Regular Expressions
`grep` and `sed`
...

07-131 Great Practical Ideas in CS

Instructors: Eduardo Feo-Flushing & Giselle Reis

Course website: https://web2.qatar.cmu.edu/cs/07131/

Regular expressions

- What's a shell?
  - Execution commands
  - Scripting commands
  - Interactive
- A bit of history
  - 1971: Thompson shell
  - 1979: Bourne shell
  - 1989: Bourne-again shell (bash)

Character classes

Class	Matches
`[abc]`	a or b or c
`[a-z]`	Any single lowercase letter
`\s`	whitespace
`\d`	digit
`\w`	alphanumeric char
`.`	any single character (ie, wildcard)

More character classes

Class	Matches
`[abc]`	a or b or c
`[a-z]`	a lowercase letter
`\s`	whitespace
`\d`	digit
`\w`	alphanumeric char
`.`	any single character (ie, wildcard)

Class	Matches
`[^abc]`	Not any of a, b or c
`[^A-Za-z]`	Not a character
`\S`	Not a whitespace
`\D`	Any non-digit
`\W`	Any non-alphanumeric char
`\.`	Just a period

Quantification

Quantifier	Matches
`a?`	Zero or one (ie. optional)
`a*`	Zero or more
`a+`	One or more
`a{3}`	Exactly 3
`a{3,}`	3 or more
`a{3,6}`	Between 3 and 6 (inclusive)

Extras

- **Boolean "or"**
  - A vertical bar separates alternatives. For example, `gray|grey` can match "gray" or "grey".
- **Grouping**
  - Parentheses are used to define the scope and precedence of the operators (among other uses). For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of "gray" or "grey".
- **Positional**
  - `^` - matches the beginning of the line.
  - `$` - matches the end of the line.

Example: phone numbers

- Multiple possible strings
  - `123-456-7890`
  - `1234567890`
  - `456-789-1234`
- But the formats follow a few patterns
  - `###-###-####`
  - `##########`

```
(\d{3}-?){2}\d{4}
```

Example - phone numbers

```
(\d{3}-?){2}\d{4}
```

Matches any 3 digits

Example - phone numbers

```
(\d{3}-?){2}\d{4}
```

Matches an optional hyphen

Example - phone numbers

Ex:
123-456-
123456-
123456

```
(\d{3}-?){2}\d{4}
```

Matches 2 groups of 3 digits

Example - phone numbers

```
(\d{3}-?){2}\d{4}
```

Matches 2 groups of 3 digits, then
4 more digits

grep

- **G**lobal **R**egular **E**xpression **P**rint
- `grep [option...] [patterns] [file...]`
  - There can be zero or more option arguments, and zero or more file arguments. The patterns argument contains one or more patterns separated by newlines, and is omitted when patterns are given via the ‘-e patterns’ or ‘-f file’ options. Typically patterns should be quoted when grep is used in a shell command.
- `grep 'needle' haystack.txt`
  - Searches haystack.txt for "needle"s.
- `grep -r 'needle' path/to/directory`
  - Searches recursively for the word *needle* in a directory

grep - note!

- If you have quotes in your pattern, make sure to either escape OR put the whole pattern in another quotation
    - `grep '”' file or $ grep \” file`
    - `grep “'” file or $ grep \' file`

- If you have spaces in your pattern, put the whole pattern in quotes!
    - `grep 'free boba voucher' file`
- Remember that in general, put quotes around your shell command arguments if it has quotes!

Matching multiple patterns/files

(escaped) pipe to match patterns

Bracket notation to match file names

```
grep  “pattern1\|pattern2\|pattern3” {file1, file2, file3}
```

equivalent to

```
grep -E “pattern1|pattern2|pattern3” {file1, file2, file3}
```

(-E flag if you don’t want to use ‘\’)

grep - note!

- There are a lot of flags and edge cases and when-to-use-what (basic vs extended vs perl), so when stuff don't work, try googling it first!
- Example:
    - `grep (this|that) file` doesn't work ...
    - Need to use egrep (or -E flag): `grep -E (this|that) file`
    - Same issue when using braced quantifiers like {3}
- Example:
    - `grep \d file` to look for a digit doesn't work... What should I do?
    - Use `grep [[:digit:]] file` or `$grep -P '\d' file`

SED - stream editor

- Can perform find and replace on a file
- `sed 's/find/replace/g' path/to/file`
    - Prints result of replacement to the command line, leaving input untouched
    - You can use regex in find!
        - `sed ‘s/[0-9]//g’ file.txt`

- `sed -i 's/find/replace/g' path/to/file`
    - `-i` for "In place"
    - Edits the file

2 labs for 2 weeks!

ZombieLab

- Be careful with escaping correctly
- Don't forget to do `chmod +x script.sh` and add `#!/bin/bash`

Hauntlab

- Remember to put quotes around regex patterns!
- Experiment with running some commands you think will work at the command line first. Then, when you think you've got an answer that will work, transfer it to your script.