tests whether a field or variable matches a regular expression
- The !~ operator tests for no match.
- You can perform both numeric and string comparisons using relational operators
- You can combine any of the patterns using the Boolean operators || (OR) and && (AND).

awk - Patterns

Relational operators (used in Patterns)

Relational operator	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
>=	Greater than or equal to
>	Greater than

awk - Actions

- The action portion of an awk command causes awk to take that action when it matches a pattern.
- When you do not specify an action, awk performs the default action, which is the print command (explicitly represented as {print}). This action copies the record (normally a line) from the input to standard output.
- When you follow a print command with arguments, awk displays only the arguments you specify.
  - These arguments can be variables or string constants.
- Unless you separate items in a print command with commas, awk catenates them.
  - Commas cause awk to separate the items with the output field separator
- You can include several actions on one line by separating them with semicolons.

awk - Variables

- In addition to supporting user variables, awk maintains program variables.
- You can use both user and program variables in the **pattern** and **action** portions of an awk command.

Variable	Meaning
$0	The current record (as a single variable)
$1–$n	Fields in the current record
FILENAME	Name of the current input file (null for standard input)
FS	Input field separator (default: SPACE or TAB)
NF	Number of fields in the current record
NR	Record number of the current record
OFS	Output field separator (default: SPACE)
ORS	Output record separator (default: NEWLINE)
RS	Input record separator (default: NEWLINE)

awk – Example 1

- Because the pattern is missing, awk selects all lines of input.
- When used without any arguments the print command displays each selected line in its entirety.
- This command copies the input to standard output.

```
$ awk '{ print }' cars
plym    fury     77      73      2500
chevy   nova     79      60      3000
ford    mustang  65      45      17000
volvo   gl       78      102     9850
ford    ltd      83      15      10500
Chevy   nova     80      50      3500
fiat    600      65      115     450
honda   accord   81      30      6000
ford    thundbd  84      10      17000
toyota  tercel   82      180     750
chevy   impala   65      85      1550
ford    bronco   83      25      9525
```

awk – Example 2

- This example has a pattern but no explicit action.
- The slashes indicate that chevy is a regular expression.
- In this case awk selects from the input just those lines that contain the string chevy (lowercase).
- When you do not specify an action, awk assumes the action is print. The following command copies to standard output all lines from the input that contain the string chevy:

```
$ awk '/chevy/' cars
chevy   nova    79      60      3000
chevy   impala  65      85      1550
```

awk – Example 3

- These two examples select all lines from the file (they have no pattern).
- The braces enclose the action; you must always use braces to delimit the action so awk can distinguish it from a pattern.
- These examples display the third field ($3), and the first field ($1) of each selected line, with and without a separating space:

```
$ awk '{print $3, $1}' cars
77 plym
79 chevy
65 ford
78 Volvo
...
```

```
$ awk '{print $3 $1}' cars
77plym
79chevy
65ford
78Volvo
...
```

awk – Example 4

- This example, which includes both a pattern and an action, selects all lines that contain the string chevy and displays the third and first fields from the selected lines:

```
$ awk '/chevy/ {print $3, $1}' cars
79 chevy
65 chevy
```

awk – Example 5

- This example uses the matches operator (~) to select all lines that contain the letter h in the first field ($1), and because there is no explicit action, awk displays all the lines it selects.

```
$ awk '$1 ~ /h/' cars
chevy    nova    79      60      3000
Chevy    nova    80      50      3500
honda    accord  81      30      6000
chevy    impala  65      85      1550
```

awk – Example 6

- The caret (^) in a regular expression forces a match at the beginning of the line or, in this case, at the beginning of the first field, and because there is no explicit action, awk displays all the lines it selects.

```
$ awk '$1 ~ /^h/' cars
honda    accord  81      30      6000
```

awk – Example 7

- This example shows three roles a dollar sign can play within awk.
  - First, a dollar sign followed by a number identifies a field ($3).
  - Second, within a regular expression a dollar sign forces a match at the end of a line or field (5$).
  - Third, within a string a dollar sign represents itself.

```
$ awk '$3 ~ /5$/ {print $3, $1, "$" $5}' cars
65 ford $17000
65 fiat $450
65 chevy $1550
```

awk – Example 8

- Square brackets surround a character class definition.
- In this example, awk selects lines that have a second field that begins with t or m and displays the third and second fields, a dollar sign, and the fifth field.
- Because there is no comma between the “$” and the $5, awk does not put a SPACE between them in the output.

```
$ awk '$2 ~ /^[tm]/ {print $3, $2, "$" $5}' cars
65 mustang $17000
84 thundbd $17000
82 tercel $750
```

awk – Examples 9 & 10

- The equal-to relational operator (==) causes awk to perform a numeric comparison between the third field in each line and the number 83.
- This awk command takes the default action, print, on each line where the comparison is successful.

```
$ awk '$3 == 83' cars
ford    ltd      83      15      10500
ford    bronco   83      25      9525
```

- The next example finds all cars priced (5th field) at or less than $3,000.

```
$ awk '$5 <= 3000' cars
plym    fury     77      73      2500
chevy   nova     79      60      3000
fiat    600      65      115     450
toyota  tercel   82      180     750
chevy   impala   65      85      1550
```

awk – Example 11

- When both sides of a comparison operator are numeric, awk defaults to a numeric comparison. To force a string comparison, double quotes can be used.
- The following examples illustrate the effect of using double quotes.

```
$ awk '"2000" <= $5 && $5 < 9000' cars
plym    fury    77    73    2500
chevy   nova    79    60    3000
Chevy   nova    80    50    3500
fiat    600     65    115   450
honda   accord  81    30    6000
toyota  tercel  82    180   750

$ awk '2000 <= $5 && $5 < 9000' cars
plym    fury    77    73    2500
chevy   nova    79    60    3000
Chevy   nova    80    50    3500
honda   accord  81    30    6000
```

- **Notice that, for example, "2000" is less then "450" as a string, but 2000 is NOT less than 450 as a number**

awk – More EXAMPLES

- **Syntax:**

  awk 'pattern {action}' filename

```
1.  plym     fury      77       73       2500
2.  chevy    nova      79       60       3000
3.  ford     mustang   65       45       17000
4.  volvo    gl        78       102      9850
5.  ford     ltd       83       15       10500
6.  Chevy    nova      80       50       3500
7.  fiat     600       65       115      450
8.  honda    accord    81       30       6000
9.  ford     thundbd   84       10       17000
10. toyota   tercel    82       180      750
11. chevy    impala    65       85       1550
12. ford     bronco    83       25       9525
```

awk 'NR == 2, NR == 4' cars - display the 2nd through 4th lines

awk -F':' '{print $6}' /etc/passwd - specifies that : is input field separator, default is space or tab

awk '$2 ~ /[0-9]/' cars - searches for reg-exp (a digit) only in the second field