Alternative OS – Lecture 7

Plan:

1. Regular expressions

1. Regular expressions

Some text processing *commands* allow you to search for *patterns* instead of fixed strings.

Definition

We call a **pattern** something that can match one or *more* strings.

Example:

?at is a pattern that can be matched by *cat, rat, fat, eat* and others.

In Unix there is a convention on how to build up such *search patterns* for *text processing* programs. This convention is called *regular expressions syntax*.

Regular expressions come in three different flavors:

• **Anchors** which tie the pattern to a location on the line
• **Character sets** which match a character at a single position
• **Modifiers** which specify how many times to repeat the expression preceding them.

Regular expressions are used to *search* for strings or *search* and *replace* strings by several Unix commands. Some of the commands accept all of the regular expressions, others may accept only some of them.

Syntax of *regular expressions:*

.	match **any** single character except <newline>
*****	match **zero or more** instances of the single character (or meta-character) immediately preceding it
[abc]	match any of the characters enclosed in the square brackets
[a-d]	match any character in the enclosed range

[^exp]	match any character **not** in the expression **exp**
^abc	the regular expression "**abc**" must start at the **beginning of the line** (Anchor)
abc$	the regular expression must end at the **end of the line** (Anchor)
\	treat the next character literally. This is normally used to escape the meaning of special characters such as "**.**" and "*****". **For example**, * would match *****
\{n,m\}	match the regular expression preceding this a minimum number of **n** times and a maximum of **m** times (0 through 255 are allowed for n and m). **Remark:** The **\{** and **\}** sets should be thought of as single operators. In this case the **** preceding the bracket does not escape its special meaning, but rather turns on a new one.
\<abc\>	will match the enclosed regular expression as long as it is a separate word. Word boundaries are defined as beginning with a <newline> or anything except a letter, digit or underscore (_) or ending with the same or a end-of-line character. **Remark:** the **\<** and **\>** sets should be thought of as single operators.
\(abc\)	saves the enclosed pattern in a buffer. Up to nine patterns can be saved for each line. You can reference these latter with the **\n** character set. **Remark:** the **\(** and **\)** sets should be thought of as single operators.
\n	where **n** is between 1 and 9. This matches the **n**th expression previously saved for this line. Expressions are numbered starting from the left. **Remark: \n** should be thought of as a single operator.
&	print the previous search pattern (used in the replacement string)

Examples of commonly used *regular expressions*:

regular expression	**matches**
cat	the string **cat**
.at	any occurrence of a letter, followed by **at**, such as cat, rat, mat, bat, fat, hat
xy*z	any occurrence of an **x**, followed by zero or more **y**'s, followed by a **z**.
^cat	**cat** at the beginning of the line
cat$	**cat** at the end of the line
*	any occurrence of an asterisk
[cC]at	**cat** or **Cat**
[^a-zA-Z]	any occurrence of a non-alphabetic character, like **9, 7, -, +, =, #**
[0-9]$	any line ending with a number
[A-Z][A-Z]*	one or more upper case letters
[A-Z]*	zero or more upper case letters (In other words, anything.)