5. REGULAR EXPRESSIONS

Elvis uses regular expressions for searching and substitutions. A regular expression is a text string in which some characters have special meanings. This is much more powerful than simple text matching.

5.1 Syntax

Elvis' regexp package treats the following one- or two-character strings (called meta-characters) in special ways:

\(*subexpression***\)**
The \(and \) metacharacters are used to delimit subexpressions. When the regular expression matches a particular chunk of text, Elvis will remember which portion of that chunk matched the *subexpression*. The :s/regexp/newtext/ command makes use of this feature.

^
The ^ metacharacter matches the beginning of a line. If, for example, you wanted to find "foo" at the beginning of a line, you would use a regular expression such as /^foo/. Note that ^ is only a metacharacter if it occurs at the beginning of a regular expression; practically anyplace else, it is treated as a normal character. (Exception: It also has a special meaning inside a [*character-list*] metacharacter, as described below.)

$
The $ metacharacter matches the end of a line. It is only a metacharacter when it occurs at the end of a regular expression; elsewhere, it is treated as a normal character. For example, the regular expression /$$/ will search for a dollar sign at the end of a line.

\<
The \< metacharacter matches a zero-length string at the beginning of a word. A word is considered to be a string of 1 or more letters, digits, or underscores. A word can begin at the beginning of a line or after 1 or more non-alphanumeric characters.

\>
The \> metacharacter matches a zero-length string at the end of a word. A word can end at the end of the line or before 1 or more non-alphanumeric characters. For example, /\<end\>/ would find any instance of the word "end", but would ignore any instances of e-n-d inside another word such as "calendar".

\@
When you're performing a search in visual mode, and the cursor is on a word before you start typing the search command, then \@ matches the word at the cursor.

\=

Ordinarily, the visual mode search command leaves the cursor on the first character of the matching text that it finds. If your regular expression includes a \= metacharacter, then it will leave the cursor at the position that matched the \=. For example, if you place \= at the end of your regular expression, then the cursor will be left after the matching text instead of at the start of it.

.

The . metacharacter matches any single character.

NOTE: If the magic option is turned off, then . is treated as an ordinary, literal character. You should use \. to get the meta-character version in this case.

[*character-list***]**

This matches any single character from the *character-list*. Inside the *character-list*, you can denote a span of characters by writing only the first and last characters, with a hyphen between them. If the *character-list* is preceded by a ^ character, then the list is inverted -- it will match any character that *isn't* mentioned in the list. For example, /[a-zA-Z]/ matches any ASCII letter, and /[^]/ matches anything other than a blank.

There is no way to quote the ']' or '-' characters, which means that if you want to include those characters as members of the list, you must place them in positions where they couldn't be mistaken for the end of the list or a range. Specifically, ']' can appear only as the first character in the list (immediately after the "[" or "[^" that starts the list) or as the last character in a range. '-' can appear there too, or immediately after the last character of a range. For example, [])}] matches a closing bracket, parentheses, or curly brace. [^-+] matches any character except '+' or '-'. Probably the trickiest example, []-]-] matches a closing bracket or a '-'. (Note that the range "]-]" matches a single bracket; we wrote it this way so that the following "-" would be in a context where it couldn't be mistaken for a range and so must be a literal '-' character.)

There are also special cases for some common character lists. When one of the following special symbols appears in a character list, the list will include all appropriate characters for that symbol *including the non-ascii characters* as indicated by the digraph table. Note that he brackets around these symbols are *in addition to* the brackets around the whole class. For example, /[[:alpha:]]/ matches any single letter, and /[[:alpha:]_][[:alnum:]_]*/ matches any C identifier.

SPECIAL SYMBOL	INCLUDED CHARACTERS
[:alnum:]	all letters and digits
[:alpha:]	all letters
[:ascii:]	all ASCII characters
[:blank:]	the space and tab characters

[:cntrl:]	ASCII control characters
[:digit:]	all digits
[:graph:]	all printable characters excluding space
[:lower:]	all lowercase letters
[:print:]	all printable characters including space
[:punct:]	all punctuation characters
[:space:]	all whitespace characters except linefeed
[:upper:]	all uppercase characters
[:xdigit:]	all hexadecimal digits

NOTE: If the magic option is turned off, then the opening [is treated as an ordinary, literal character. To get the meta-character behavior, you should use \[*character-list*] in this case.

\s, \S, \d, \D, \w, \W, \p, and \P
These are all shortcuts for certain character lists. The lowercase \s, \d, \w, and \p symbols match (respectively) any whitespace character, digit, alphanumeric character, or any printable character. The uppercase versions are the opposites; they match any single character that the lowercase versions *don't* match.

\0, \a, \b, \f, \r, and \t
These are control characters, just as they would be in C strings. Note that there is no \n.

\{*n***\} or \{***n***}**
This is a closure operator, which means that it can only be placed after something that matches a single character. It controls the number of times that the single-character expression should be repeated. The \{*n*\} or \{*n*} operator, in particular, means that the preceding expression should be repeated exactly *n* times. For example, /^-\{80\}$/ matches a line of eighty hyphens, and /\<[[:alpha:]]\{4}\>/ matches any four-letter word.

\{*n*,*m***\} or \{***n*,*m***}**
This is a closure operator which means that the preceding single-character expression should be repeated between *n* and *m* times, inclusive. If the *m* is omitted (but the comma is present) then *m* is taken to be infinity. For example, /"[^"]\{3,5\}"/ matches any pair of quotes which contains three, four, or five non-quote characters. /.\{81,}/ matches any line which contains more than 80 characters.

*****
The * metacharacter is a closure operator which means that the preceding single-character expression can be repeated zero or more times. It is equivalent to \{0,\}. For example, /.*/ matches a whole line.

NOTE: If the magic option is turned off, then * is treated as an ordinary, literal character. You should use * to get the meta-character version in this case.

\+
The \+ metacharacter is a closure operator which means that the preceding single-character expression can be repeated one or more times. It is equivalent to \{1,\}. For example, /.\+/ matches a whole line, but only if the line contains at least one character. It doesn't match empty lines.

\?
The \? metacharacter is a closure operator which indicates that the preceding single-character expression is optional -- that is, that it can occur 0 or 1 times. It is equivalent to \{0,1\}. For example, /no[-]\?one/ matches "no one", "no-one", or "noone".

Anything else is treated as a normal character which must exactly match a character from the scanned text. The special strings may all be preceded by a backslash to force them to be treated normally.

5.2 Substitutions

The :s command has at least two arguments: a regular expression, and a substitution string. The text that matched the regular expression is replaced by text which is derived from the substitution string.

You can use any punctuation character to delimit the regular expression and the replacement text. The first character after the command name is used as the delimiter. Most folks prefer to use a slash character most of the time, but if either the regular expression or the replacement text uses a lot of slashes, then some other punctuation character may be more convenient.

Most other characters in the substitution string are copied into the text literally but a few have special meaning:

SYMBOL	MEANING
^M	Insert a newline (instead of a carriage-return)
&	Insert a copy of the original text
~	Insert a copy of the previous replacement text
\1	Insert a copy of that portion of the original text which matched the first set of \(\) parentheses
\2-\9	Do the same for the second (etc.) pair of \(\)
\U	Convert following characters to uppercase
\L	Convert following characters to lowercase
\E	End the effect of \U or \L
\u	Convert the next character to uppercase
\l	Convert the next character to lowercase
\#	Insert the line number, as a string of digits
\0	Insert a nul character
\a	Insert a bell character
\b	Insert a backspace character
\f	Insert a form-feed character
\n	Insert a line-feed character

\r	Insert a carriage-return character
\t	Insert a tab character

These may be preceded by a backslash to force them to be treated normally. The delimiting character can also be preceeded by a backslash to include it in either the regular expression or the substitution string.

Traditionally \0 was a synonym for the & symbol -- they both inserted a copy of the matching text. Elvis breaks from tradition here to make \0 insert a NUL character because there would otherwise be no way to have a substitution insert a NUL character.

5.3 Options

Elvis has two options which affect the way regular expressions are used. These options may be examined or set via the :set command.

The first option is called "[no]magic". This is a boolean option, and it is "magic" (TRUE) by default. While in magic mode, all of the meta-characters behave as described above. In nomagic mode, the ., [...], and * characters loose their special meaning unless preceeded by a backslash. Also, in substitution text the & and ~ characters are treated literally unless preceeded by a backslash.

The second option is called "[no]ignorecase". This is a boolean option, and it is "noignorecase" (FALSE) by default. While in ignorecase mode, the searching mechanism will not distinguish between an uppercase letter and its lowercase form, except in a character list metacharacter. In noignorecase mode, uppercase and lowercase are treated as being different.

Also, the "[no]wrapscan" and "[no]autoselect" options affect searches.

5.4 Examples

This example changes every occurrence of "utilize" to "use":

```
:%s/utilize/use/g
```

This example deletes all whitespace that occurs at the end of a line anywhere in the file.

```
:%s/\s\+$//
```

This example converts the current line to uppercase:

```
:s/.*/\U&/
```

This example underlines each letter in the current line, by changing it into an "underscore backspace letter" sequence. (The ^H is entered as "control-V backspace".):

```
:s/[a-zA-Z]/_^H&/g
```

This example locates the last colon in a line, and swaps the text before the colon with the text after the colon. The first \(\) pair is used to delimit the stuff before the colon, and the second pair delimit the stuff after. In the substitution text, \1 and \2 are given in reverse order to perform the swap:

```
:s/\(.*\):\(.*\)/\2:\1/
```