Good	Better	Comments
`(whiskey)`	`(?:whiskey)`	Capture groups add unnecessary overhead and impact overall performance use them only when necessary.
`splunk|splash`	`spl(?:unk|ash)`	Try to “factor” on the left, when you can, while exposing required text. Less alternation is better.
`(?:aussie$\|gypsie$)`	`(?:aus|gyp)sie$`	Try to “factor” on the right when input text is close to end of the line. Most regex engines will anchor at end of line when “$” is present.
`0{3,7}`	`0000{0,4}`	Typically exposing required or literal text makes the engine execute the regex faster
`(.)*`	`.*`	Useless parenthesis add unnecessary overhead. As above, use them only when necessary.
`matty[:]`	`matty:`	The character class/set (indicated by `[]`) will add unnecessary overhead when not needed.
`^genti|^collar`	`^(?:genti|collar)`	Anchoring the regex at the beginning of the line will result in improved performance with most regex engines.
`delaney$\|connery$`	`(delaney|connery)$`	I said, anchor the regex!
`^src.*:`	`^src[^:]*:`	Using a negated character class/set instead of lazy/greedy quantifiers will typically result in faster regexes. Lazy/greedy quantifiers will make the regex engines backtrack which ultimately impacts overall performance.
`bride|brian`	`bri(?:de|an)`	Full alternation is more expensive than partial alternation. Also, in this case the regex engine will alternate only **AFTER** ‘`bri`’ has been matched.
`(?:edu|com|net|…)`	`(?:com|edu|net|…)`	Leading the engine to a match by placing the ost popular match first may result in faster execution in some engines.
`^.*(answer)`	`^.{42}(answer)`	Specifying an exact position inside the string and leading the engine to a match, will help improve performance drastically compared to using a simple greedy/lazy quantifier.
`.*?a`	`^.*a`	If **‘a’** is near the end of the input string will match faster as less backtracking will be required.
`.*a`	`^.*?a`	If **‘a’** is near the beginning of the input string the regex engine will match faster.
`:[^:]*:`	`:[^:]*+:`	Ex. in **‘:destination’** the second regex fails **faster.**
`:[^:]*:`	`:(?>[^:]*):`	Same as above, using different notation. Explanation: Atomic grouping or possessive quantifiers instruct the regex engine not to keep the states captured by * or + therefore preventing it from unsuccessfully backtracking and in turn failing faster.