Regular Expressions for Perl, C, PHP, Python, Java, and .NET

Regular Expression

Pocket Reference

O'REILLY®

Tony Stubblebine

Perl/Programming

O'REILLY®

Regular Expression Pocket Reference

Regular expressions are a powerful mechanism for matching and manipulating text. Although regular expressions are available in a variety of modern programming languages, each language sports a different set of features, and their syntaxs are sometimes subtly incompatible. Many languages implement regular expressions, but no two languages do it the same.

Regular Expression Pocket Reference contains a quick introduction to regular expression concepts, along with reference sections for the most common and powerful regex implementations. No more puzzling through cryptic or disorganized manual pages! This book contains reference tables and guidelines for using regular expressions in Perl, Java, Python, .NET, C#, PHP, the PCRE library, JavaScript, and various shell tools (*vi*, *awk*, *egrep*, and *sed*).

As a companion to the groundbreaking "definitive" book on the topic, *Mastering Regular Expressions* by Jeffrey Friedl, this small pocket reference will help you sort out the syntax and quirks of regexes in whatever language you choose to program in.

Visit O'Reilly on the Web at *www.oreilly.com*

ISBN 0-596-00415-X
US $9.95
CAN $14.95
90000
9 780596 004156
6 36920 00415 8

Regular Expression
Pocket Reference

Tony Stubblebine

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Regular Expression Pocket Reference
by Tony Stubblebine

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media, Inc. books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor:	Nathan Torkington
Production Editor:	Genevieve d'Entremont
Cover Designer:	Hanna Dyer
Interior Designer:	David Futato

Printing History:
August 2003: First Edition.

0-596-00415-X
[C] [6/06]

Contents

Regular Expression Pocket Reference

Regular expressions (known as regexps or regexes) are a way to describe text through pattern matching. You might want to use regular expressions to validate data, to pull pieces of text out of larger blocks, or to substitute new text for old text.

Regular expression syntax defines a language you use to describe text. Today, regular expressions are included in most programming languages as well as many scripting languages, editors, applications, databases, and command-line tools. This book aims to give quick access to the syntax and pattern-matching operations of the most popular of these languages.

About This Book

This book starts with a general introduction to regular expressions. The first section of this book describes and defines the constructs used in regular expressions and establishes the common principles of pattern matching. The remaining sections of the book are devoted to the syntax, features, and usage of regular expressions in various implementations.

The implementations covered in this book are Perl, Java, .NET and C#, Python, PCRE, PHP, the *vi* editor, JavaScript, and shell tools.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Used for emphasis, new terms, program names, and URLs

`Constant width`
: Used for options, values, code fragments, and any text that should be typed literally

`Constant width italic`
: Used for text that should be replaced with user-supplied values

Acknowledgments

The world of regular expressions is complex and filled with nuance. Jeffrey Friedl has written the definitive work on the subject, *Mastering Regular Expressions* (O'Reilly), a work on which I relied heavily when writing this book. As a convenience, this book provides page references to *Mastering Regular Expressions*, Second Edition (MRE) for expanded discussion of regular expression syntax and concepts.

This book simply would not have been written if Jeffrey Friedl had not blazed a trail ahead of me. Additionally, I owe him many thanks for allowing me to reuse the structure of his book and for his suggestions on improving this book. Nat Torkington's early guidance raised the bar for this book. Philip Hazel, Ron Hitchens, A.M. Kuchling, and Brad Merrill reviewed individual chapters. Linda Mui saved my sanity and this book. Tim Allwine's constant regex questions helped me solidify my knowledge of this topic. Thanks to Schuyler Erle and David Lents for letting me bounce ideas off of them. Lastly, many thanks to Sarah Burcham for her contributions to the "Shell Tools" sections and for providing the inspiration and opportunity to work and write for O'Reilly.

Introduction to Regexes and Pattern Matching

A *regular expression* is a string containing a combination of normal characters and special metacharacters or metasequences. The normal characters match themselves. *Metacharacters* and *metasequences* are characters or sequences of characters that represent ideas such as quantity, locations, or types of characters. The list in the section "Regex Metacharacters, Modes, and Constructs" shows the most common metacharacters and metasequences in the regular expression world. Later sections list the availability of and syntax for supported metacharacters for particular implementations of regular expressions.

Pattern matching consists of finding a section of text that is described (matched) by a regular expression. The underlying code that searchs the text is the *regular expression engine*. You can guess the results of most matches by keeping two rules in mind:

1. *The earliest (leftmost) match wins*
   Regular expressions are applied to the input starting at the first character and proceeding toward the last. As soon as the regular expression engine finds a match, it returns. (See MRE 148-149, 177–179.)

2. *Standard quantifiers are greedy*
   Quantifiers specify how many times something can be repeated. The standard quantifiers attempt to match as many times as possible. They settle for less than the maximum only if this is necessary for the success of the match. The process of giving up characters and trying less-greedy matches is called backtracking. (See MRE 151–153.)

Regular expression engines have subtle differences based on their type. There are two classes of engines: Deterministic Finite Automaton (DFA) and Nondeterministic Finite Automaton (NFA). DFAs are faster but lack many of the features of

an NFA, such as capturing, lookaround, and non-greedy quantifiers. In the NFA world there are two types: Traditional and POSIX.

DFA engines
: DFAs compare each character of the input string to the regular expression, keeping track of all matches in progress. Since each character is examined at most once, the DFA engine is the fastest. One additional rule to remember with DFAs is that the alternation metasequence is greedy. When more than one option in an alternation (`foo|foobar`) matches, the longest one is selected. So, rule #1 can be amended to read "the longest leftmost match wins." (See MRE 155–156.)

Traditional NFA engines
: Traditional NFA engines compare each element of the regex to the input string, keeping track of positions where it chose between two options in the regex. If an option fails, the engine backtracks to the most recently saved position. For standard quantifiers, the engine chooses the greedy option of matching more text; however, if that option leads to the failure of the match, the engine returns to a saved position and tries a less greedy path. The traditional NFA engine uses ordered alternation, where each option in the alternation is tried sequentially. A longer match may be ignored if an earlier option leads to a successful match. So, rule #1 can be amended to read "the first leftmost match after greedy quantifiers have had their fill." (See MRE 153–154.)

POSIX NFA engines
: POSIX NFA Engines work similarly to Traditional NFAs with one exception: a POSIX engine always picks the longest of the leftmost matches. For example, the alternation `cat|category` would match the full word "category" whenever possible, even if the first alternative ("cat") matched and appeared earlier in the alternation. (See MRE 153–154.)