Document Number:	*Copy Number:*

TATA CONSULTANCY SERVICES

vILP – Advanced Filters and Regular Expression

Content Manual Version 1.0

April 2014

DOCUMENT RELEASE NOTICE

Notice No.
Client
Project
Document details

Name	Version No.	Author	Description
vILP – Advanced Filters and Regular Expression	1.1	PROGOTI	An overview of use of filters with regular expressions

Revision details:

Action taken (Y/N)	Preceding page No.	New page No.	Revision description

Change Register serial numbers covered:
The documents or revised pages are subject to document control.
Please keep them up-to-date using the release notices from the distributor of the document.
These are confidential documents. Unauthorised access or copying is prohibited.

Approved by : _____ Date: _____

Authorised by: _____ Date: _____

DOCUMENT REVISION LIST

Client
Project
Document Name
Release Notice Reference (for release)

Rev. No.	Revision	Revision description	Page No.	Prev page No.	Action taken	Addenda /New page	Release Notice reference

CONTENTS

COURSE 1- FILTERS USING REGULAR EXPRESSION

1.1 Objective

- To provide overview of the Regular expression.
- How to use them with grep or sed for a given problem statement
- What are the various operations we can use with Regular Expression

1.2 Introduction

A regular expression is a set of characters that specify a pattern.

Regular expressions are used when you want to search for specific lines of text containing a particular pattern. Most of the UNIX utilities operate on ASCII files a line at a time. Regular expressions search for patterns on a single line, and not for patterns that start on one line and end on another.

It is simple to search for a specific word or string of characters. Almost every editor on every computer system can do this. Regular expressions are more powerful and flexible. You can search for words of a certain size. You can search for a word with four or more vowels that end with an "s". Numbers, punctuation characters, you name it, a regular expression can find it. What happens once the program you are using find it is another matter. Some just search for the pattern. Others print out the line containing the pattern. Editors can replace the string with a new pattern. It all depends on the utility.

One point to note is that regular expressions are not wildcards. The regular expression 'c*t' does not mean 'match "cat", "cot"' etc. In this case, it means 'match zero or more 'c' characters followed by a t', so it would match 't', 'ct', 'cccct' etc.

Secondly we shouls understand that shell meta-characters are expanded before the shell passes the arguments to the program. To prevent this expansion, the special characters in a regular expression must be quoted when passed as an option from the shell.

1.2.1 The Structure of a Regular Expression

There are three important parts to a regular expression.

- Anchors : These are used to specify the position of the pattern in relation to a line of text.
- Character Sets : The set of characters that match one or more characters in a single position.
- Modifiers: They specify how many times the previous character set is repeated.

A simple example that demonstrates all three parts is the regular expression is :

"^#*"

Here ,

- The up arrow , **“^”**, is an anchor that indicates the beginning of the line.
- The character **"#"** is a simple character set that matches the single character "#".
- The asterisk **“*”** is a modifier. In a regular expression it specifies that the previous character set can appear any number of times, including zero.

1.2.2 Types Of Regular Expression

There are also two types of regular expressions:

- the "Basic" regular expression,(BRE)
- the "extended" regular expression.(ERE)

A few utilities like awk and egrep use the extended expression. Most use the "regular" regular expression. From now on, if I talk about a "regular expression," it describes a feature in both types.

1.2.2 The Anchor Characters: ^ and $

Anchores are used when we want to search for a pattern that is at one end or the other, of a line. The character **"^"** is the starting anchor, and the character **"$"** is the end anchor. The regular expression "^A" will match all lines that start with a capital A. The expression "A$" will match all lines that end with the capital A. If the anchor characters are not used at the proper end of the pattern, then they no longer act as anchors. That is, the "^" is only an anchor if it is the first character in a regular expression. The "$" is only an anchor if it is the last character. The expression "$1" does not have an anchor. Neither is "1^". If you need to match a "^" at the beginning of the line, or a "$" at the end of a line, you must escape the

special characters with a backslash.

Following list provides a summary:

Pattern	Matches
^A	"A" at the beginning of a line
A$	"A" at the end of a line
A^	"A^" anywhere on a line
$A	"$A" anywhere on a line
^^	"^" at the beginning of a line
$$	"$" at the end of a line

The use of "^" and "$" as indicators of the beginning or end of a line is a convention the utilities in unix use. The vi editor uses these two characters as commands to go to the beginning or end of a line.

1.2.3 The Character Set

The character set also called “character class” in a regular expression , is used to tell the regex engine to match only one out of several characters. The simplest character set is a character. We have to simply place the characters we want to match between square brackets. If we want to match an a or an e, we have to use [ae]. For example, we can use gr[ae]y to match either gray or grey.

- A character set matches only a single character.
  In case of the above example, `gr[ae]y` does not match `graay`, `graey` or any such thing.
- The order of the characters inside a character set does not matter. The results are identical.
- A hyphen can be used inside a character class to specify a range of characters.
  Example , [0-9] matches a single digit between 0 and 9. Alphabets range also can be specified such as [a-z],[A-Z] etc We can use more than one range. If you wanted to search for a word that starts with a capital letter "T", is the first word on a line,the second letter is a lower case letter,is exactly three letters long, and the third letter is a vowel, the regular expression would be "^T[a-z][aeiou] "
- Some characters have a special meaning in regular expressions. If we want to search for such a character, we have to escape it with a backslash.

1.2.3.1 Exception in the character class

If we want to search for all the characters except those in the square bracket, then the ^ (Caret) symbol needs to be used as the first character after open square bracket. The expression **"^[^aeiou]"** is to searc for a line which does not start with the vowel letter. Here , the first caret symbol in regular expression represents beginning of the line. However, caret symbol inside the square bracket represents “except” — i.e match except everything in the square bracket.
Thus some times the anchors “^”and “$” , can no't be considered as anchor . Similarly the characters "]" and "-" do not have a special meaning if they directly follow "[". Here are some examples:

Regular Expression	Matches
[]	The characters "[]"
[0]	The character "0"
[0-9]	Any number
[^0-9]	Any character other than a number
[-0-9]	Any number or a "-"
[0-9-]	Any number or a "-"
[^-0-9]	Any character except a number or a "-"
[]0-9]	Any number or a "]"
[0-9]]	Any number followed by a "]"
[0-9-z]	Any number, or any character between "9" and "z".
[0-9\-a\]]	Any number, or a "-", a "a", or a "]"

1.2.3.2 Match any character

The character "." is one one of thespecial meta-characters. By itself it will match any character, except the end-of-line character. Thus the pattern that will match a line with a single characters is

^.$

1.2.3.3 Repeating character sets

The third part of a regular expression is the modifier. It is used to specify how may times you expect to see the previous character set.

The repetition modifier * find no or one, one or more, and zero or more repeats, respectively.

Examples:

Expression	**Matches**
Go*gle	Gogle,Google,Gooogle, and so on.
"[0-9]*"	zero or more numbers.

Now as an example ,the pattern "^#*" matches any number of "#'s" at the beginning of the line, including zero. That is this will match every line, because every line starts with zero or more "#'s". Is this usefull?

At first glance, it might seem that starting the count at zero is useless. However it is not so. To search for an unknown number of characters it is very important. Suppose you wanted to look for a number at the beginning of a line, and there may or may not be spaces before the number. Just use "^ *" to match zero or more spaces at the beginning of the line. If you need to match one or more, just repeat the character set. That is, "[0-9]*" matches zero or more numbers, and "[0-9][0-9]*" matches one or more numbers.

1.2.3.3.1 Matching a specific number of sets with \{ and \}

We cannot specify a maximum number of sets with the "*" modifier. There is a special pattern we can use to specify the minimum and maximum number of repeats, by putting those two numbers between "\{" and "\}". Normally a backslash turns off the special meaning for a character. A period is matched by a "\." and an asterisk is matched by a "*".

- A modifier can specify amounts such as none, one, or more; one or more; zero or one; five to ten; and exactly three.

  For example , A user name is a string beginning with a letter followed by at least two, but not more than seven letters or numbers followed by the end of the string. Then the regular expression is **^[A-z][A-z0-9]{2,7}**

- A repetition modifier must be combined with other patterns; the modifier has no meaning by itself.

  For example , modifiers like "*" and "\{1,5\}" only act as modifiers if they follow a character set. If they were at the beginning of a pattern, they would not be a modifier.

1.3 grep with Regular expression

Search for 'vivek' in /etc/passswd

grep vivek /etc/passwd

Search vivek in any case (i.e. case insensitive search)

grep -i -w vivek /etc/passwd

Search vivek or raj in any case

grep -E -i -w 'vivek|raj' /etc/passwd

1.3.1 Line and word anchors

Search lines starting with the vivek only

grep ^vivek /etc/passwd

To display only lines starting with the word vivek only i.e. do not display vivekgite, vivekg

grep -w ^vivek /etc/passwd

To Find lines ending with word foo

grep 'foo$' filename

1.3.2 Character classes

To match Vivek or vivek.

grep '[vV]ivek' filename

OR

grep '[vV][il][Vv][Ee][kK]' filename

To match digits (i.e match vivek1 or Vivek2 etc)

grep -w '[vV]ivek[0-9]' filename

1.3.3 Wildcards

To match all 3 character word starting with "b" and ending in "t".

grep '\<b.t\>' filename

Where,

•\< Match the empty string at the beginning of word
•\> Match the empty string at the end of word.

Print all lines with exactly two characters

grep '^..$' filename

Display any lines starting with a dot and digit

grep '^\.[0-9]' filename

1.3.4 Escaping the dot

To find an IP address 192.168.1.254

grep '192\.168\.1\.254' /etc/hosts

1.3.5 Search a Pattern Which Has a Leading – Symbol

Searches for all lines matching '--test--' using -e option . Without -e, grep would

attempt to parse '--test--' as a list of options

grep -e '--test--' filename

1.3.6 Test Sequence

To Match a character "v" two times

egrep "v{2}" filename

To match both "col" and "cool"

egrep 'co{1,2}l' filename

Note : egrep is the same as **grep -E**. It interpret PATTERN as an extended regular expression

1.3.7 grep OR Operator

Suppose the file “employee” has the following data:

```
Id Name Skill Dept Salary
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
400  Nisha   Manager    Marketing   $9,500
500  Randy   Manager    Sales       $6,000
```

To find the records of those who are either from Tech or Sales dept.

We can use the following syntaxes :

1) Syntax : grep 'word1\|word2' filename
grep 'Tech\|Sales' employee

2) Syntax : grep -E 'pattern1|pattern2' fileName

grep -E 'Tech|Sales' employee

3) Syntax : grep -e Tech -e Sales filename

grep -e Tech -e Sales employee

1.3.8 grep AND Operator

There is no AND operator in grep. But, we can simulate AND using

- grep -E option.

  Syntax : grep -E 'word1.*word2 ' filename
  grep -E 'word1.*word2|'word2.*word1' filename

- multiple grep command separated by pipe

  Syntax : grep 'word1' filename | grep 'word2'

For example , to find all Manager and Sales ,

grep -E 'Manager.*Sales|Sales.*Manager' employee

OR

grep Manager employee | grep Sales

1.4 Video

http://www.youtube.com/watch?v=WX5Zfflvdt0

1.5 External link

http://www.cs.nyu.edu/~mohri/unix08/lect4.pdf

1.6 Quiz

Q 1. Which of the following regular expressions will match the pattern TCS only at the beginning of

a) \^TCS
b) ^TCS
c) $TCS
d) TCS$

Answer : b

Q 2. Which of the following commands uses correct syntax for matching the patterns bunk or bank at the end of a line of text?

a) grep 'b[au]nk$' myFile
b) grep “b[au]nk$” myFile
c) grep b[au]nk$ myFile
d) grep b[a-u]nk$ myFile

Answer : a

Q 3 Which of the following meanings can a $ character assume within a regular expression?

a) When a regular expression is single quoted, the $ character acts as an anchor if placed at the end of the expression.
b) When a regular expression is single quoted and $ is there , the $ character acts as an anchor.
c) When a regular expression is single quoted, the $ character is treated as part of the syntax of shell variables.
d) When a regular expression is double quoted, the $ character acts as an anchor if placed at the end of the expression.

Answer : a

Q 4. In the expression **grep "A 'stop\!' sign" myfile ,** the character \ is used to

a) Represent the pattern to be searched is “stop\!”

b) Represent to search for patterns like stops!, stop!,stope! Etc. , that zero or one character.

c) escape the ! character

d) print the searched lines in new line.

Answer : c

Q 5 . The Regular Expression that matches any one of the following:
a b c d 5 6 7 8 X Y Z is..
a) [a-d][5-8][X-Z]
b) [a-d5-8X-Z]
c) [a-d/5-8/X-Z]
d) ['a-d"5-8"X-Z']

Answer : b