CDOT
SICT AR Meeting Area
People

get involved with CDOT

as a Student
as an Open Source Community Member
as a Company

courses

BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840

ULI101 Week 11

Contents [hide]

Regular Expressions [edit]

- In computing, a regular expression (abbreviated regexor regexp) is a sequence of characters that forms a search pattern
- It is mainly for use in pattern matching with strings, or string matching i.e. "find and replace"-like operations
- Uses a backslash (to suppress the meaning of metacharacters. Escaping a character that is not a metacharacter is an error.

Metacharacters [edit]

Metacharacter Description

`.` Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, a.c matches "abc", etc., but [a.c] matches only "a", ".", or "c".

`[ ]` A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z", as does [a-cx-z]. The - character is treated as a literal character if it is the last or the first (after the ^) character within the brackets: [abc-], [-abc]. Note that backslash escapes are not allowed. The] character can be included in a bracket expression if it is the first (after the ^) character: []abc].

`[^ ]` Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than "a", "b", or "c". `[^a-z]` matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.

`^` Matches the starting position within the string. In line-based tools, it matches the starting position of any line.

Real World Mozilla
RHT524
SBR600
SEC520
SPO600
SRT210
ULI101

course projects

Course Project List
Potential Course Projects
Contrib Opportunities

links

CDOT
Planet CDOT
FSOSS

Tools

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Page information

`$` Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.

`(` `)` Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry,). A marked subexpression is also called a block or capturing group. BRE mode requires $$.

`*` Matches the preceding element zero or more times. For example, ab*c matches "ac", "abc", "abbbc", etc. [xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.

Source: https://en.wikipedia.org/wiki/Regular_expression

sed - Stream Editor [edit]

Checks for address match, one line at a time, and performs instruction if address matched. Prints all linesto standard output by default (suppressed by -n option).

Syntax:

```
sed 'address instruction' filepathname
```

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]/`). default address (if none is specified) will match every line

instruction

can be one of the following:

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 expressions, similar to vi substitution

Unless you instruct it not to, sed sends all lines selected or not to standard output. When you use the `-n` optionon the command line, sed sends only certain lines,such as those selected by a Print(p) instruction, to standard output.

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 seven.
This is line seven.
```

```
Eighth and last.
```

In the next 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.
```

The next command line uses the Quit instructionto cause sed to display only the beginning of a file. In this case sed displays the first five lines of linesjust 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.
```

The next 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 () followed by an ampersand (&). The ampersand (&) takes on the value of what the regular expression matched. This type of substitution is useful for indenting a file to create a left margin

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

The next 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. Therefore, this 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 '/[0-9][0-9][0-9]$/q'myfile
sfun 11
cool 12
Super 12a
Happy112
```

More sed examples

```
$ cat 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
```

`sed '3,6 p' cars`	display lines 3 through 6 (these lines will be doubled, since all lines printed by default)
`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 lines matching regular expression, 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]/apple/' cars`	substitute first occurrence of a digit on each linewith the string “apple”
`sed 's/[0-9]/apple/g' cars`	substitute every occurrenceof a digit on each line with the string “apple”
`sed '5,8 s/[0-9]/apple/' cars`	substitute only on lines 5 to 8

awk - Pattern matching and processing [edit]

Checks for pattern match, one line at a time, and performs action if pattern matched

Syntax:

`awk 'pattern {action}' filepathname`

pattern

The pattern selects lines from the input. The awk utility performs the action on all lines that the pattern selects. If a program line does not contain a pattern, awk selects all lines in the input.

action

The braces surrounding the actionenable awk to differentiate it from the pattern. If a program line does not contain an action, awk copies the selected lines to standard output

awk Patterns: (~ and !~) [edit]

- You can use a regular expression, enclosed within slashes, as a pattern.
- The `~` operator tests whether a fieldor variable matches a regular expression
- The `!~` operator tests for no match.
- You can perform both numericand stringcomparisons using relational operators(on next slide)
- You can combine any of the patterns using the Boolean operators `|| (OR)` or `&& (AND)`.

Operator	Meaning
`<`	Less than
`<=`	Less than or equal to
`==`	Equal to
`!=`	Not equal to
`>=`	Greater than or equal to
`>`	Greater than

awk Actions [edit]

- The action portion of a awkcommand causes awkto take that action when it matches a pattern.
- When you do not specify an action, awkperforms 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, awkdisplays only the arguments you specify.
- These arguments can be variables or string constants.
- Unless you separate items in a print command with commas, awk concatenates 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 [edit]

In addition to supporting user variables, awk maintains program variables. You can use both user and program variables in the patternand actionportions of a awk program.

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)

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 program copies the input to standard output.

```
$ awk '{ print }' cars
plym fury 1970 73 2500
chevy malibu 1999 60 3000
ford mustang 1965 45 10000
volvo s80 1998 102 9850
```

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

```
$ awk '/chevy/' cars
chevy malibu 1999 60 3000
chevy malibu 2000 50 3500
chevy impala 1985 85 1550
```

The next example selects all lines from the file (it has no pattern). The braces enclose the action; you must always use braces to delimit the action so awk can distinguish it from the pattern. This example displays the third field ($3), a SPACE(the output field separator, indicated by the comma), and the first field ($1) of each selected line:

```
$ awk '{print $3,$1}' cars
1970 plym
1999 chevy
1965 ford
1998 volvo

$ awk '{print $3 $1}' cars
1970plym
1999chevy
1965ford
1998volvo
```

The next 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
1999 chevy
2000 chevy
1985 chevy
```

The next pattern 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 malibu 1999 60 3000
chevy malibu 2000 50 3500
honda accord 2001 30 6000
chevy impala 1985 85 1550
```

The caret (^) in a regular expression forces a match at the beginning of the lineor, 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 2001 30 6000,
```

The next example shows three roles a dollar sign can play in a awk program.

- First, a dollar sign followed by a number names a field.
- 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
1965 ford $10000
1985 bmw $450
1985 chevy $1550
```

Brackets surround a character class definition. In the next example, awkselects 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 commabetween the “$” and the $5, awkdoes not put a SPACE between them in the output.

```
$ awk'$2 ~ /^[tm]/ {print $3, $2, "$" $5}' cars
1999 malibu $3000
1965 mustang $10000
2003 thundbd $10500
2000 malibu $3500
2004 taurus $17000
```

The equal-to relational operator (==) causes awkto perform a numeric comparison between the third field in each line and the number 1985. The awk command takes the default action, print, on each line where the comparison is true.

```
$ awk'$3 == 1985' cars
bmw 325i 1985 115 450
chevy impala 1985 85 1550
```

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

```
$gawk '$5 <= 3000' cars
plym fury 1970 73 2500
chevy malibu 1999 60 3000
bmw 325i 1985 115 450
toyota rav4 2002 180 750
chevy impala 1985 85 1550
```

When you use double quotationmarks, awk performs textual comparisonsby using the ASCII (or other local) collating sequence as the basis of the comparison. When you need to perform a numeric comparison, do notuse quotation marks. The next example gives the intended result. It is the same as the previous example except it omits the double quotation marks.

```
$ awk '2000 <= $5 && $5 < 9000' cars
plym fury 1970 73 2500
chevy malibu 1999 60 3000
chevy malibu 2000 50 3500
honda accord 2001 30 6000
```

`awk 'pattern {action}' filename`

`awk 'NR == 2, NR == 4' cars`	display the 2nd through 4th lines (default action is to print entireline)
`awk '/chevy/' cars`	display only lines matching regular expression, same as grep 'chevy'cars
`awk '{print $3, $1}' cars`	includes an output field separator (variable OFS, default is space)`
`awk -F':' '{print $6}' /etc/passwd`	specifies that : is input field separator, default is space or tab
`awk '/chevy/ {print $3, $1}' cars`	display third and first fiield of lines matching regular expression
`awk '$3 == 65' cars`	display only lines with a third field value of 65
`awk '$5 <= 3000' cars`	display only lines with a fifth field value that is less than or equal to 3000
`awk '$2 ~ /[0-9]/' cars`	searches for reg-exp (a digit) only in the second field

```
$ cat 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
```

Categories: Pages with syntax highlighting errors | ULI101 | ULI101-2018