Scripting & Computer Environments

Regular Expressions (Regex)

IIIT-H

Aug 26, 2015

...Previously & Today...

Previously: Basic filters

- Redirection & Piping (`>`, `>>`, `<`, `|`)
- Simple Filters (`cat`, `wc`, `tr`, `tee`, `...`)
- Shell Wildcards (`?`, `*`, `[]`, `!`, ∧, `-`, `...`)

Today:

- Basics of Regex
- Regex-aware Filters: `grep`✓, `sed`, `awk` ...

Brainstorm

1. Filters?
2. Shell wildcards/metacharacters? Examples? Uses?
3. Regex? Applications? Regex vs Wildcards?

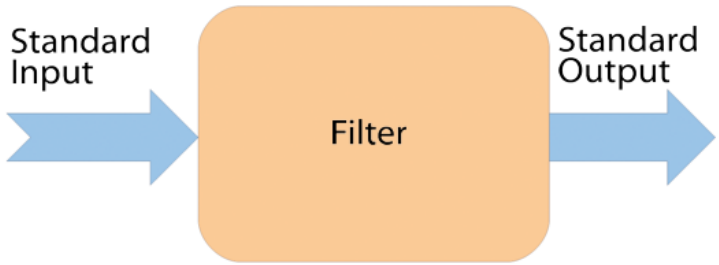
Brainstorm

1. Filters?
2. Shell wildcards/metacharacters? Examples? Uses?
3. Regex? Applications? Regex vs Wildcards?

Brainstorm

1. Filters?
2. Shell wildcards/metacharacters? Examples? Uses?
3. Regex? Applications? Regex vs Wildcards?

Recap: Filters



- Simply, commands that use both the `STDIN` and `STDOUT`.
- Read input stream $\rightarrow$ [transform it] $\rightarrow$ output the result.
- Example application domain: text processing/manipulation.

  e.g. `cat`, `wc`, `tr`, `grep`, `sed`, `awk`, etc

Recap: Shell Wildcards/Metacharacters

- Characters with special meaning to the shell

```
*  ?  <  >  |  [ ]  '  "  ;  { }  ( )  !  &  ^  \n ...
```

- Expanded by the shell first (a.k.a. Globbing).
- ? matches any single character.
- * matches 0+ number of characters (but '.' at beginning and '/' of pathnames).
- [...] matches any element in the set.
- Characters with special meaning inside []: - (hyphen), ∧, !
- \ turns off their special meaning (a.k.a. *Escaping*).

Examples

```
ls -l ?????
```

```
rm -i *.c
```

```
cp [A-Z]* MyDir    ,    cp [A-Z] MyDir
```

```
ls -l file[!A-Z]*   /  ls -l file[^A-Z]*
```

```
echo \\
```

Recap: Searching for Files

- One of the basic operations of any OS.
- Linux offers some commands: `locate`, `whereis`, `find` ...

```
find <where> -name <search criteria>
```

```
find / -name 'file[^12].c'

find ~ -name 'My??*'

find . -name '*199[0-9]*'

find .. -perm -644

find . -size -2000k -mtime 1 -name '*.html'
```

Checkpoint!

1. Decode this command:

```
tr 'a-z' '0-9' < input.txt | sort -rn | uniq | tail > ouput.txt
```

2. Extract the 3^{rd} field from lines 10 through 20 of /etc/passwd.
3. The first 3 largest files in the current directory? Only name and size?
4. Remove digits from all C programs from the home directory that have a line count of 2.

Checkpoint!

1. Decode this command:

```
tr 'a-z' '0-9' < input.txt | sort -rn | uniq | tail > ouput.txt
```

2. Extract the 3^{rd} field from lines 10 through 20 of `/etc/passwd`.

3. The first 3 largest files in the current directory? Only name and size?

4. Remove digits from all C programs from the home directory that have a link count of 2.

Checkpoint!

1. Decode this command:

```
tr 'a-z' '0-9' < input.txt | sort -rn | uniq | tail > ouput.txt
```

2. Extract the 3^{rd} field from lines 10 through 20 of `/etc/passwd`.

3. The first 3 largest files in the current directory? Only name and size?

4. Remove digits from all C programs from the home directory that have a link count of 2.

Checkpoint!

1. Decode this command:

```
tr 'a-z' '0-9' < input.txt | sort -rn | uniq | tail > ouput.txt
```

2. Extract the 3^{rd} field from lines 10 through 20 of `/etc/passwd`.

3. The first 3 largest files in the current directory? Only name and size?

4. Remove digits from all C programs from the home directory that have a link count of 2.

Regex

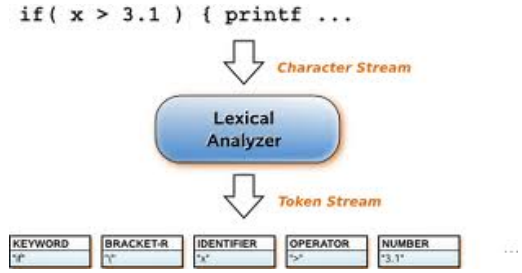
Regular Expression (Regex)

- A specific search pattern entered to find a particular target string.
- Is like a mathematical expression (operands + operators).
- Interpretted by the command, and not by the shell.
- Application areas?

Regular Expression (Regex)

- A specific search pattern entered to find a particular target string.
- Is like a mathematical expression (operands + operators).
- Interpretted by the command, and not by the shell.
- Application areas?

Text mining



Translators (e.g. compiler)

Security
(e.g. injection attacks, data validation ...)

web scraping, crawling, search engines ...

Regex-Aware Filters: grep

Remember the `find` command?

`grep`

"Globally (g) search a file for a regular expression (re) and print (p) the result."

```
grep [options] pattern file(s)
```

Some options: `-i`, `-v`, `-c`, `-e`, `--color=[auto|always|never]` ...

- Supports both Basic Regular Expressions (BRE) and Extended Regular Expressions (ERE).
- man/info grep for more ✓

Regex Metacharacters

- Regex metacharacters overshadow the shell's.

The . & * Metacharacters

- '.' matches any single character except the newline character (\n).
- Similar to the '?' shell metacharacter.
- '*' matches 0+ occurence of the *immediately preceding* character.
- The combination .* means *"any or none"* (same as the shell's *).

Regex Metacharacters

- Regex metacharacters overshadow the shell's.

The . & * Metacharacters

- '.' matches any single character except the newline character (\n).
- Similar to the '?' shell metacharacter.
- '*' matches 0+ occurence of the *immediately preceding* character.
- The combination .* means *"any or none"* (same as the shell's *).

Example

`.`	`bb*`
`ab..`	`s*printf`
`b*`	`A.*Z`

The [ ] Metacharacter (a.k.a. Character Class)

- Matches any one of the enclosed characters within.
- Use hyphen (-) *within it* to specify range.
- Use caret ($\wedge$) *within it* to negate a character class.

```
[bcf]ar          [a-zA-Z]*          xyz[^6-9]
```

The [ ] Metacharacter (a.k.a. Character Class)

- Matches any one of the enclosed characters within.
- Use hyphen (-) *within it* to specify range.
- Use caret ($\wedge$) *within it* to negate a character class.

```
[bcf]ar          [a-zA-Z]*          xyz[^6-9]
```

Positional Markers: ($\wedge$, \$, $<$ and $>$)

- $\wedge$ matches beginning of a line.
- \$ matches end of a line.
- $<$ matches start of a word.
- $>$ matches end of a word.

Example

```
ls -l | grep ‘^d’

^$

grep ‘^bash’ /usr/share/dict/words


grep ‘shell$’ /usr/share/dict/words


grep ‘\<computer’ /usr/share/dict/words


grep ‘computer\>’ /usr/share/dict/words
```

Regex Metacharacters: Quantifiers

Most of them must be escaped (in BRE)!

- *Asterisk/Kleene star (*)* - matches 0+ occurence(s) of an expression.
- *Optional (\?)* - matches 0 or 1 occurence of an expression
- *Alternation (\|)* - matches either of the expressions it sits between.
- *Plus (\+)* - matches 1+ occurrence(s) of an expression

```
d*
```

```
M[sr]\|Miss
```

```
Saviou\?r
```

```
ho\+ray
```

Regex Metacharacters: Repetetion

a.k.a. Interval Regular Expressions (IRE)

- {m} matches the preceding regex *exactly* 'm' times.
- {m,} matches the preceding regex *atleast* 'm' times.
- {m,n} matches the preceding regex 'm' to 'n' times.

```
a\{3\}
SR\{5,\}
SR\{,3\}
AB\{1,4\}
```

Q: Write the regex metacharacters *, + and ? in this notation.

Regex Metacharacters: Repetetion

a.k.a. Interval Regular Expressions (IRE)

- {m} matches the preceding regex *exactly* 'm' times.
- {m,} matches the preceding regex *atleast* 'm' times.
- {m,n} matches the preceding regex 'm' to 'n' times.

```
a\{3\}          SR\{,3\}

SR\{5,\}        AB\{1,4\}
```

Q: Write the regex metacharacters `*`, `+` and `?` in this notation.

Regex Metacharacters: Grouping

The Group Metacharacter: '\(expr\)'

- Used to group expressions together and match them.

```
a\(bc\)*        an\(an\) \+          \(w\(xy\)\{2\} z\)\{2\}
```

Regex Metacharacters: Grouping

The Group Metacharacter: '\(expr\)'

- Used to group expressions together and match them.

```
a\(bc\)*        an\(an\) \+          \(w\(xy\)\{2\} z\)\{2\}
```

The Save Metacharacter (Backreference): \1...\9

- Copies a matched string to one of 9 buffers for later reference.
- The 1st matched text copied to buffer 1, the 2nd to buffer 2 ...

```
\( [A-Z] \) .* \1
```

(Read about \b with backreference. You will need it.)

Regex Metacharacters: Shortcuts

More readable Named Character Classes exist in dealing with more complex expressions.

- `[:alnum:]` - alphanumeric characters; same as `[a-zA-Z0-9]`
- `[:alpha:]` - alphabetic characters; same as `[a-zA-Z]`
- `[:digit:]` - digits; same as `[0-9]`
- `[:upper:]` - upper case characters; same as `[A-Z]`
- `[:lower:]` - lower case characters; same as `[a-z]`
- `[:space:]` - any white space character, including tabs.
- `[:punct:]` - Punctuation characters.

```
ls -l | grep [[:digit:]]
```

Extended Regular Expressions (EREs)

- No need to escape metacharacters.
- Thus, cleaner and more readable.
- Defines additional metacharacter sets.
- Use `grep` with the `-E` flag.
- Alternatively, use `egrep` without `-E`.

```
ls -l | grep -E 'iii?t'

egrep '(ha+){1,3}'
```

Checkpoint!

1. Regex to match the following patterns:

```
cat, caat, caaa...at, etc
cat, ct, at, t
dog, Dog, dOg, DOG, dOG, etc
```

2. Decode these Regexs:

```
grep "^mo.*ing$" /usr/share/dict/words

grep '[[:digit:]bc] [^x-y]*$' /usr/share/dict/words
```

3. Given the previous dictionary file, write a regex to:
   - Find all words that begin and end with a vowel.
   - Find all five-character words that begin and end with a vowel.
   - Find all five-character words that begin and end with the same vowel.

Checkpoint!

1. Regex to match the following patterns:

```
cat, caat, caaa...at, etc
cat, ct, at, t
dog, Dog, dOg, DOG, dOG, etc
```

2. Decode these Regexs:

```
grep "^mo.*ing$" /usr/share/dict/words

grep '[[:digit:]bc] [^x-y]*$' /usr/share/dict/words
```

3. Given the previous dictionary file, write a regex to:
   - Find all words that begin and end with a vowel.
   - Find all five-character words that begin and end with a vowel.
   - Find all five-character words that begin and end with the same vowel.

Checkpoint!

1. Regex to match the following patterns:

```
cat, caat, caaa...at, etc
cat, ct, at, t
dog, Dog, dOg, DOG, dOG, etc
```

2. Decode these Regexs:

```
grep "^mo.*ing$" /usr/share/dict/words

grep '[[:digit:]bc] [^x-y]*$' /usr/share/dict/words
```

3. Given the previous dictionary file, write a regex to:
   - Find all words that begin and end with a vowel.
   - Find all five-character words that begin and end with a vowel.
   - Find all five-character words that begin and end with the same vowel.