Regular Expression - Intro

- Patterns that define a set of strings (or, pieces of a string)
- *Not* wildcards (similar notion, but different thing)
- Used by utilities such as *vim*, *emacs*, *grep*, *sed*, *ed*, *tr*, *perl*, *awk*, *etc.*
  - Note, the primitives for each of these vary slightly. I can't keep them all straight either. We experiment. Or look it up.
Unix Syntax for Regular Expressions

Many Unix commands (grep, egrep, awk, editors) use regular expressions for denoting patterns. The notation is similar amongst commands, though there are a few differences (see man pages).

It pays to get comfortable using regular expressions (see examples at the end).
Regular Expressions
(decreasing order of precedence)

- c     any non-special character matches itself
- ^     beginning of line
- $     end of line
- .     any single character
- [...] any one of the characters in …; ranges like a-z are legal
- [^...] any single character not in …; ranges are legal
- r*    zero or more occurrences of regular expression r
- r+    one or more occurrences of regular expression r
- r1r2  regular expressions r1 followed by r2
- r1|r2  regular expressions r1 or r2
- ( r)   regular expression r. Used for grouping. Can be nested
- \     the escape character (makes special characters normal, or, in some utilities, gives special meaning to normal characters)

No regular expression matches a new line
Simple Patterns - literals

- Fundamental building block is the single literal character
- A literal string matches itself
- E.g.
  - Consider this simple input file:
    
    You see your cat
    pass you in your car
    He waves to you
Example – simple patterns

- grep prints lines that contain a string matched by the regular expression
- I'll use egrep here, because the primitives don't need to be quoted

```
$ egrep you regexp-input
pass by in your car
He waves to you

$ egrep pass regexp-input
pass by in your car
```
Any character

Use the dot . to match any single character

'.at' matches 'bat', 'cat', 'rat', but not 'at'
There are three operators used to build regular expressions. Let $R$ and $S$ be regular expressions and $L(R)$ the set of strings that match $R$.

- **Union**
  
  $R \cup S$
  
  $L(R \cup S) = L(R) \cup L(S)$

- **Concatenation**
  
  $RS$
  
  $L(RS) = \{rs, r \in R \text{ and } s \in S\}$

- **Closure**
  
  $R^*$
  
  $L(R^*) = \{\varepsilon, R, RR, RRR, \ldots\}$
To get any line that contains "by" or "waves" (the single quote protect the | from being interpreted by the shell)

$ egrep 'by|waves' regexp-input

pass by in your car

He waves to you

Equivalent:

$ egrep '(by)|waves)' regexp-input
[] – character classes

-match any *single* character in the brackets:
  - '[Yy]ou' matches 'you' or 'You'
  - '[brc]at' matches 'cat', 'bat', or 'rat, but *not* 'at', 'hat', 'Bat'

- Ranges work fine:
  - '0x[0-7]' matches 0x3, 0x5, 0x0, but *not* 0x8
[^] – invert class

If ^ is the first character after the [, then the entire expression matches any single character not listed. Ranges still work.

- '[^rbc]' matches "hat", "zat", "Rat", but not "rat", "cat", "bat"
Pre-defined character classes

The following work in some contexts, and have analogs in other contexts

\d – any digit
\w – word character (alphanumeric or _)
\s – whitespace
\D – any non-digit
\W – any non-word character
\S – any non-whitespace
Closure

* after a regular expression, means zero or more:
  - 'ba*t' matches "bt", "bat", "baat", "baat", etc.
  - '(ba)*t' matches "t", "bat", "babat", etc.
  - '[_a-zA-Z][_a-zA-Z0-9]*' describes any legal C identifier

+ means one or more

? means zero or one
Anchors – line

- ^,\$ - match the beginning (end) of a line

$ egrep '\[yY\]ou' regexp-input
You see the cat
pass by in your car
He waves to you

$ egrep '\^[yY]ou' regexp-input
You see the cat

$ egrep '\[yY\]ou\$' regexp-input
He waves to you
Anchors – word

Use \< and \> to match the beginning and/or end of a word:

$ egrep '\<[yY]ou\>' regexp-input

You see the cat

He waves to you
Each utility handles slightly different flavors of regular expressions.
Some treat certain characters as special, while others might want them quoted to get special behavior.
Vi (vim), e.g., has "magic" and "no-magic"
sed is much like vim
grep takes regular expression, and extended regular expressions
Perl has added many extensions
Experiment, and RTFM.