# Scripting & Computer Environments

*Advanced Filters*

IIIT-H

Aug 14, 2013

# ...Previously & Today...

Previously: Basic filters

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

Today: Power filters

- Regular Expression (Regex) basics
- 2 Regex-Aware Filters:
  1. `grep`
  2. `sed`

# ...Previously & Today...

Previously: Basic filters

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

Today: Power filters

- Regular Expression (Regex) basics
- 2 Regex-Aware Filters:
  1. `grep`
  2. `sed`

# Brainstorm

- Filters?
- Shell metacharacters?
- Regular expressions?

# Filters

- Simply, commands that use both the `STDIN` and `STDOUT`
- Read input stream → [transform it] → output the result.
- Example application: text filtering

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

# Recap: Shell Metacharacters/Wildcards

- Characters with special meaning to the shell

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

- Expanded by the shell first.

- `?` matches any single character

- `*` matches 0+ number of characters (but ‘.’ and ‘/’ )

- `[]` matches any element in the set.

- Some characters with special meaning inside `[]`: `-` (hyphen), $\wedge$, !

- \ turns off the special meaning

### Example

```
ls -l ?????

rm -i *.c

cp [A-Z]* MyDir

ls -l file[^A-Z]*   or  ls -l file[!A-Z]*

echo \\
```

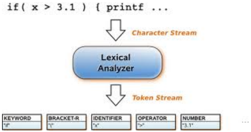
## Regular Expression (Regex)

- 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 ...)

DNA base sequences

# 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 files(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` |

# 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 Character Class Metacharacter: ‘[ ]’

- 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.

## The Character Class Metacharacter: ‘[ ]’

- 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!

- *Asterisk (*)* - 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

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

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

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

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\}
```

## The Save Metacharacter (Backreference): ‘\(. . .\)’

- 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: 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): ‘\(...\)’

- 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 classes exist in dealing with more complex expressions.

## Named Character Classes

- `[: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:]` - Punctutation 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}’
```

## Examples

1. Decode these Regexs:

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

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

2. Find all 5-character words from `/usr/share/dict/words` that begin with ‘I’ and end with ‘a’.

3. Match lines containing the years 1992-2009 from a file named `file.txt`

4. Search for a 7-digit phone number, possibly with a space or hyphen in the middle (e.g. 123-4567 or 123 4567), from `teldir.txt`

## Examples

1. Decode these Regexs:

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

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

2. Find all 5-character words from `/usr/share/dict/words` that begin with ‘I’ and end with ‘a’.

3. Match lines containing the years 1992-2009 from a file named `file.txt`

4. Search for a 7-digit phone number, possibly with a space or hyphen in the middle (e.g. 123-4567 or 123 4567), from `teldir.txt`

## Examples

1. Decode these Regexs:

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

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

2. Find all 5-character words from `/usr/share/dict/words` that begin with ‘I’ and end with ‘a’.

3. Match lines containing the years 1992-2009 from a file named `file.txt`

4. Search for a 7-digit phone number, possibly with a space or hyphen in the middle (e.g. 123-4567 or 123 4567), from `teldir.txt`

## Examples

1. Decode these Regexs:

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

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

2. Find all 5-character words from `/usr/share/dict/words` that begin with ‘I’ and end with ‘a’.

3. Match lines containing the years 1992-2009 from a file named `file.txt`

4. Search for a 7-digit phone number, possibly with a space or hyphen in the middle (e.g. 123-4567 or 123 4567), from `teldir.txt`

# Regex-Aware Filters: `sed`

- `Sed` stands for **s**tream **ed**itor.
- Derived from `ed`, the original unix editor.
- A powerful *non-interactive* text manipulation tool.
- Operates on a stream of text it receives (e.g. from `STDIN`, pipeline) on the fly and writes the output to `STDOUT`.
- Line-based processing cycle.
  - `read` → `buffer (aka pattern space)` → `edit` → `print`
- A complete programming language (see a game written in `sed`).

## sed usage

`sed [options] ‘instruction’ file`

- `instruction` - is user-supplied edit command with the form: `‘address action’`
- `address` - specifies where in the text to take the action at.
- `action` - specifies action commands ( substitute, delete, print, etc).
- Common options include:
  - `-n` suppress the default printing when using the print (p) command.
  - `-e` for multiple instructions per line, each preceded by it.
  - `-f <file>` read instruction from `<file>`.

## Address Specifiers

`sed [option] ‘address action’ <filename>`

`address` can be specified as:

- `‘n action’` → take `<action>` at line number n.
- `‘m,n action’` → take `<action>` between lines m and n.
- `‘m~n action’` → starting from line m, take `<action>` on every n$^{th}$ line from m.
- `‘$ action’` → take `<action>` on the last line.
- `‘N! action’`→ take `<action>` on all but line n.

## Action Specifiers

```
sed [options] ‘address action’ <filename>
```

`action` can be:

- `p` print line(s).
- `d` delete line(s).
- `s/old/new` substitute *first occurrence* of ‘old’ by ‘new’.
- `w <filename>` write edited output to `<filename>`.
- `q` quit after reading specified lines.

# sed Operations: Print (without regex)

```
sed -n ‘address p’ filename
```

## Example

```
sed -n ‘3p’ file.txt                       (try without -n)

sed -n ‘1,5p’ file.txt

sed -n ‘2~2p’ file.txt

sed -n ‘$p’ file.txt

sed -n ‘4,$!p’ file.txt

sed -n -e ‘1p’ -e ‘3,5p’ file.txt

sed -n ‘1p;3,5p’ file.txt                  (; is delimiter)

sed ‘10q’ file.txt                         (head??)
```

## Print Format (with regex)

```
(1)  sed -n ‘/regex/p’ filename
(2)  sed -n ‘/regex/, Np’ filename
(3)  sed -n ‘N, /regex/p’ filename
(4)  sed -n ‘/regex1/,/regex2/p’ filename
```

1. emulates `grep`
2. matches `regex` upto the N$^{th}$ line
3. matches `regex` from N$^{th}$ line onwards
4. matches lines between the two regexs.

### Example

```
ls -l | sed -n ‘/^.\{5\}w/p’

ls -l | sed -n ‘/^.....w/p’

sed -n ‘/foo/,/bar/p’ MyFile.txt
```

## Print Format (with regex)

```
(1)  sed -n ‘/regex/p’ filename
(2)  sed -n ‘/regex/, Np’ filename
(3)  sed -n ‘N, /regex/p’ filename
(4)  sed -n ‘/regex1/,/regex2/p’ filename
```

1. emulates **grep**
2. matches **regex** upto the N$^{th}$ line
3. matches **regex** from N$^{th}$ line onwards
4. matches lines between the two regexs.

### Example

```
ls -l | sed -n ‘/^.\{5\}w/p’

ls -l | sed -n ‘/^.....w/p’

sed -n ‘/foo/,/bar/p’ MyFile.txt
```

# sed Operations: Deletion

## Delete Format

```
(1)  sed ‘address d’ filename

(2)  sed ‘/regex/d’ filename
```

1. without regex
2. with regex

## Example

```
sed ‘1,5d’ Myfile.txt

sed ‘/b[oa]*/d’ Myfile.txt

sed ‘/^$/d’ Myfile.txt

cat Myfile.txt | sed ‘/^....$/d’
```

# sed Operations: Deletion

## Delete Format

```
(1)  sed ‘address d’ filename

(2)  sed ‘/regex/d’ filename
```

1. without regex
2. with regex

## Example

```
sed ‘1,5d’ Myfile.txt

sed ‘/b[oa]*/d’ Myfile.txt

sed ‘/^$/d’ Myfile.txt

cat Myfile.txt | sed ‘/^....$/d’
```

# Substitution (without regex)

Find-and-replace is what `sed` is best at.

## Substitution Operator (s//)

```
sed '[address] s/old/new/flags' filename
```

- Searches for occurrence of `old` and substitutes it with `new` at the specified address (optional).
- Common flags include :
  - `a number` specifies which occurrence must be replaced.
  - `g` replaces every (global) occurrence of `old` with `new`.
  - `i` case-insensitive operation.
  - `w filename` write to the given file.

### Example

```
sed ‘s/one/ek/’ hinglish.txt

sed -n ‘s/four/char/gp’ hinglish.txt

sed -n ‘s/three/teen/gpw output.txt’ hinglish.txt

sed -n ‘1,3s/four/char/pw output.txt’ hinglish.txt
```

### Substitution Format (with regex)

```
sed '/regex/s/old/new/flags' filename
```

- Searches for pattern `<old>` and replaces with `<new>` string wherever `<regex>` matches.
- The expression `/regex/` is optional.

### Example

```
sed ‘/#/s/include/define/g’ input.txt      (@ lines with #)

sed ‘s/saviou\?r/SAVIOR/g’ input.txt

sed ‘s/singer/lead &/’ input.txt         (& is an operator)

sed ‘s/\(Day\)\(Happy\)\(Independence\)/\2 \3 \1 /g’ input.txt
```