链滴

sed 中的正则表达式

作者：jianhongli

原文链接：https://ld246.com/article/1554037624727

来源网站：链滴

<blockquote>
<p>GNU sed 中的基本正则与扩展正则在使用起来有些差异.有时为了简化语法会使用扩展正则.但是展正则后有些简单的语法亦有可能变得复杂.因此对两者之间的差异与细节有一个基本的了解很有必要;/p>
</blockquote>
<h3 id="字符加号-">字符加号 <code>+</code></h3>
<h4 id="基本语法-----">基本语法 [<code>BRE</code>]</h4>
<pre><code class="language-bash highlight-chroma">$ echo '+b=c' > foo
$ sed -n '/a+b/p' foo
a+b=c
</code></pre>
<h4 id="扩展语法-----">扩展语法 [<code>ERE</code>]</h4>
<pre><code class="language-bash highlight-chroma">$ echo '+b=c' > foo
$ sed -E -n '/a\+b/p' foo
a+b=c
</code></pre>
<h3 id="一个以上-a-跟随着字母-b---加号作为特殊元字符--">一个以上 a 跟随着字母 b [加号作为殊元字符]</h3>
<h4 id="基本语法------">基本语法 [<code>BRE</code>]</h4>
<pre><code class="language-bash highlight-chroma">$ echo aab > foo
$ sed -n '/a\+b/p' foo
aab
</code></pre>
<h4 id="扩展语法------">扩展语法 [<code>ERE</code>]</h4>
<pre><code class="language-bash highlight-chroma">$ echo aab > foo
$ sed -E -n '/a+b/p' foo
aab
</code></pre>
<h3 id="BRE-语法概览">BRE 语法概览</h3>
<table>
<thead>
<tr>
<th>语法</th>
<th>说明</th>
<th>备忘</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char</code></td>
<td>单个普通字符,匹配自身</td>
<td>无</td>

</tr>
<tr>
<td><code>*</code></td>
<td>Matches a sequence of zero or more instances of matches for the preceding regular exp
ession,
 which must be an ordinary character, a special character preceded by <code>\</
ode>, a <code>.</code>,
 a grouped regexp (see below), or a bracket expression.
s a GNU extension, a postfixed regular expression can also be followed by <code>*</code>;

for example,<code>a**</code> is equivalent to <code>a*</code>.
POSIX 1003.1
2001 says that <code>*</code> stands for itself when it appears at the start of
 a regula
 expression or subexpression,
but many nonGNU implementations do not support this a
d
 portable scripts should instead use <code>*</code> in these contexts.</td>
<td></td>
</tr>
<tr>
<td><code>.</code></td>
<td>Matches any character, including newline.</td>
<td></td>
</tr>
<tr>
<td><code>^</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>$</code></td>
<td>It is the same as <code>^</code>, but refers to end of pattern space.
<code>$</
ode> also acts as a special character only at the end of the regular expression or
 subexp
ession (that is, before <code>\)</code> or <code>|</code>), and its use at the end of
 subexpression is not portable.</td>
<td></td>
</tr>
<tr>
<td><code>\+</code></td>
<td>As <code>*</code>, but matches one or more. It is a GNU extension.</td>
<td></td>
</tr>
<tr>
<td><code>\?</code></td>
<td>As <code>*</code>, but only matches zero or one. It is a GNU extension.</td>
<td></td>
</tr>
<tr>
<td><code>\{i\}</code></td>
<td>As <code>*</code>, but matches exactlyisequences (iis a decimal integer; for portability

 keep it between 0 and 255 inclusive).</td>
<td></td>
</tr>
<tr>
<td><code>\{i,j\}</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>\{i,\}</code></td>

<td></td>
<td></td>
</tr>
<tr>
<td><code>\(regexp\)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>regexp1|regexp2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>regexp1regexp2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>\digit</code></td>
<td>Matches thedigit-th <code>\(...\)</code> parenthesized subexpression in the regular e
pression.
This is called a_back reference_. Subexpressions are implicitly numbered
y counting occurrences of <code>\(</code> left-to-right.</td>
<td></td>
</tr>
<tr>
<td><code>\n</code></td>
<td>Matches the newline character.</td>
<td></td>
</tr>
<tr>
<td><code>\char</code></td>
<td>Matcheschar, wherecharis one of <code>$</code>,<code>*</code>,<code>.</code>
<code>[</code>,<code>\</code>, or <code>^</code>.
Note that the only C-like bac
slash sequences that you can portably assume to be
interpreted are <code>\n</code>
nd <code>\\</code>; in particular <code>\t</code> is not portable, and matches a ‘t’ u
der
most implementations of <code>sed</code>, rather than a tab character.</td>
<td></td>
</tr>
<tr>
<td><code>[list]</code> or <code>[^list]</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
<h3 id="ERE-语法概览">ERE 语法概览</h3>
The only difference between basic and extended regular expressions is in the behavior of
a few characters: ‘?’, ‘+’, parentheses, braces (‘{}’), and ‘|’. While basic regular expr
ssions require these to be escaped if you want them to behave as special characters, when us
ng extended regular expressions you must escape them if you want them_to match a literal c
aracter_. ‘|’ is special here because ‘|’ is a GNU extension – standard basic regular expres
ions do not provide its functionality.