Regular Expressions
On the 28th of April 2012 the contents of the English as well as German Wikibooks and Wikipedia projects were licensed under Creative Commons Attribution-ShareAlike 3.0 Unported license. A URI to this license is given in the list of figures on page 41. If this document is a derived work from the contents of one of these projects and the content was still licensed by the project under this license at the time of derivation this document has to be licensed under the same, a similar or a compatible license, as stated in section 4b of the license. The list of contributors is included in chapter Contributors on page 39. The licenses GPL, LGPL and GFDL are included in chapter Licenses on page 45, since this book and/or parts of it may or may not be licensed under one or more of these licenses, and thus require inclusion of these licenses. The licenses of the figures are given in the list of figures on page 41. This PDF was generated by the \LaTeX typesetting software. The \LaTeX source code is included as an attachment (source.7z.txt) in this PDF file. To extract the source from the PDF file, you can use the pdfdetach tool including in the poppler suite, or the \url{http://www.pdflabs.com/tools/pdfkt-the-pdf-toolkit/} utility. Some PDF viewers may also let you save the attachment to a file. After extracting it from the PDF file you have to rename it to source.7z. To uncompress the resulting archive we recommend the use of \url{http://www.7-zip.org/}. The \LaTeX source itself was generated by a program written by Dirk Hünniger, which is freely available under an open source license from \url{http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger/wb2pdf}. 
Contents

1 Introduction 3

2 Syntaxes 7

3 Simple Regular Expressions 9
   3.1 Examples ........................................ 10
   3.2 Use in Tools .................................... 10

4 Basic Regular Expressions 11
   4.1 Use in Tools .................................... 11

5 Perl-Compatible Regular Expressions 13
   5.1 Examples ........................................ 13
   5.2 Use in Tools .................................... 15
   5.3 See also ......................................... 16
   5.4 Links ............................................. 16

6 POSIX Basic Regular Expressions 17
   6.1 External links .................................. 20
   6.2 Use in Tools .................................... 20
   6.3 Links ............................................. 20

7 POSIX-Extended Regular Expressions 21
   7.1 Use in Tools .................................... 23
   7.2 Links ............................................. 23

8 Non-POSIX Basic Regular Expressions 25
   8.1 Use in Tools .................................... 25

9 Emacs Regular Expressions 27
   9.1 Use in Tools .................................... 27
   9.2 Links ............................................. 27

10 Shell Regular Expressions 29
   10.1 Use in Tools .................................... 29

11 Implementation 31

12 Glossary 33

13 Links 37
   13.1 See also ......................................... 37
13.2 External Links ......................................................... 37

14 Contributors ....................................................... 39

List of Figures ......................................................... 41

15 Licenses .............................................................. 45
   15.1 GNU GENERAL PUBLIC LICENSE ............................. 45
   15.2 GNU Free Documentation License ............................ 46
   15.3 GNU Lesser General Public License .......................... 47
1 Introduction

1.0.1 What is a regular expression?

A regular expression is a method of representing a string matching pattern. Regular expressions enable strings that match a particular pattern within textual data records to be located and modified and they are often used within utility programs and programming languages that manipulate textual data. Regular expressions are extremely powerful.

1.0.2 Example applications

Various software applications use regular expressions to locate, select or modify particular sections text. For example, a regular expression could be used to:

- replace the word "snake" with the word "serpent" throughout an entire piece of text
- locate pieces of text containing the words "fox" and "sheep" on the same line

1.0.3 Regular expression components

Regular expressions are made up of three types of components:

- anchors used to specify the position of the pattern in relation to a line of text.
- character sets used to match one or more characters in a single position.
- modifiers used to specify how many times a character set is repeated.

1.0.4 Syntax varies across application programs

The syntax of regular expressions varies across application programs. For example the shell uses a limited form of regular expression called shell regular expressions for filename substitution, whereas AWK\(^1\) uses a superset of extended regular expressions syntax.

1.0.5 Supporting Software

Regular expressions are supported by various software tools, including command line tools, plain text editors and programming languages. Most of these tools are available for various computing platforms, including Linux, Windows and Mac OS X. The tools use slightly different syntax styles. Let’s look at some notable ones.

\(^1\) http://en.wikibooks.org/wiki/AWK
The tools that use regular expressions are enlisted below:
A regular expression can be considered to be a little computer program that finds or isolates a subset of a larger set of text. In the same way that an ordinary computer program needs a computer to execute it, a regular expression needs a software application to interpret it — to give it meaning.

For example, a regular expression can be used to tell an editor to find the next occurrence of the word "Chapter" followed by several spaces and digits. Or you can use a regular expression to tell the UNIX grep command to show only those lines of a file that contain the word "Wiki" followed by either the word "Books" or the word-fragment "pedia". We will discuss the exact syntax of such regular expressions in the next chapter.

Category:Regular Expressions

pt:Expressões regulares/Introdução

---

2  http://en.wikibooks.org/wiki/Category%3ARegular%20Expressions

3  http://pt.wikibooks.org/wiki/Express%C3%B5es%20regulares%2FIntrodu%C3%A7%C3%A3o
2 Syntaxes

There are several variants of regular expressions. These variants differ not only in their concrete syntax but also in their capabilities. Individual tools that support regular expressions also have their own peculiarities.

- ../Simple Regular Expressions/\(^1\) - widely used for backwards compatibility, but deprecated on POSIX-compliant systems.
- ../Basic Regular Expressions/\(^2\) - used by some Unix shell tools
- ../Perl-Compatible Regular Expressions/\(^3\) - used by Perl and some application programs
- ../POSIX Basic Regular Expressions/\(^4\) - provides extensions for consistency between utility programs. These extensions are not supported by some traditional implementations of Unix tools.
- ../POSIX-Extended Regular Expressions/\(^5\) - may be supported by some Unix utilities via the -E command line switch
- ../Non-POSIX Basic Regular Expressions/\(^6\) - provides additional character classes not supported by POSIX
- ../Emacs Regular Expressions/\(^7\) - used by the Emacs editor
- ../Shell Regular Expressions/\(^8\) - a limited form of regular expression used for pattern matching and filename substitution

2.0.6 Greedy expressions

Quantifiers such as * and + match as much as they can: they are greedy. For some uses, their greediness does not fit. For example, let as assume you want to find the first string enclosed in quotation marks, in the following text:

These words include "cat", "mat", and "pat".

The pattern ".*" matches the italicized part of the text below, that is, "cat", "mat", and "pat" instead of the desired "cat":

These words include "cat", "mat", and "pat".

To fix this, some flavours of regular expressions provide non-greedy operators such as *?, +?, and }?. In PHP, adding a "U" at the end of the regexp makes the quantifier non-greedy,
as in "/\.*\u/\". In flavours that support neither of the two options, you can specify what is 
not to be matched, as in ("[^"]*\") to fix the discussed example. However, when dealing 
with bracketed expressions, (\[[^\[\]\]*\]\]) fails to match on A B C [[D E/ F G]].

2.0.7 Comparison table

A comparison table or matrix that shows which features or flavors of regular expressions are 
available in which tool or programming language is available from regular-expressions.info\(^9\). 

\(^9\) http://www.regular-expressions.info/refflavors.html 
\(^10\) http://en.wikibooks.org/wiki/Category%3ARegular%20Expressions
3 Simple Regular Expressions

The Simple Regular Expression syntax is widely used on Unix based systems for the purposes of backwards compatibility. Most regular-expression-aware Unix utilities, such as `grep` and `sed`, use it by default while providing support for extended regular expressions with command line arguments (see below). This syntax is deprecated on POSIX compliant systems and should not be used by new utilities.

When simple regular expression syntax is being used, most characters, except metacharacters, are treated as literal characters and match only themselves (for example, ”a” matches ”a”, ”(bc” matches ”(bc”, etc).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>The dot operator matches any single character.</td>
</tr>
<tr>
<td>[]</td>
<td>boxes enable a single character to be matched against character lists or character ranges.</td>
</tr>
<tr>
<td>[^]</td>
<td>A complement box enables a single character not within a character list or character range to be matched.</td>
</tr>
<tr>
<td>^</td>
<td>A caret anchor matches the start of the line (or any line, when applied in multiline mode)</td>
</tr>
<tr>
<td>$</td>
<td>A dollar anchor matches the end of the line (or any line, when applied in multiline mode)</td>
</tr>
<tr>
<td>( )</td>
<td>parentheses are used to define a marked subexpression. The matched text section can be recalled at a later time.</td>
</tr>
<tr>
<td>\n</td>
<td>Where n is a digit from 1 to 9; matches what the n th marked subexpression matched. This irregular construct has not been adopted in the extended regular expression syntax.</td>
</tr>
<tr>
<td>*</td>
<td>A single character expression followed by * matches zero or more copies of the expression. For example, ”ab<em>c” matches ”ac”, ”abc”, ”abbc” etc. ”[xyz]</em>” matches ””, ”x”, ”y”, ”zx”, ”zyx”, and so on.</td>
</tr>
<tr>
<td>\n *</td>
<td>where n is a digit from 1 to 9, matches zero or more iterations of what the n th marked subexpression matched. For example, ”(a.)c\1*” matches ”abcab” and ”abcabab” but not ”abcac”.</td>
</tr>
</tbody>
</table>
| An expression enclosed in ”\(” and ”\)” followed by \* is deemed to be invalid. In some cases (e.g. /usr/bin/xpg4/grep of SunOS 5.8), it matches zero or more iterations of the string that the enclosed expression matches. In other cases (e.g. /usr/bin/grep of SunOS 5.8), it matches what the enclosed expression matches, followed by a literal ”*”.


3.1 Examples

Examples:

- "^[hc]at"
  - Matches hat and cat but only at the beginning of a line.
- "^[hc]at$"
  - Matches hat and cat but only at the end of a line.

3.2 Use in Tools

Tools and languages that utilize this regular expression syntax include:

- Grep\(^1\)
- sed\(^2\)

---

4 Basic Regular Expressions

Basic Regular Expressions: Note that particular implementations of regular expressions interpret the backslash symbol differently in front of some of the metacharacters. For example, egrep and perl interpret unbackslashed parentheses and vertical bars as metacharacters, reserving the backslashed versions to mean the literal characters themselves. Old versions of grep did not support the pipe alternation operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>The dot operator matches any single character.</td>
</tr>
<tr>
<td>[]</td>
<td>boxes enable a single character to be matched against a character lists or character range.</td>
</tr>
<tr>
<td>[^]</td>
<td>A compliment box enables a single character not within in a character list or character range to be matched.</td>
</tr>
<tr>
<td>*</td>
<td>An asterisk specifies zero or more characters to match.</td>
</tr>
<tr>
<td>^</td>
<td>The caret anchor matches the beginning of the line</td>
</tr>
<tr>
<td>$</td>
<td>The dollar anchor matches the end of the line</td>
</tr>
</tbody>
</table>

Examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;.at&quot;</td>
<td>any three-character string like hat, cat or bat</td>
</tr>
<tr>
<td>&quot;[hc]at&quot;</td>
<td>hat and cat</td>
</tr>
<tr>
<td>&quot;[^b]at&quot;</td>
<td>all the matched strings from the regex &quot;.at&quot; except bat</td>
</tr>
<tr>
<td>&quot;[hc]at$&quot;</td>
<td>hat and cat but only at the beginning of a line</td>
</tr>
<tr>
<td>&quot;[hc]at$&quot;</td>
<td>hat and cat but only at the end of a line</td>
</tr>
</tbody>
</table>

Since many ranges of characters depends on the chosen locale setting (e.g., in some settings letters are organized as abc..yzABC..YZ while in some others as aAbBcC..yYzZ).

The ..../Posix Basic Regular Expressions/\(^1\) syntax provided extensions for consistency between utility programs such as grep, sed and awk. These extensions are not supported by some traditional implementations of Unix tools.

4.1 Use in Tools

Tools and languages that utilize this regular expression syntax include: TBD

---

\(^1\) Chapter 5.4 on page 16
5 Perl-Compatible Regular Expressions

Perl has a richer and more predictable syntax than even the ../POSIX Extended Regular Expressions/\(^1\) syntax. An example of its predictability is that \ always quotes a non-alphanumeric character. An example of something that is possible to specify with Perl but not POSIX is whether part of the match wanted to be greedy or not. For instance in the pattern /a.*b/, the .* will match as much as it can, while in the pattern /a.*?b/, .*? will match as little. So given the string "a bad dab", the first pattern will match the whole string, and the second will only match "a b".

For these reasons, many other utilities and applications have adopted syntaxes that look a lot like Perl's. For example, Java, Ruby, Python, PHP, exim, BBEdit, and even Microsoft's .NET Framework all use regular expression syntax similar to that used in perl. Not all "Perl-compatible" regular expression implementations are identical, and many implement only a subset of Perl's features.

5.1 Examples

**Conventions** used in the examples: The character 'm' is not always required to specify a perl match operation. For example, m/[\^-abc]/ could also be rendered as /[\^-abc]/. The 'm' is only necessary if the user wishes to specify a match operation without using a forward-slash as the regex delimiter. Sometimes it is useful to specify an alternate regex delimiter in order to avoid "delimiter collision". See 'perldoc perlre\(^2\)' for more details.

```plaintext
metacharacter(s) ;; the metacharacters column specifies the regex syntax being demonstrated
=˜ m// ;; indicates a regex match operation in perl
=˜ s/// ;; indicates a regex substitution operation in perl
```

In the table heading below, "M-c" stands for "Metacharacters".

<table>
<thead>
<tr>
<th>M-c</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| .   | Normally matches any character except a newline. Within square brackets the dot is literal. | if ("Hello World
n" =˜ m/

/.) { print "Yes"; # Has length >= 5
n
}; |

---

2. [http://perldoc.perl.org/perlre.html](http://perldoc.perl.org/perlre.html)
<table>
<thead>
<tr>
<th>M-c</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| ()  | Groups a series of pattern elements to a single element. When you match a pattern within parentheses, you can use any of $1, $2, ... later to refer to the previously matched pattern. | if ("Hello World\n" =~ m/\(H..\).(o..)/) {  
    print "We matched '$1' and '$2'\n";
}  
Output: 'Hel' and 'o W'; |
| +   | Matches the preceding pattern element one or more times. | if ("Hello World\n" =~ m/l+/) {  
    print "One or more \"l\"'s in the string\n";
} |
| ?   | Matches the preceding pattern element zero or one times. | if ("Hello World\n" =~ m/H.?e/) {  
    print "There is an 'H' and a 'e' separated by ";
    print "0-1 characters (Ex: He Hoe)\n";
} |
| ?   | Modifies the *,+, or \{M,N\}'d regexp that comes before to match as few times as possible. | if ("Hello World\n" =~ m/\(l.+?o)/) {  
    print "Yep"; # The non-greedy match with 'l' followed
    # by one or more characters is 'llo' rather than 'llo wo'.
} |
| *   | Matches the preceding pattern element zero or more times. | if ("Hello World\n" =~ m/el*o/) {  
    print "There is an 'e' followed by zero to many ";
    print "'l' followed by 'o' (eo, elo, ello, elllo)\n";
} |
| \{M,N\} | Denotes the minimum M and the maximum N match count. | if ("Hello World\n" =~ m/l\{1,2\}/) {  
    print "There is a substring with at least 1 ";
    print "and at most 2 l's in the string\n";
} |
| \[\ldots\] | Denotes a set of possible character matches. | if ("Hello World\n" =~ m/\[aeiou\]+/) {  
    print "Yep"; # Contains one or more vowels
} |
| | Separates alternate possibilities. | if ("Hello World\n" =~ m/\(Hello|Hi|Pogo)/) {  
    print "At least one of Hello, Hi, or Pogo is ";
    print "contained in the string.\n";
} |
| \b | Matches a word boundary. | if ("Hello World\n" =~ m/llo\b/) {  
    print "There is a word that ends with 'llo'\n";
} |
| \w | Matches an alphanumeric character, including \“\_\”. | if ("Hello World\n" =~ m/\w/) {  
    print "There is at least one alphanumeric ";
    print "character in the string (A-Z, a-z, 0-9, _)\n";
} |
| \W | Matches a non-alphanumeric character, excluding \“\_\”. | if ("Hello World\n" =~ m/\W/) {  
    print "The space between Hello and ";
    print "World is not alphanumeric\n";
} |
<table>
<thead>
<tr>
<th>M-c</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| \s  | Matches a whitespace character (space, tab, newline, form feed) | if ("Hello World\n" =~ m/\s+/\n/) {
  print "All the if statements return a TRUE value.";
} |
| \S  | Matches anything BUT a whitespace.                               | if ("Hello World\n" =~ m/\S/\n/) {
  print "Matchesawhitespacechar-
  acter (space, tab, newline,
  form feed)";
} |
| \d  | Matches a digit, same as [0-9].                                 | if ("99 bottles of beer on the wall." =~ m/\d+/) {
  print "$1 is the first number in the string'\n";
} |
| \D  | Matches a non-digit.                                            | if ("Hello World\n" =~ m/\D/) {
  print "There is at least one character in the string";
} |
| \b  | Matches the beginning of a line or string.                      | if ("Hello World\n" =~ m/\b/) {
  print "Starts with the characters 'He'\n";
} |
| $   | Matches the end of a line or string.                            | if ("Hello World\n" =~ m/\b/) {
  print "Is a line or string ";
} |
| \A  | Matches the beginning of a string (but not an internal line).  | if ("Hello\nWorld\n" =~ m/\A/) {
  print "\n";
} |
| \Z  | Matches the end of a string (but not an internal line).         | if ("Hello\nWorld\n" =~ m/\Z/) {
  print "\n";
} |
| \[^\] | Matches every character except the ones inside brackets.       | if ("Hello World\n" =~ m/[^abc]/) {
  print "\n";
} |

5.2 Use in Tools

Tools and languages that utilize Perl regular expression syntax include:

- Java³
- leafnode
- Perl⁴

Perl-Compatible Regular Expressions

- Python\(^5\)
- PHP\(^6\)

### 5.3 See also

- Perl Programming/Regular Expressions Reference\(^7\)

### 5.4 Links

- Perl regular expressions\(^8\) at perl.org
- Perl Compatible Regular Expressions library\(^9\) at pcre.org
- Perl Regular Expression Syntax\(^10\) at boost.org
- W:Regular_expressions/#Standard_Perd\(^11\)

---

8  [http://perldoc.perl.org/perlre.html](http://perldoc.perl.org/perlre.html)
10 [http://www.boost.org/doc/libs/1_34_1/libs/regex/doc/syntax_perl.html](http://www.boost.org/doc/libs/1_34_1/libs/regex/doc/syntax_perl.html)
6 POSIX Basic Regular Expressions

The POSIX Basic Regular Expression (BRE) syntax provided extensions to achieve consistency between utility programs such as grep, sed and awk. These extensions are not supported by some traditional implementations of Unix tools.

6.0.1 History

Traditional Unix regular expression syntax followed common conventions that often differed from tool to tool. The POSIX Basic Regular Expressions syntax was developed by the IEEE, together with an extended variant called Extended Regular Expression1 syntax. These standards were designed mostly to provide backward compatibility with the traditional ../Simple Regular Expressions/2 syntax, providing a common standard which has since been adopted as the default syntax of many Unix regular expression tools.

6.0.2 Syntax

In POSIX Basic Regular Expression syntax, most characters are treated as literals — they match only themselves (e.g., a matches "a"). The exceptions, listed below, are called metacharacters or metasequences.

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>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 &quot;abc&quot;, etc., but [a.c] matches only &quot;a&quot;, &quot;,&quot;, or &quot;c&quot;.</td>
</tr>
</tbody>
</table>

1 http://en.wikibooks.org/wiki/ERE
2 Chapter 2.0.7 on page 8
<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>A bracket expression. Matches a single character that is contained within the brackets. For example, <code>[abc]</code> matches &quot;a&quot;, &quot;b&quot;, or &quot;c&quot;, and <code>[a-z]</code> specifies a range which matches any lowercase letter from &quot;a&quot; to &quot;z&quot;. These forms can be mixed: <code>[abcx-z]</code> matches &quot;a&quot;, &quot;b&quot;, &quot;c&quot;, &quot;x&quot;, &quot;y&quot;, or &quot;z&quot;, as does <code>[a-cx-z]</code>. The - character is treated as a literal character if it is the last or the first character within the brackets: <code>[abc-]</code>, <code>[-abc]</code>. The ] character can be included in a bracket expression if it is the first character: <code>[abc]</code>. The bracket expression may also contain character classes(^3), equivalence classes(^4), and collating characters(^5).</td>
</tr>
<tr>
<td>[^ ]</td>
<td>Matches a single character that is not contained within the brackets. For example, <code>[^abc]</code> matches any character other than &quot;a&quot;, &quot;b&quot;, or &quot;c&quot;, and <code>[^a-z]</code> matches any single character that is not a lowercase letter from &quot;a&quot; to &quot;z&quot;. These forms can be mixed: <code>[^abcx-z]</code> matches any character other than &quot;a&quot;, &quot;b&quot;, &quot;c&quot;, &quot;x&quot;, &quot;y&quot;, or &quot;z&quot;. The - character is treated as a literal character if it is the last character or the first characted after <code>^</code>: <code>[^abc-]</code>, <code>[^abc]</code> The ] character is treated as a literal character if it is the first character after <code>^</code>: <code>[^]abc</code>. The expression may also contain character classes(^6), equivalence classes(^7), and collating characters(^8).</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Matches the starting position within the string, if it is the first character of the regular expression.</td>
</tr>
<tr>
<td><code>$</code></td>
<td>Matches the ending position of the string, if it is the last character of the regular expression.</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Matches the preceding element zero or more times. For example, <code>ab*c</code> matches &quot;ac&quot;, &quot;abc&quot;, &quot;abbbc&quot;, etc. <code>[xyz]*</code> matches &quot;&quot;, &quot;x&quot;, &quot;y&quot;, &quot;z&quot;, &quot;zx&quot;, &quot;zyx&quot;, &quot;zyxzy&quot;, and so on.</td>
</tr>
<tr>
<td>BRE: <code>{m \}</code></td>
<td>ERE: <code>{m }</code> Matches the preceding element exactly <code>m</code> times. For example, <code>a\{3\}</code> matches only &quot;aaa&quot;.</td>
</tr>
<tr>
<td>BRE: <code>{m , \}</code></td>
<td>ERE: <code>{m , }</code> Matches the preceding element at least <code>m</code> times. For example, <code>a\{3,\}</code> matches &quot;aaa&quot;, &quot;aaa&quot;, &quot;aaaa&quot;, &quot;aaaaa&quot;, &quot;aaaaaaa&quot;, and so on.</td>
</tr>
<tr>
<td>BRE: <code>{m , n \}</code></td>
<td>ERE: <code>{m , n }</code> Matches the preceding element at least <code>m</code> and not more than <code>n</code> times. For example, <code>a\{3,5\}</code> matches only &quot;aaa&quot;, &quot;aaa&quot;, and &quot;aaaaaa&quot;. This is not found in a few older instances of regular expressions.</td>
</tr>
<tr>
<td>Metacharacter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>BRE: ( )</td>
<td>Defines a subexpression. It is treated as a single element. For example, ( ab^* ) matches &quot;a&quot;, &quot;ab&quot;, &quot;abb&quot; and so on, while ( (ab)^* ) matches &quot;, &quot;ab&quot;, &quot;abab&quot;, &quot;ababab&quot;, and so on. The string matched within the parentheses can be recalled later (see the next entry, ( n )). A subexpression is also called a marked subexpression, a block or a capturing group.</td>
</tr>
<tr>
<td>ERE: ( )</td>
<td>Matches what the ( n ) th marked subexpression matched, where ( n ) is a digit from 1 to 9. This construct is theoretically irregular (an expression with this construct does not obey the mathematical definition of regular expression), and was not adopted in the POSIX ERE syntax.</td>
</tr>
</tbody>
</table>

Examples:
- \.at matches any three-character string ending with "at", including "hat", "cat", and "bat".
- \[hc\]at matches "hat" and "cat".
- \[^bl]at matches all strings matched by \.at except "bat".
- "\[hc\]at matches "hat" and "cat", but only at the beginning of the string or line.
- \[hc\]at$ matches "hat" and "cat", but only at the end of the string or line.
- \[::-]\ matches any single character surrounded by "]" and "[" since the brackets are escaped, for example: "]a/" and "]b/".

### 6.0.3 Character classes

The POSIX standard defines some classes or categories of characters as shown below. These classes are used within brackets.

<table>
<thead>
<tr>
<th>POSIX class</th>
<th>similar to</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:upper:]</td>
<td>[A-Z]</td>
<td>uppercase letters</td>
</tr>
<tr>
<td>[:lower:]</td>
<td>[a-z]</td>
<td>lowercase letters</td>
</tr>
<tr>
<td>[:alpha:]</td>
<td>[A-Za-z]</td>
<td>upper- and lowercase letters</td>
</tr>
<tr>
<td>[:digit:]</td>
<td>[0-9]</td>
<td>digits</td>
</tr>
<tr>
<td>[:xdigit:]</td>
<td>[0-9A-Fa-f]</td>
<td>hexadecimal digits</td>
</tr>
<tr>
<td>[:alnum:]</td>
<td>[A-Za-z0-9]</td>
<td>digits, upper- and lowercase letters</td>
</tr>
<tr>
<td>[:punct:]</td>
<td>[\ [:cntrl:]]</td>
<td>punctuation (all graphic characters except letters and digits)</td>
</tr>
<tr>
<td>[:blank:]</td>
<td>[\t]</td>
<td>space and TAB characters only</td>
</tr>
<tr>
<td>[:space:]</td>
<td>[\t\n\r\f\v]</td>
<td>blank (whitespace) characters</td>
</tr>
<tr>
<td>[:cntrl:]</td>
<td>\ [:cntrl:]</td>
<td>control characters</td>
</tr>
<tr>
<td>[:graph:]</td>
<td>[\ [:cntrl:]]</td>
<td>graphic characters (all characters which have graphic representation)</td>
</tr>
<tr>
<td>[:print:]</td>
<td>[[:graph] ]</td>
<td>graphic characters and space</td>
</tr>
</tbody>
</table>
For example,

- \(a[[:digit:]]b\) matches "a0b", "a1b", ..., "a9b".
- \(a[:digit:]b\) is an error: character classes must be in brackets.
- \([[:digit:]:abc]\) matches any digit, "a", "b", and "c".
- \(abc[:digit:]\) is the same as above.
- \([^ABZ[:lower:]]\) matches any character except lowercase letters, A, B, and Z.

### 6.0.4 Collating symbols

Collating symbols, like character classes, are used in brackets and have the form \([.ch.]\). Here \(ch\) is a digraph. Collating systems are defined by the locale.

### 6.0.5 Equivalence classes

Equivalence classes, like character classes and collating symbols, are used in brackets and have the form \([=a=]\). They stand for any character which is equivalent to the given. According to the standard\[http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html#tag_09_03_03,\]

For example, if 'a', 'à', and 'â' belong to the same equivalence class, then "[=a=]b", "[=à=]b", and "[=â=]b" are each equivalent to "[aa\â]b".

Equivalence classes, like collating symbols, are defined by the locale.

### 6.1 External links


### 6.2 Use in Tools

Tools and languages that utilize this regular expression syntax include:

- W:TextPad\[http://en.wikipedia.org/wiki/TextPad\]

### 6.3 Links

- POSIX Basic Regular Expressions\[http://www.regular-expressions.info/posix.html\] at regular-expressions.info
- POSIX Basic Regular Expression Syntax\[http://www.boost.org/doc/libs/1_44_0/libs/regex/doc/html/boost_regex/syntax/basic_syntax.html\] at boost.org

---

9  http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html
10 http://en.wikipedia.org/wiki/TextPad
11 http://www.regular-expressions.info/posix.html
12 http://www.boost.org/doc/libs/1_44_0/libs/regex/doc/html/boost_regex/syntax/basic_syntax.html
7 POSIX-Extended Regular Expressions

The more modern "extended" regular expressions can often be used with modern Unix utilities by including the command line flag "-E".

POSIX extended regular expressions are similar in syntax to the traditional Unix regular expressions, with some exceptions. The following metacharacters are added:

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>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 &quot;abc&quot;, etc., but [a.c] matches only &quot;a&quot;, &quot;,&quot;, or &quot;c&quot;.</td>
</tr>
<tr>
<td>[ ]</td>
<td>A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches &quot;a&quot;, &quot;b&quot;, or &quot;c&quot;. [a-z] specifies a range which matches any lowercase letter from &quot;a&quot; to &quot;z&quot;. These forms can be mixed: [abcx-z] matches &quot;a&quot;, &quot;b&quot;, &quot;c&quot;, &quot;x&quot;, &quot;y&quot;, or &quot;z&quot;, 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: [a]bc.</td>
</tr>
<tr>
<td>^</td>
<td>Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than &quot;a&quot;, &quot;b&quot;, or &quot;c&quot;. [^a-z] matches any single character that is not a lowercase letter from &quot;a&quot; to &quot;z&quot;. As above, literal characters and ranges can be mixed.</td>
</tr>
<tr>
<td>^</td>
<td>Matches the starting position within the string. In line-based tools, it matches the starting position of any line.</td>
</tr>
<tr>
<td>$</td>
<td>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.</td>
</tr>
<tr>
<td>BRE: \</td>
<td>Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, \n). A marked subexpression is also called a block or capturing group.</td>
</tr>
<tr>
<td>ERE: ( )</td>
<td></td>
</tr>
</tbody>
</table>
Metacharacter | Description
--- | ---
\n | Matches what the \(n\)th marked subexpression matched, where \(n\) is a digit from 1 to 9. This construct is theoretically **irregular** and was not adopted in the POSIX ERE syntax. Some tools allow referencing more than nine capturing groups.

* | 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.

BRE: \{m , n \} ERE: \{m , n \} | Matches the preceding element at least \(m\) and not more than \(n\) times. For example, \(a\{3,5\}\) matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regular expressions.

• + — Match the last "block" one or more times - "ba+" matches "ba", "baa", "baaa" and so on
• ? — Match the last "block" zero or one times - "ba?" matches "b" or "ba" • | — The choice (or set union) operator: match either the expression before or the expression after the operator - "abc|def" matches "abc" or "def".

Also, backslashes are removed: \{ {... } \} becomes \{ ... \} and \( \{ ... \} \) becomes \(...\). Examples:

• "[hc]+at" matches with "hat", "cat", "hhat", "chat", "hcat", "ccchat" etc.
• "[hc]?at" matches "hat", "cat" and "at"
• "([cC]at)|([dD]og)" matches "cat", "Cat", "dog" and "Dog"　

The characters (,),[,.,*?,+,\,], and $ are special symbols and have to be escaped with a backslash symbol in order to be treated as literal characters. For example:

"a\.((\(|\))\)" matches with the string "a.)" or "a.(")

Modern regular expression tools allow a quantifier to be specified as non-greedy, by putting a question mark after the quantifier: \(\{\{.*?\}\}\).

### 7.0.1 Character classes

The POSIX standard defines some classes or categories of characters as shown in the following table:

<table>
<thead>
<tr>
<th>POSIX class</th>
<th>similar to</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:upper:]</td>
<td>[A-Z]</td>
<td>uppercase letters</td>
</tr>
<tr>
<td>[:lower:]</td>
<td>[a-z]</td>
<td>lowercase letters</td>
</tr>
<tr>
<td>[:alpha:]</td>
<td>[A-Za-z]</td>
<td>upper- and lowercase letters</td>
</tr>
<tr>
<td>[:alnum:]</td>
<td>[A-Za-z0-9]</td>
<td>digits, upper- and lowercase letters</td>
</tr>
<tr>
<td>[:digit:]</td>
<td>[0-9]</td>
<td>digits</td>
</tr>
<tr>
<td>[:xdigit:]</td>
<td>[0-9A-Fa-f]</td>
<td>hexadecimal digits</td>
</tr>
<tr>
<td>[:punct:]</td>
<td>[!,?,...]</td>
<td>punctuation</td>
</tr>
<tr>
<td>[:blank:]</td>
<td>[\t\n\r\f\v]</td>
<td>space and TAB characters only</td>
</tr>
<tr>
<td>[:space:]</td>
<td></td>
<td>blank (whitespace) characters</td>
</tr>
</tbody>
</table>
### Use in Tools

Tools and languages that utilize this regular expression syntax include:

- **AWK**
  - uses a superset of the extended regular expression syntax

### Links

- [Regular_expression#Character_classes](http://en.wikipedia.org/wiki/Regular_expression%23Character_classes)
- [Character Clasess that are Always Supported](http://www.boost.org/doc/libs/1_44_0/libs/regex/doc/html/boost_regex/syntax/character_classes/std_char_classes.html)
- [POSIX Basic Regular Expressions](http://www.regular-expressions.info/posix.html)
- [POSIX Extended Regular Expression Syntax](http://www.boost.org/doc/libs/1_44_0/libs/regex/doc/html/boost_regex/syntax/basic_extended.html)
8 Non-POSIX Basic Regular Expressions

Non POSIX Basic Regular Expression Syntax: An additional non-POSIX class understood by some tools is [:word:], which is usually defined as [:alnum:] plus underscore. This form of regular expression is used to reflect the fact that in many programming languages these characters may be used in identifiers.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>The dot operator matches any single character.</td>
</tr>
<tr>
<td>[]</td>
<td>Boxes enable a single character to be matched against a character lists or character range.</td>
</tr>
<tr>
<td>[ˆ]</td>
<td>A compliment box enables a single character not within in a character list or character range to be matched.</td>
</tr>
<tr>
<td>*</td>
<td>An asterisk specifies zero or more characters to match.</td>
</tr>
<tr>
<td>^</td>
<td>The caret anchor matches the beginning of the line</td>
</tr>
<tr>
<td>$</td>
<td>The dollar anchor matches the end of the line</td>
</tr>
</tbody>
</table>

The editor vim further distinguishes word and word-head classes (using the notation \w and \h) since in many programming languages the characters that can begin an identifier are not the same as those that can occur in other positions.

(For an ASCII chart color-coded to show the POSIX classes, see ASCII\(^1\).)

8.1 Use in Tools

Tools and languages that utilize this regular expression syntax include:

- vim\(^2\)

---

\(^1\) [http://billposer.org/Linguistics/Computation/ascii.html](http://billposer.org/Linguistics/Computation/ascii.html)

9 Emacs Regular Expressions

Notes on regular expressions used in text editor Emacs:

- For backslash escaping (magic vs literal), Emacs uses a mixture of BRE and ERE. Like in ERE, Emacs supports unescaped +, ?. Like in BRE, Emacs supports escaped \(, \), \{|, \}.
- GNU extensions to regular expressions supported by Emacs include \w, \W, \b, \B, \<, \>, \', \' (start and end of buffer)
- No "\s" like in PCRE; whitespace is matched by "\s-".
- No "\d" like in PCRE; use [0-9] or [[:digit:]]
- No lookahead and no lookbehind like in PCRE
- Emacs regexp can match characters by syntax using mode-specific syntax tables ("\sc", "\sc", "\s") or by categories ("\ce", "\cg").

9.1 Use in Tools

Tools and languages that utilize this regular expression syntax include:

- Emacs\(^1\)

9.2 Links

- Regular Expressions\(^2\) at emacswiki.org
- Perl and Emacs regular expressions compared\(^3\) at lemoda.net

---

10 Shell Regular Expressions

The Unix shell recognises a limited form of regular expressions used with filename substitution:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>The hook operator specifies any single character.</td>
</tr>
<tr>
<td>[]</td>
<td>boxes enable a single character to be matched against a character lists or character range.</td>
</tr>
<tr>
<td>[!]</td>
<td>A compliment box enables a single character not within in a character list or character range to be matched.</td>
</tr>
<tr>
<td>*</td>
<td>An asterisk specifies zero or more characters to match.</td>
</tr>
</tbody>
</table>

Some operators behave differently in the shell: The asterisk and hook operators do not not need to follow a previous character in the shell and they exhibit non traditional regular expression behaviour.

Unsupported Constructs: Within the shell, a compliment box is formed using the pling symbol. The shell does not support the use of a cared box for character list exclusion. In the shell, a caret symbol within a box will simply be treated as one of the characters within the character list for matching.

10.1 Use in Tools

Tools and languages that utilize this regular expression syntax include:

- Bourne compatible shells
11 Implementation

11.0.1 Implementations and running times

There are at least 3 different algorithms that decide if (and how) a given string matches a regular expression. They are based on different representations of the regular expression as a Finite Automation and on the amount of functionality present in the matcher.

1. An NFA based matcher without back-references and look ahead/behind. An input of size $O(n)$ can be tested against a regular expression of size $O(m)$ in time $O(nm)$, and additional $O(m)$ extra space by simulating an NFA using Thompson’s algorithm\(^1\). If $c$ sub-match capture groups are to be recorded, then the running time increases to $O(nm \log c)$, but the space requirement remains $O(m)$\(^2\).

2. An NFA based matcher with back-references and look ahead/behind. Such a matcher needs to be implemented using backtracking. An input of size $O(n)$ can be tested against a regular expression of size $O(m)$ in time $O(2^m n)$ using backtracking. Some effort is needed to ensure that the backtracking based matcher doesn’t enter an infinite loop, testing the same path over and over again.

3. A DFA based matcher. DFA based matchers can’t support back-references, sub-match captures, or look ahead/behind. This is the oldest and fastest kind of matcher and relies on a result in formal language theory that allows every nondeterministic Finite State Machine (NFA) to be transformed into a deterministic finite state machine (DFA). The algorithm performs or simulates this transformation and then runs the resulting DFA on the input string, one symbol at a time. The latter process (DFA matching) takes time that is proportional to the length of the input string. More precisely, a regular expression of size $m$ on an input alphabet of size $S$ can be converted into a DFA in time $O(2^m S)$, and subsequently an input string of size $n$ can be tested against a DFA of any size in time $O(n)$.

The DFA based algorithm is fast to match input against a regular expression, but can be used only for matching and not for recalling grouped subexpressions. There is a variant that can recall grouped subexpressions, but its running time slows down to $O(n^2m)$.

The running time of the backtracking based algorithm can be exponential, which simple implementations exhibit when matching against expressions like “(a|aa)*b” that contain both alternation and unbounded quantification and force the algorithm to consider an exponential number of subcases. More complex implementations identify and speed up various common cases where they would otherwise run slowly.

---

1. [http://swtch.com/~rsc/regexp/regexp1.html](http://swtch.com/~rsc/regexp/regexp1.html)
2. [https://github.com/dhruvbird/regexp-js/blob/master/complexity.md#capturing-sub-matches](https://github.com/dhruvbird/regexp-js/blob/master/complexity.md#capturing-sub-matches)
Even though backtracking implementations only give an exponential guarantee in the worst case, they allow much greater flexibility and provide more expressive power. For instance any implementation that allows the use of backreferences, or implements the various improvements that Perl introduced, must use a backtracking implementation.

Some implementations try to provide the best of both algorithms by first running a fast DFA match to see if the string matches the regular expression at all, and only in that case perform a potentially slower backtracking match.

Category:Regular Expressions

3  http://en.wikibooks.org/wiki/Category%3ARegular%20Expressions
12 Glossary

This is a glossary of the book.
\A
In some flavors, the beginning of a string but not of a line in the string
\b
In some flavors, a word boundary
\B
In some flavors, a complement to \b
BRE
Basic regular expressions
\d
In some flavors, a digit
\D
In some flavors, a complement to \d
Emacs
A scriptable text editor with support for regular expressions
ERE
Extended regular expressions
GNU
A project to create a free-as-in-freedom operating system, which provides extensions to regular expressions used in tools such as Grep or Sed
Greedy
Of an operator, matching as much as it can
Grep
A command-line tool for finding lines in a text files that match a regular expression
Java
A byte-compiled programming language with support for regular expressions in its standard library since version 1.4
JavaScript
A scripting language for the web supported by web browsers, with built-in support for regular expressions

Metacharacter
A character or sequence of characters with a special meaning, such as "." or "\+".

PCRE
Perl compatible regular expressions

Perl
An interpreted scripting language noted for its regular expressions

PHP
An interpreted scripting language with support for regular expressions

Regrex
A regular expression

Regular expression
A string containing special characters indicating patterns, intended to match literal strings

\s
In some flavors, a whitespace character: space, tab, newline, form feed

\s-
In Emacs, a whitespace character

\S
In some flavors, a complement to \s

Sed
A non-interactive editor or command-line tool noted for its "s" command substituting strings that match a regular expression with other strings

\u13F
In some flavors, the character with the hexadecimal Unicode value of 13F.

Vim
A scriptable text editor with support for regular expressions

\w
In some flavors, an alphanumeric character, including "_"

\W
In some flavors, a complement to \w

\xF7
In some flavors, the character with the hexadecimal ASCII value of F7.
\x{13F}
In some flavors, the character with the hexadecimal Unicode value of 13F.
\Z
In some flavors, the end of a string but not of a line in the string
\<
In some flavors, an empty string before the beginning of a word
\>
In some flavors, an empty string after the end of a word
^
The beginning of a line
$
The end of a line
.
Any single character, but possibly not a newline
[
The opening of a character class
]
The closing of a character class
(
In some flavors, the opening of a group
)
In some flavors, the closing of a group
\(
In some flavors, the opening of a group
\)
In some flavors, the closing of a group
{
In some flavors, the opening of a match-count iterator
}
In some flavors, the closing of a match-count iterator
\{
In some flavors, the opening of a match-count iterator
\}
In some flavors, the closing of a match-count iterator
|
In some flavors, a marking of an alternative
\|
In some flavors, a marking of an alternative
\1
In some flavors, a backreference to the 1st group
\2
In some flavors, a backreference to the 2nd group
*
Any number of the previous
+
In some flavors, one or more of the previous
\+
In some flavors, one or more of the previous
?
In some flavors, one or none of the previous
\?
In some flavors, one or none of the previous
*?
In some flavors, a non-greedy version of *
+?
In some flavors, a non-greedy version of +
)?
In some flavors, a non-greedy version of }
13 Links

13.1 See also

- JavaScript/Regular Expressions
- Perl Programming/Regular Expressions Reference
- Regular Expressions using R

13.2 External Links

- regular-expressions.info - a regex tutorial site
- Text and Data Manipulation with Regular Expressions in .NET Development — tutorial, reference and Ready to use Regular Expression patterns for VB.NET, C#.NET and ASP.NET development
- Emacs regular expressions at cs.utah.edu
- W:Regular expression
- W:Comparison of regular expression engines
- W:List of regular expression software

Category:Regular Expressions
Category:Regular Expressions

---

1 http://en.wikibooks.org/wiki/JavaScript%2FRegular%20Expressions
2 http://en.wikibooks.org/wiki/Perl%20Programming%2FRegular%20Expressions%20Reference
4 http://www.regular-expressions.info/
6 http://www.cs.utah.edu/dept/old/texinfo/emacs18/emacs_17.html
7 http://en.wikipedia.org/wiki/Regular%20expression
8 http://en.wikipedia.org/wiki/Comparison%20of%20regular%20expression%20engines
9 http://en.wikipedia.org/wiki/List%20of%20regular%20expression%20software
10 http://en.wikibooks.org/wiki/Category%3ARegular%20Expressions
11 http://en.wikibooks.org/wiki/Category%3ARegular%20Expressions
14 Contributors

<table>
<thead>
<tr>
<th>Edits</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Adrignola</td>
</tr>
<tr>
<td>1</td>
<td>Albmont</td>
</tr>
<tr>
<td>85</td>
<td>Dan Polansky</td>
</tr>
<tr>
<td>1</td>
<td>Derbeth</td>
</tr>
<tr>
<td>1</td>
<td>Dirk Hünniger</td>
</tr>
<tr>
<td>9</td>
<td>Franl</td>
</tr>
<tr>
<td>1</td>
<td>Jguk</td>
</tr>
<tr>
<td>3</td>
<td>LoStrangolatore</td>
</tr>
<tr>
<td>1</td>
<td>MOrpheu$</td>
</tr>
<tr>
<td>66</td>
<td>Markhobley</td>
</tr>
<tr>
<td>1</td>
<td>Novwik</td>
</tr>
<tr>
<td>1</td>
<td>PAC2</td>
</tr>
<tr>
<td>2</td>
<td>Panic2k4</td>
</tr>
<tr>
<td>1</td>
<td>Recent Runes</td>
</tr>
<tr>
<td>6</td>
<td>Sae1962</td>
</tr>
<tr>
<td>1</td>
<td>Van der Hoorn</td>
</tr>
</tbody>
</table>

9  http://en.wikibooks.org/wiki/User:MOrpheu\protect\char"0024\relax{}
List of Figures

- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- CFR: Copyright free use.

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses\textsuperscript{17}. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrowser.

\textsuperscript{17} Chapter 15 on page 45
The GNU General Public License (GPL) is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange. It is implemented in the form of a copyright licence attached to the source code of the software.

The GPL is a copyleft licence, which means that it is designed to ensure that users are free to do anything with the software, including running, studying, sharing and changing it. The licence is designed to prevent any restriction on the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.

The "GPL" refers to the GNU General Public License. The GPL is a copyleft licence for free software. It is designed to ensure that users are free to run, study, share and change the software. The GPL is intended to protect the freedom of users, and to provide a mechanism for software interchange.
Licensees are permitted to use, and to distribute copies of this license document, in any medium, provided that they also license and licenseしてください。On a 2001年3月1日, 13x104
46
produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.

produced by some word processors for output purposes only.
The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL. You may convey a Combined Work under sections 1 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions. If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that sees the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy. 3. Object Code Incorporating Material from Library Header Files. The object code form of an Application may incorporate material from a header file that is part of the Library. You must cause the object code, as well as the object code with which the object code is combined, to carry statements of license notice and disclaimers of warranty similar to those of the above.

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. *b) Accompany the Combined Work with a copy of the Corresponding Application Code. *c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. *d) Do one of the following:

i) Convey the object code for the Combined Work under terms of your choice, provided that you also convey the object code for the Library under the terms of this License, in accordance with section 6 of the GNU GPL.

ii) Offer, together with the object code of the Combined Work, a modification of the Corresponding Application Code, if one exists, such that when the Combined Work is combined with the modification, the resulting Combined Work is entirely covered by this License. If the Combined Work is a Library, the Corresponding Application Code must be included in the Combined Work under terms of your choice, provided that you also convey the Library under the terms of this License, in accordance with section 6 of the GNU GPL. If you convey a Library under this License, and the Library as thus received includes the Corresponding Application Code, then you must convey the Corresponding Application Code under terms of your choice as well. If you convey the Combined Work under terms of your choice, you may also convey Corresponding Application Code under terms of your choice, and you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

5. Combined Licenses. You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined Library under terms of your choice, if you do both of the following:

a) Accompany the combined Library with a copy of the same work based on the Library, uncombined with any other library facilities, covered under the terms of this License. *b) Give prominent notice with the combined Library that part of it is a work based on the Library, and conveying both the Library and the combined Library under terms of this License, is permitted provided that the version of the linked Version of the Library that is used in the combined Library is under the terms of this License.

6. Revised Versions of the GNU Lesser General Public License. The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a proxy can decide whether a new version of the Library or any later version published by the Free Software Foundation applies to it, you have the option of following the most recent version decided on by the proxy or of choosing any later version published by the Free Software Foundation. If the Library as you received it specifies that no proxy exists, then anyone may give you a version of the Library published by the Free Software Foundation or a later version published by the Free Software Foundation, at a price no more than the cost of conveying a copy of the version as you receive it. If you convey a covered work, you distribute the corresponding source code in the following cases:

a) Convey the object code for the Combined Work under terms of your choice, provided that you also convey the object code for the Library under the terms of this License, in accordance with section 6 of the GNU GPL.

b) Convey a copy of the object code of the Combined Work under terms of your choice, provided that you also convey the object code for the Library under the terms of this License, in accordance with section 6 of the GNU GPL.

c) Convey a copy of the object code of the Combined Work under terms of your choice, provided that you also convey the object code for the Library under the terms of this License.

d) Convey a copy of the object code of the Combined Work under terms of your choice, provided that you also convey the object code for the Library under the terms of this License.

15.3 GNU Lesser General Public License