Korn Shell 93 Extended Patterns

Finnbarr P. Murphy

(fpm@fpmurphy.com)

According to `David Korn`, a shell is primarily a string processing language. Pattern matching is an important compoment of any such language and indeed `Korn Shell 93 (ksh93)` has excellent support for *`extended patterns`* as well as *`regular expressions`*. *`Extended patterns`* can be thought of as class or type of *`extended regular expressions`*. Both the *`bash`* and *`zsh`* shells have something similar but not as comprehensive. However, as usual, *`extended patterns`* are documented quite tersely in the *ksh93* man page.

The purpose of this post is to explain, with some examples, how to use the power of *`extended patterns`* in your *`ksh93`* scripts. It is assumed that you are reasonably familiar with basic regular expressions (BREs) as implemented in *`sed`*, *`grep`* or *`awk`*. If you need a introductory tutorial on regular expressions, here is one at *`IBM developerWorks`*.

The following table shows the the basic pattern matching *`operators`* in *`ksh93`*.

`?(pattern)`	Match if found 0 or 1 times
`*(pattern)`	Match if found 0 or more times
`+(pattern)`	Match if found 1 or more times
`@(pattern1|…)`	Match if any of the *patterns* found
`!(pattern)`	Match if no *pattern* found

Note that an *`operator`* must preceed a *`pattern`* in *`ksh93`* whereas in *`egrep, sed`* and *`awk, the operator`* is placed after the *`pattern`*.

Here is an example of how to use the above *operators* to modify the contents of the string *str*:

```
str="Joe Mike and Dave are all good friends"

print ${str//a?(re)/_}
# output: Joe Mike _nd D_ve _ _ll good friends

print ${str//g*(o)/_}
# output: Joe Mike and Dave are all _d friends

print ${str//+(o)/_}
# output: J_ Mike and Dave are all g_d friends

print ${str//@(Joe|Mike|Dave)/_}
# output: _ _ and _ are all good friends

print ${str//@(Joe|Mike|g*(o))/_}
# output: _ _ and Dave are all _d friends

print ${str//!(Joe)/_}
# output: _

print ${str//!(Joe|Mike|Dave)/_}
# output: _
```

In the above example, I have shown the expected output as a comment below each *`print`* statement. Here is another example which should futher clarify your understanding of these

pattern operators. Note particularly the output of of the last *print* statement.

```
str='An extended pattern expression'
print ${str//e/#}
print ${str//[^e]/#}
print ${str//+(e)/#}
print ${str//-(e)/#}
print ${str//?(e)/#}
print ${str//*(e)/#}
print ${str//!(e)/#}</pre>
```

which produces the following output.

```
An #xt#nd#d patt#rn #xpr#ssion
###e##e##e######e###e###e######
An #xt#nd#d patt#rn #xpr#ssion
An extended pattern expression
#A#n# #x#t#n#d#d# #p#a#t#t#r#n# #x#p#r#s#s#i#o#n#
#A#n# #x#t#n#d#d# #p#a#t#t#r#n# #x#p#r#s#s#i#o#n#
#
```

The following table show a number of pattern matching *interval quantifiers*.

