GREP cheat sheet

characters — what to seek

ring matches **ring**, sp**ring**board, **ring**tone, etc.

. matches almost any character

h.o matches **hoo**, **h2o**, **h/o**, etc.

Use **** to search for these special characters:

[\ ^ $. | ? * + ( ) { }

ring**\?** matches **ring?**

\(quiet**\)** matches **(quiet)**

c:****windows matches **c:\windows**

alternatives — | (OR)

cat|dog match **cat** or **dog**

order matters if short alternative is part of longer

id|identity matches **id** or **id**entity

regex engine is "eager", stops comparing

as soon as 1st alternative matches

identity|id matches **id** or **identity**

order longer to shorter when alternatives overlap

(To match whole words, see scope and groups.)

character classes — [allowed] or [^NOT allowed]

[aeiou] match any vowel

[^aeiou] match a NON vowel

r**[iau]**ng match **ring**, w**rang**le, sp**rung**, etc.

gr**[ae]**y match **gray** or **grey**

[a-zA-Z0-9] match any letter or digit

(In **[ ]** always escape **. \ ]** and sometimes **^ -** .)

shorthand classes

\w "word" character (letter, digit, or underscore)

\d digit

\s whitespace (space, tab, vtab, newline)

\W, **\D**, or **\S**, (NOT word, digit, or whitespace)

[\D\S] means not digit OR whitespace, both match

[^\d\s] disallow digit AND whitespace

occurrences — ? * + {n} {n,} {n,n}

? 0 or 1

colou?r match **color** or **colour**

***** 0 or more

[BW]ill**[ieamy's]*** match **Bill**, **Willy**, **William's** etc.

+ 1 or more

[a-zA-Z]+ match 1 or more letters

{n} require n occurrences

\d{3}-**\d{2}**-**\d{4}** match a SSN

{n,} require n or more

[a-zA-Z]{2,} 2 or more letters

{n,m} require n - m

[a-z]\w{1,7} match a UW NetID

scope — \b \B ^ $

\b "word" edge (next to non "word" character)

\bring word starts with "ring", ex **ringtone**

ring**\b** word ends with "ring", ex **spring**

\b9\b match single digit **9**, not 19, 91, 99, etc..

\b[a-zA-Z]{6}**\b** match 6-letter words

\B NOT word edge

\Bring**\B** match **springs** and **wringer**

^ start of string **$** end of string

^\d***$** entire string must be digits

^[a-zA-Z]{4,20}**$** string must have 4-20 letters

^[A-Z] string must begin with capital letter

[\.!?"')]**$** string must end with terminal puncutation

groups — ( )

(in|out)put match **input** or **output**

\d{5}**(-\d{4})?** US zip code ("+ 4" optional)

Locate all PHP input variables:

\$_**(GET|POST|REQUEST|COOKIE|SESSION|SERVER)**\

[.+\]

NB: parser tries EACH alternative if match fails after group.

Can lead to catastrophic backtracking.

back references — \n

each **( )** creates a numbered "back reference"

(to**) (**be**)** or not **\1 \2** match **to be or not to be**

([^\s]**)\1**{2} match non-space, then same twice more **aaa**, **...**

\b**(\w+)**\s+**\1**\b match doubled words

non-capturing group — (?:) prevent back reference

on(?:click|load) is faster than **on(click|load)**

use non-capturing or atomic groups when possible

atomic groups — (?>a|b) (no capture, no backtrack)

(?>red|green|blue)

faster than non-capturing

alternatives parsed left to right without return

(?>id|identity)\b matches **id**, but not **id**entity

"id" matches, but "\b" fails after atomic group,

parser doesn't backtrack into group to retry 'identity'

If alternatives overlap, order longer to shorter.

lookahead — (?=) (?!) lookbehind — (?<=) (?<!)

\b\w+?**(?=**ing\b**)** match **warbl**ing, **str**ing, **fish**ing, ...

\b**(?!**\w+ing\b**)** \w+\b words NOT ending in "ing"

(?<=\bpre**)**.*?\b match pre**tend**, pre**sent**, pre**fix**, ...

\b\w{3}**(?<!**pre**)** \w*?\b words NOT starting with "pre"

(lookbehind needs 3 chars, \w{3}, to compare w/"pre")

\b\w+ **(?<!**ing**)** \b match words NOT ending in "ing"

(see LOOKAROUND notes below)

if-then-else — (?ifthen|else)

*** greedy versus *? lazy**

***** **+** and **{n,}** are greedy — match as much as possible

<.+> finds 1 big match in **bold**

***?** **+?** and **{n,}?** are lazy — match as little as possible

<.+?> finds 2 matches in ****bold****

comments — (?#comment)

(?#year)(19|20)\d\d embedded comment

(?x)(19|20)\d\d **#year** free spacing & EOL comment

(see modifiers)

match "Mr." or "Ms." if word "her" is later in string

M**(?(?=**.*?\bher\b**)s|r)**\. lookahead for word "her" (requires lookaround for IF condition)

modifiers — i s m x

ignore case, single-line, multi-line, free spacing

(?i)[a-z]*****(?-i)** ignore case ON / OFF

(?s).*****(?-s)** match multiple lines (causes **.** to match newline)

(?m)^.*;$**(?-m)** **^** & **$** match lines not whole string

(?x) #free-spacing mode, this EOL comment ignored

\d{3} #3 digits (new line but same pattern)

-\d{4} #literal hyphen, then 4 digits

(?-x) (?#free-spacing mode OFF)

/regex/**ismx** modify mode for entire string

A few examples:

capture P content using if-then-else to allow for attributes in opening tag
(?s)<p(?(?=\s)\ .*?)>(.*?)</p>

(?s)<p(?(?=\s)\ .*?)>(.*?)</p> span multiple lines

(?s)**<p**(?(?=\s)\ .*?)>(.*?)</p> locate opening "<p"

(?s)<p**(?**(?=\s)\ .*?**)**>(.*?)</p> create an if-then-else

(?s)<p(?**(?=\s)**\ .*?)>(.*?)</p> lookahead for a whitespace character

(?s)<p(?(?=\s)**\ .*?**)>(.*?)</p> if found, attempt lazy match of any characters until ...

(?s)<p(?(?=\s)\ .*?)**>**(.*?)</p> closing angle brace

(?s)<p(?(?=\s)\ .*?)>**(.*?)**</p> capture lazy match of all characters until ...

(?s)<p(?(?=\s)\ .*?)>(.*?)**</p>** closing "</p>"

The lookahead prevents matches on PRE, PARAM, and PROGRESS tags by only allowing more characters in the opening tag if P is followed by whitespace. Otherwise, ">" must follow "<p".

LOOKAROUND notes

(?=) if you can find ahead

(?!) if you can NOT find ahead

(?<=) if you can find behind

(?<!) if you can NOT find behind

convert Firstname Lastname to Lastname, Firstname (& visa versa)

Pattern below uses lookahead to capture everything up to a space, characters, and a newline.

The 2nd capture group collects the characters between the space and the newline.

This allows for any number of names/initials prior to lastname, provided lastname is at the end of the line.

Find: `(.*)**(?= .*\n)** (.*)\n`

Repl: `\2, \1\n` — insert 2nd capture (lastname) in front of first capture (all preceding names/initials)

Reverse the conversion.

Find: `(.*?), (.*?)\n` — group 1 gets everything up to ", " — group 2 gets everything after ", "

Repl: `\2 \1\n`

lookaround groups are non-capturing

If you need to capture the characters that match the lookaround condition, you can insert a capture group inside the lookaround.

`(?=**(**sometext**)**)` the inner () captures the lookahead

This would NOT work: `**(**(?=sometext)**)**` Because lookaround groups are zero-width, the outer () capture nothing.

lookaround groups are zero-width

They establish a condition for a match, but are not part of it.

Compare these patterns: re?d vs r(?=e)d

r**e?**d — match an "r", an optional "e", then "d" — matches **red** or **rd**

r**(?=e)**d — match "r" (IF FOLLOWED BY "e") then see if "d" comes after "r"

The lookahead seeks "e" only for the sake of matching "r".

Because the lookahead condition is ZERO-width, the expression is logically impossible.

It requires the 2nd character to be both "e" and "d".

For looking ahead, "e" must follow "r".

For matching, "d" must follow "r".

fixed-width lookbehind

Most regex engines depend on knowing the width of lookbehind patterns. Ex: **(?<=h1)** or **(?<=\w{4})** look behind for "h1" or for 4 "word" characters.

This limits lookbehind patterns when matching HTML tags, since the width of tag names and their potential attributes can't be known in advance.

variable-width lookbehind

.NET and **JGSoft** support variable-width lookbehind patterns. Ex: **(?<=\w+)** look behind for 1 or more word characters.

The first few examples below rely on this ability.

match text bound by simple HTML tags (NB: <\w+> does not match tags with attributes.)
(?<=<(\w+)>).*(?=</\1>)

Lookaround groups define the context for a match. Here, we're seeking **.*** ie., 0 or more characters.
A positive lookbehind group **(?<= . . .)** preceeds. A positive lookahead group **(?= . . .)** follows.
These set the boundaries of the match this way:

(?<=<(\w+)>**)**.*(?=</\1>) look behind current location

(?<=**<**(\w+)**>**).*(?=</\1>) for < > surrounding ...

(?<=<**(\w+)**>).*(?=</\1>) one or more "word" characters. The ( ) create a capture group to preserve the name of the presumed tag: DIV, H1, P, A, etc.

(?<=<(\w+)>)**.***(?=</\1>) match anything until

(?<=<(\w+)>).***(?=**</\1>**)** looking ahead from the current character

(?<=<(\w+)>).*(?=**</**\1**>**) these characters surround

(?<=<(\w+)>).*(?=</**\1**>) the contents of the first capture group

In other words, advance along string until an opening HTML tag preceeds. Match chars until its closing HTML tag follows.
The tags themselves are not matched, only the text between them.

To span multiple lines, use the **(?s)** modifier. **(?s)**(?<=<cite>).*(?=</cite>) Match <cite> tag contents, regardless of line breaks.

match text bound by HTML tags, including tags with attributes (not nested, though)
(?<=<(\w+?)\ ?.*?>).*(?=</\1>)

As in example above, the first group **(\w+)** captures the presumed tag name, then an optional space and other characters **\ ?.*?** allow for attributes before the closing **>**.

match text bound by HTML tags when a class attribute = "red"
(?<=<(\w+?)\ ?.*?class=".*?\bred\b.*?".*?>).*(?=</\1>)

class=".*?\bred\b.*?" this new part looks for **class="** and **red** and **"** somewhere in the opening tag

\b ensures "red" is a single word

.*? allow for other characters on either side of "red" so pattern matches class="red" and class="blue red green" etc.

match complex opening & closing xhtml tags and all text between
(?i)<**([a-z][a-z0-9]*)[^>]***>**.*?**</**\1**>**

Here, the first group captures only the tag name. The tag's potential attributes are outside the group.

(?i)<([a-z][a-z0-9]*)[^>]*>.*?</\1> set ignore case ON
(?i)<(**[a-z]**[a-z0-9]*)[^>]*>.*?</\1> find an opening tag by matching 1 letter after <
(?i)<([a-z]**[a-z0-9]***)[^>]*>.*?</\1> then match 0 or more letters or digits
(?i)<**(**[a-z][a-z0-9]***)**[^>]*>.*?</\1> make this tag a capture group
(?i)<([a-z][a-z0-9]*)**[^>]***>.*?</\1> match 0 or more characters that aren't > — this allows attributes in opening tag
(?i)<([a-z][a-z0-9]*)[^>]***>**.*?</\1> match the presumed end of the opening tag

(NB: This markup <a onclick="valueOK('a>b')"> would end the match early. Doesn't matter here. Subsequent < pulls match to closing tag. But if you attempted to match only the opening tag, it might be truncated in rare cases.)

(?i)<([a-z][a-z0-9]*)[^>]*>**.*?**</\1> lazy match of all of tag's contents
(?i)<([a-z][a-z0-9]*)[^>]*>.*?**</\1>** match the closing tag — \1 refers to first capture group

IF condition — phone number w/optional parentheses around area code (and optional space after closing parens)
(\()?\d{3}(?(1)\) ?|[- \.])\d{3}[- \.]\d{4}

The IF condition can be set by a backreference (as here) or by a lookaround group.

(**()?**\d{3} optional group ( )? matches "(" prior to 3-digit area code \d{3} — group creates back reference #1
(?(1)\) ?**|**[-/ \.]**)** (1) refers to group 1, so if "(" exists, match ")" followed by optional space, else match one of these: "- / . "
\d{3}[- \.]\d{4} rest of phone number

groups can be named (assume a file of lastname, firstname altered using "preg_replace()")
(?#find)**(**\b.+**)**, **(**\b.*\b**)** (?#replace)**\2 \1**
(?#find)**(?P<lname>**\b.+**)**, **(?P<fname>**\b.*\b**)** (?#replace) **(?P=fname) (?P=lname)**

For a quick overview: http://www.codeproject.com/KB/dotnet/regextutorial.aspx.

For a good tutorial: http://www.regular-expressions.info.