Lecture 9B: Regular Expressions

Regular Expressions

- many Unix utilities use regular expressions: grep, sed, awk, vi, perl, Tcl
- shell filename matches are not regular expressions (eg. *.c)

Regular expressions can be a daunting subject, and when they get complex they are difficult to read for non-experienced programmers. Nevertheless, they are a very powerful tool and worth practicing. You can find many interactive tools online which can help you refine your regular expressions.

Think of regular expressions as being similar to *filename expansions*, which you've already studied. For example:

```
ls -l a*
```

Returns all files and directories that begin with 'a' followed by zero-or-more other characters. Regular expressions are similar, but aren't used with the `ls` or `find` commands. They are used to search for patterns *inside* files, not filenames.

Let's use cars again for a couple examples. We've seen this before:

```
grep 'c' cars
```

> chevy nova 79 60 3000
>
> honda accord 81 30 6000
>
> toyota tercel 82 180 750
>
> chevy impala 65 85 1550
>
> ford bronco 83 25 9525

Basic Special Characters

This is returning any line with the letter 'c' in it. Using a '.' will match any single character, much like '?' does in the shell.

```
grep '.t' cars
```

> ford mustang 65 45 17000
>
> ford ltd 83 15 10500

- toyota tercel 82 180 750
- toyota rav4 08 65 12000
- chevy volt 12 20 15000

Notice now that we have returned a lot of results where any character is in front of a `t`. This includes a **space** in the third result. That is, on the third line, we are matching the pattern `t`. This can cause you problems if you aren't careful.

Using the '*' is a little bit different than using it with file expansions. With file expansion, using '*' will match zero or more of any character. With regular expressions, '*' is a *quantifier*. It is looking at the character in front of it, and it modifies that search so that we are searching for *zero or more* of that character.

For example, what do you expect to see from this command?

```
grep 'to*' cars
```

Well, if we were thinking about file expansions, we would assume that we are searching for any files or directories that have a t, an o, followed by zero or more of any character. Here, the meaning is different. We are looking for all results that have a t, followed by zero or more os. So the result we get is this:

- ford mustang 65 45 17000
- ford ltd 83 15 10500
- toyota tercel 82 180 750
- toyota rav4 08 65 12000
- chevy volt 12 20 15000

To make the '*' behave the way it does with file expansions, use '.*' instead.

```
grep 'ford.*83' cars
```

- **ford ltd 83** 15 10500

In this case, by using `.*`, grep is allowing zero or more characters of any type between 'ford' and '83.'

Ranges

Now, maybe we want to match anything with a 'ch' or a 'th.' For this, we can use square brackets, and inside put whatever characters we want to match.

```
grep '[tc]h' cars
```

- chevy nova 79 60 3000
- ford thundbd 84 10 17000

- **ch**evy impala 65 85 1550

This also works with ranges:

```
grep '[0-9]' cars
```

Let's say we want to match all numbers with two digits. You might start by trying this:

```
grep '[0-9][0-9]' cars
```

Which will give us:

- chevy nova **79 60 3000**
- ford mustang **65 45 17000**
- volvo gl **78 10**2 **9850**
- honda civic **98 11**2 **3200**
- ford ltd **83 15 10500**
- Chevy nova **80 50 3500**
- honda accord **81 30 6000**
- toyota tercel **82 18**0 **75**0
- toyota rav4 **08 65 12000**
- chevy impala **65 85 1550**
- chevy volt **12 20 15000**

This is maybe not the expected behaviour. We are matching any two digits, including those that are inside a larger number. What can be do to filter out the results we don't want? Well, we could force our regular expression to match spaces inside our pattern, but this wouldn't match with things like the newline character (`\n`). So let's use the `-w` option we learned before:

```
grep -w '[0-9][0-9]' cars
```

- chevy nova **79 60** 3000
- ford mustang **65 45** 17000
- volvo gl **78** 102 9850
- honda civic **98** 112 3200
- ford ltd **83 15** 10500
- Chevy nova **80 50** 3500
- honda accord **81 30** 6000

- toyota tercel **82** 180 750
- toyota rav4 **08 65** 12000
- chevy impala **65 85** 1550
- chevy volt **12 20** 15000

Positional Characters

We can match all lines that *start* with a character, using '^'.

```
grep '^f' cars
```

- **f**ord mustang 65 45 17000
- **f**ord ltd 83 15 10500
- **f**iat 600 65 115 450
- **f**ord thundbd 84 10 17000
- **f**erd bronco 83 25 9525

A '$' will do the same for the end of a line:

```
grep '5$' cars
```

- ferd bronco 83 25 952**5**

Notice that the ^ will have a different meaning once it's put inside square brackets. In this case, it will *exclude* the characters put inside the brackets:

```
grep '[^f]ord' cars
```

- honda ac**cord** 81 30 6000

Note that you can use '₩' to escape from a regular expression. For example, if you want to match a dash or a period, you can like this: `grep '\-.*\$' example.txt`

- Dalmatians are black**-and-white dogs that cost $**1000.

Wildcards Vs. Regex

Wildcards	Explanation	Regex
?	Match any single character	.
*	Match zero-to-many of any character	.*
N/A	Match zero-to-many of a *preceding* character	*

Wildcards	Explanation	Regex
[ ]	Match a single character in a range	[ ]
[!]	Match a single character *not* in a range	[^]
N/A	Match with beginning of a line	^
N/A	Match with end of a line	$

Extended Regular Expressions

Extended Regular Expressions are *not* on the quiz or exam. However, you might find them to be very useful in a real-world situation.

extended regular expressions are not recognized directly by grep, can use egrep or grep -E: - {num} following any character matches "num" occurrences of that character

So about that last example. (Before the bonus) `egrep "[0-9]{5}" cars`

This is going to match *5* digits exactly in the cars file. We can also match a range, here's what that looks like:

- {min, max} following any character matches "min" to "max" occurrences of that character - "max" is optional

```
egrep "[0-9]{3,4}" cars
```

This is valid. So is this:

```
egrep "[0-9]{1,}" cars
```

- + following any character denotes *one or more* occurrences of that character - same as {1,}

```
egrep '[0-9]+' cars
```

That output should match the example above. Now take a moment to work through this example:

```
egrep "^[^ ]+ +[^ ]+ +65" cars
```

- ^ indicates the start of the line.
- **[^]** indicates a character that *isn't* a space
- **[^]+** the plus sign modifies the [^] in front of it, and indicates *one or more*. So match one or more characters that aren't a space.
- ** +** indicates one or more spaces. Put those together, and then run the search. What did you expect to match?

- **?** following any character denotes *zero or one* occurrence of that character - same as {0, 1}

```
egrep "ch?e" cars
```

- **(reg-exp)** parentheses used for grouping

  ```
  egrep "^([^ ]+ +){2}65" cars
  ```

  This expands to the example above!

- **|** means OR, matches reg-exp on *either side* of the vertical bar

  ```
  egrep "ford|chevy" cars
  ```

  ```
  egrep "(ford|chevy) +[^ ]+ +65" cars
  ```

other examples of regular expressions

- `(Mr|Mrs) Smith` - match either "Mr Smith" or "Mrs Smith"
- `Mrs? Smith` - match either "Mr Smith" or "Mrs Smith"
- `[a-zA-Z]+` - match one or more letters
- `^[a-zA-Z]\*$` - match lines with only letters
- `[^0-9]+` - match string not containing digits
- `[+-]?([0-9]+[.]?[0-9]*|[.][0-9]+)([eE][+-]?[0-9]+)?` - match valid "C" programming numbers

Summary

- regular expressions
  - `.` : match one character
  - `[ ]` : match one of whatever is in the brackets. Works with individual characters or ranges.
  - `[^ ]` : *excludes* one of whatever is in the brackets. (Inverts the search).
  - `^` : Indicates the beginning of a line.
  - `$` : Indicates the end of a line.
  - `*` : Matches zero or more of whatever precedes it.

Exercises

1. Write a regex to match both 'color' and 'colour'.

colou*r

2. Write a regex to catch either 'cash' or 'catch' in *all* files in the current directory.

grep -r cat*[cs]h .

3. Return all the cars in `cars` that *aren't* made by Ford.

grep -iv 'ford' cars

4. Catch all the amounts that start with $ and end with . and then two numbers.

₩$[0-9]*₩.[0-9][0-9]

BONUS

Catch all the things in html flags.

(<.*>.*</.*> But this doesn't include line terminators!)