A

Regular Expressions

In This Appendix

A *regular expression* defines a set of one or more strings of characters. A simple string of characters is a regular expression that defines one string of characters: itself. A more complex regular expression uses letters, numbers, and special characters to define many different strings of characters. A regular expression is said to *match* any string it defines.

This appendix describes the regular expressions used by ed, vim, emacs, grep, awk/mawk/gawk, sed, Perl, and many other utilities. Refer to page 517 for more information on Perl regular expressions. The regular expressions used in shell ambiguous file references are different and are described in "Filename Generation/Pathname Expansion" on page 136.

CHARACTERS

As used in this appendix, a *character* is any character *except* a NEWLINE. Most characters represent themselves within a regular expression. A *special character,* also called a *metacharacter,* is one that does not represent itself. If you need to use a special character to represent itself, you must quote it as explained on page 891.

DELIMITERS

A character called a *delimiter* usually marks the beginning and end of a regular expression. The delimiter is always a special character for the regular expression it delimits (that is, it does not represent itself but marks the beginning and end of the expression). Although vim permits the use of other characters as a delimiter and grep does not use delimiters at all, the regular expressions in this appendix use a forward slash (/) as a delimiter. In some unambiguous cases, the second delimiter is not required. For example, you can sometimes omit the second delimiter when it would be followed immediately by RETURN.

SIMPLE STRINGS

The most basic regular expression is a simple string that contains no special characters except the delimiters. A simple string matches only itself (Table A-1). In the examples in this appendix, the strings that are matched are underlined and look like this.

Table A-1 Simple strings

Regular expression	Matches	Examples
/ring/	ring	ring, spring, ringing, stringing
/Thursday/	Thursday	Thursday, Thursday's
/or not/	or not	or not, poor nothing

SPECIAL CHARACTERS

You can use special characters within a regular expression to cause the regular expression to match more than one string. A regular expression that includes a

special character always matches the longest possible string, starting as far toward the beginning (left) of the line as possible.

PERIODS

A period (.) matches any character (Table A-2).

Table A-2 Periods

Regular expression	Matches	Examples
/ .alk/	All strings consisting of a SPACE followed by any character followed by alk	will talk, may balk
/.ing/	All strings consisting of any character preceding ing	sing song, ping, before inglenook

BRACKETS

Brackets ([]) define a *character class*[1] that matches any single character within the brackets (Table A-3). If the first character following the left bracket is a caret (^), the brackets define a character class that matches any single character not within the brackets. You can use a hyphen to indicate a range of characters. Within a character-class definition, backslashes and asterisks (described in the following sections) lose their special meanings. A right bracket (appearing as a member of the character class) can appear only as the first character following the left bracket. A caret is special only if it is the first character following the left bracket. A dollar sign is special only if it is followed immediately by the right bracket.

Table A-3 Brackets

Regular expression	Matches	Examples
/[bB]ill/	Member of the character class b and B followed by ill	bill, Bill, billed
/t[aeiou].k/	t followed by a lowercase vowel, any character, and a k	talkative, stink, teak, tanker
/# [6–9]/	# followed by a SPACE and a member of the character class 6 through 9	# 60, # 8:, get # 9
/[^a–zA–Z]/	Any character that is not a letter (ASCII character set only)	1, 7, @, ., }, Stop!

1. GNU documentation calls these List Operators and defines Character Class operators as expressions that match a predefined group of characters, such as all numbers (see Table V-32 on page 864).

ASTERISKS

An asterisk can follow a regular expression that represents a single character (Table A-4). The asterisk represents *zero* or more occurrences of a match of the regular expression. An asterisk following a period matches any string of characters. (A period matches any character, and an asterisk matches zero or more occurrences of the preceding regular expression.) A character-class definition followed by an asterisk matches any string of characters that are members of the character class.

Table A-4 Asterisks

Regular expression	Matches	Examples
/ab*c/	a followed by zero or more b's followed by a c	ac, abc, abbc, debbcaabbbc
/ab.*c/	ab followed by zero or more characters followed by c	abc, abxc, ab45c, xab 756.345 x cat
/t.*ing/	t followed by zero or more characters followed by ing	thing, ting, I thought of going
/[a–zA–Z]*/	A string composed only of letters and SPACES	1. any string without numbers or punctuation!
/(.*)/	As long a string as possible between (and)	Get (this) and (that);
/([^)]*)/	The shortest string possible that starts with (and ends with)	(this), Get (this and that)

CARETS AND DOLLAR SIGNS

A regular expression that begins with a caret (^) can match a string only at the beginning of a line. In a similar manner, a dollar sign ($) at the end of a regular expression matches the end of a line. The caret and dollar sign are called anchors because they force (anchor) a match to the beginning or end of a line (Table A-5).

Table A-5 Carets and dollar signs

Regular expression	Matches	Examples
/^T/	A T at the beginning of a line	This line..., That Time..., In Time
/^+[0–9]/	A plus sign followed by a digit at the beginning of a line	+5 +45.72, +759 Keep this...
/:$/	A colon that ends a line	...below:

QUOTING SPECIAL CHARACTERS

You can quote any special character (but not parentheses [except in Perl; page 521] or a digit) by preceding it with a backslash (Table A-6). Quoting a special character makes it represent itself.

Table A-6 Quoted special characters

Regular expression	Matches	Examples
/end\./	All strings that contain end followed by a period	The end., send., pretend.mail
/ \\/	A single backslash	\
/ */	An asterisk	*.c, an asterisk (*)
/ \[5\]/	[5]	it was five [5]
/and\/or/	and/or	and/or

RULES

The following rules govern the application of regular expressions.

LONGEST MATCH POSSIBLE

A regular expression always matches the longest possible string, starting as far toward the beginning of the line as possible. Perl calls this type of mach a *greedy match* (page 520). For example, given the string

```
This (rug) is not what it once was (a long time ago), is it?
```

the expression **/Th.*is/** matches

```
This (rug) is not what it once was (a long time ago), is
```

and **/(.*)/** matches

```
(rug) is not what it once was (a long time ago)
```

However, **/([^)]*)/** matches

```
(rug)
```

Given the string

```
singing songs, singing more and more
```

the expression **/s.*ing/** matches

```
singing songs, singing
```

and **/s.*ing song/** matches

```
singing song
```

EMPTY REGULAR EXPRESSIONS

Within some utilities, such as vim and less (but not grep), an empty regular expression represents the last regular expression you used. For example, suppose you give vim the following Substitute command:

```
:s/mike/robert/
```

If you then want to make the same substitution again, you can use the following command:

```
:s//robert/
```

Alternatively, you can use the following commands to search for the string **mike** and then make the substitution

```
/mike/
:s//robert/
```

The empty regular expression (**//**) represents the last regular expression you used (**/mike/**).

BRACKETING EXPRESSIONS

You can use quoted parentheses, \(and \), to *bracket* a regular expression. (However, Perl uses unquoted parentheses to bracket regular expressions; page 521.) The string that the bracketed regular expression matches can be recalled, as explained in "Quoted Digit." A regular expression does not attempt to match quoted parentheses. Thus a regular expression enclosed within quoted parentheses matches what the same regular expression without the parentheses would match. The expression **/\(rexp\)/** matches what **/rexp/** would match; **/a\(b*\)c/** matches what **/ab*c/** would match.

You can nest quoted parentheses. The bracketed expressions are identified only by the opening \(, so no ambiguity arises in identifying them. The expression **/\([a–z]\([A–Z]*\)x\)/** consists of two bracketed expressions, one nested within the other. In the string **3 t dMNORx7 l u**, the preceding regular expression matches **dMNORx**, with the first bracketed expression matching **dMNORx** and the second matching **MNOR**.

THE REPLACEMENT STRING

The vim and sed editors use regular expressions as search strings within Substitute commands. You can use the ampersand (**&**) and quoted digits (***n***) special characters to represent the matched strings within the corresponding replacement string.

AMPERSAND

Within a replacement string, an ampersand (&) takes on the value of the string that the search string (regular expression) matched. For example, the following vim Substitute command surrounds a string of one or more digits with **NN**. The ampersand in the replacement string matches whatever string of digits the regular expression (search string) matched:

```
:s/[0-9][0-9]*/NN&NN/
```

Two character-class definitions are required because the regular expression **[0–9]*** matches *zero* or more occurrences of a digit, and *any* character string constitutes zero or more occurrences of a digit.

QUOTED DIGIT

Within the search string, a bracketed regular expression, **\(xxx\)** [**(xxx)** in Perl], matches what the regular expression would have matched without the quoted parentheses, **xxx**. Within the replacement string, a quoted digit, ***\n***, represents the string that the bracketed regular expression (portion of the search string) beginning with the ***n***th \(matched. (Perl accepts a quoted digit for this purpose, but Perl's preferred style is to precede the digit with a dollar sign [***$n;*** page 521]). For example, you can take a list of people in the form

```
last-name, first-name initial
```

and put it in the form

```
first-name initial last-name
```

with the following vim command:

```
:1,$s/\([^,]*\), \(.*\)/\2 \1/
```

This command addresses all lines in the file (**1,$**). The Substitute command (**s**) uses a search string and a replacement string delimited by forward slashes. The first bracketed regular expression within the search string, **\([^,]*\)**, matches what the same unbracketed regular expression, **[^,]***, would match: zero or more characters not containing a comma (the **last-name**). Following the first bracketed regular expression are a comma and a SPACE that match themselves. The second bracketed expression, **\(.*\)**, matches any string of characters (the **first-name** and **initial**).

The replacement string consists of what the second bracketed regular expression matched (**\2**), followed by a SPACE and what the first bracketed regular expression matched (**\1**).

EXTENDED REGULAR EXPRESSIONS

This section covers patterns that use an extended set of special characters. These patterns are called *full regular expressions* or *extended regular expressions*. In addition

to ordinary regular expressions, Perl and vim provide extended regular expressions. The three utilities egrep, grep when run with the –E option (similar to egrep), and mawk/gawk provide all the special characters included in ordinary regular expressions, except for \(and \), as well those included in extended regular expressions.

Two of the additional special characters are the plus sign (+) and the question mark (?). They are similar to *, which matches *zero* or more occurrences of the previous character. The plus sign matches *one* or more occurrences of the previous character, whereas the question mark matches *zero* or *one* occurrence. You can use any one of the special characters *, +, and ? following parentheses, causing the special character to apply to the string surrounded by the parentheses. Unlike the parentheses in bracketed regular expressions, these parentheses are not quoted (Table A-7).

Table A-7 Extended regular expressions

Regular expression	Matches	Examples
/ab+c/	a followed by one or more b's followed by a c	yabcw, abbc57
/ab?c/	a followed by zero or one b followed by c	back, abcdef
/(ab)+c/	One or more occurrences of the string ab followed by c	zabcd, ababc!
/(ab)?c/	Zero or one occurrence of the string ab followed by c	xc, abcc

In full regular expressions, the vertical bar (|) special character is a Boolean OR operator. Within vim, you must quote the vertical bar by preceding it with a backslash to make it special (\|). A vertical bar between two regular expressions causes a match with strings that match the first expression, the second expression, or both. You can use the vertical bar with parentheses to separate from the rest of the regular expression the two expressions that are being ORed (Table A-8).

Table A-8 Full regular expressions

Regular expression	Meaning	Examples
/ab|ac/	Either ab or ac	ab, ac, abac *(**abac** is two matches of the regular expression)*
/^Exit|^Quit/	Lines that begin with Exit or Quit	Exit, Quit, No Exit
/(D|N)\. Jones/	D. Jones or N. Jones	P.D. Jones, N. Jones

APPENDIX SUMMARY

A regular expression defines a set of one or more strings of characters. A regular expression is said to match any string it defines.

In a regular expression, a special character is one that does not represent itself. Table A-9 lists special characters.

Table A-9 Special characters

Character	Meaning
.	Matches any single character
*	Matches zero or more occurrences of a match of the preceding character
^	Forces a match to the beginning of a line
$	A match to the end of a line
\	Quotes special characters
\<	Forces a match to the beginning of a word
\>	Forces a match to the end of a word

Table A-10 lists ways of representing character classes and bracketed regular expressions.

Table A-10 Character classes and bracketed regular expressions

Class	Defines
[*xyz*]	Defines a character class that matches *x*, *y*, or *z*
[^*xyz*]	Defines a character class that matches any character except *x*, *y*, or *z*
[*x–z*]	Defines a character class that matches any character *x* through *z* inclusive
\(*xyz*\)	Matches what *xyz* matches (a bracketed regular expression; not Perl)
(*xyz*)	Matches what *xyz* matches (a bracketed regular expression; Perl only)

In addition to the preceding special characters and strings (excluding quoted parentheses, except in vim), the characters in Table A-11 are special within full, or extended, regular expressions.

Table A-11 Extended regular expressions

Expression	Matches
+	Matches one or more occurrences of the preceding character
?	Matches zero or one occurrence of the preceding character

Table A-11 Extended regular expressions (continued)

Expression	Matches
(*xyz*)+	Matches one or more occurrences of what ***xyz*** matches
(*xyz*)?	Matches zero or one occurrence of what ***xyz*** matches
(*xyz*)*	Matches zero or more occurrences of what ***xyz*** matches
xyz*|*abc	Matches either what ***xyz*** or what ***abc*** matches (use | in vim)
(*xy*|*ab*)*c*	Matches either what ***xyc*** or what ***abc*** matches (use | in vim)

Table A-12 lists characters that are special within a replacement string in sed, vim, and Perl.

Table A-12 Replacement strings

String	Represents
&	Represents what the regular expression (search string) matched
n	A quoted number, ***n***, represents what the ***n***th bracketed regular expression in the search string matched
$*n*	A number preceded by a dollar sign, ***n***, represents what the ***n***th bracketed regular expression in the search string matched (Perl only)