# Regular Expressions

Based on slides from Dianna Xu

Bryn Mawr College
CS246 Programming Paradigm

## Basic Unix Commands

| | | |
|---|---|---|
| pwd | passwd | w |
| ls –a –l | cat | who |
| man | more/less | which |
| info | chmod | finger |
| cd | head | diff |
| cp | tail | wc |
| mv | find | echo |
| rm | egrep | sort |
| mkdir | rmdir | uniq |

## Unix Commands : Display Files

| | |
|---|---|
| cat report.c | {prints file on stdout, no pauses} |
| cat >newfile | {reads from stdin, writes to 'newfile'} |
| cat a1.txt a2.txt test.txt >newfile | {combine 3 files into 1} |
| more report.c | {space for next page, b to previous page, q to quit} |
| less file1 file2 | {:n – go to the next file :p – go to the previous file} |
| grep hello *.txt | {search *.txt files for 'hello'} |

## Regular Expressions

- A regular expression is a sequence of characters that represents a pattern.
- Describe a pattern to match, a sequence of characters, not words, within a line of text
- An expression that describes a set of strings
- Gives a concise description of the set without listing all elements
- There are usually multiple regular expressions matching the same set

## The Structure of a RegEx

- **Anchors** are used to specify the position of the pattern in relation to a line of text.
- **Character Sets** match one or more characters in a single position.
- **Modifiers** specify how many times the previous character set is repeated.

## The Anchor Characters: ^ and $

- '^' is the starting anchor and '$' is the end anchor
- If the anchor characters are not used at the proper end of the pattern, they no longer act as anchors.

| Pattern | Matches |
|---|---|
| ^A | "A" at the beginning of a line |
| A$ | "A" at the end of a line |
| A^ | "A^" anywhere on a line |
| $A | "$A" anywhere on a line |
| ^^ | "^" at the beginning of a line |
| $$ | "$" at the end of a line |

## Match Any Character with .

- Single character matches itself
- The character '.' by itself matches any character, except for the new-line character.
- Example:
  - ^.$

## Specify a Range of Characters [ ]

- Use the square brackets to identify the exact characters.
- The pattern that will match any line of text that contains exactly one number
  - ^[0123456789]$
  - ^[0-9]$
  - [A-Za-z0-9_]
- Character sets can be combined by placing them next to each other.
  - ^T[a-z][aeiou]

## Exceptions in a Character Set

| Pattern | Matches |
|---|---|
| [0-9] | Any number |
| [^0-9] | Any character other than a number |
| [-0-9] | Any number or a "-" |
| [0-9-] | Any number or a "-" |
| [^-0-9] | Any character except a number or a "-" |
| []0-9] | Any number or a "]" |
| [0-9]] | Any number followed by a "]" |
| [0-9-z] | Any number, or any character between "9" and "z". |
| [0-9\-a\]] | Any number, or a "-", a "a", or a "]" |

## Repeating Character Sets with *

- The special character '*' matches **zero or more** copies.
  - '[0-9]*' : matches zero or more numbers
  - '[0-9][0-9]*' : matches one or more numbers
  - '^#*' : matches any number of "#s" at the beginning of the line, including **zero**.
  - '^ *' :

## Named Classes

| | |
|---|---|
| [:alnum:] | Alphanumeric characters: **\w == [[:alnum:]], \W ==[^[:alnum:]]** |
| [:alpha:] | Alphabetic characters: [:lower:] and [:upper:]. |
| [:blank:] | Blank characters: space and tab. |
| [:cntrl:] | Control characters. In ASCII, these characters have octal codes 000 through 037, and 177 (`DEL'). |
| [:digit:] | 0 1 2 3 4 5 6 7 8 9 |
| [:graph:] | Graphical characters: [:alnum:] and [:punct:] |
| [:lower:] | Lower-case letters |
| [:print:] | Printable characters |
| [:punct:] | Punctuation characters |
| [:space:] | tab, newline, vertical tab, form feed, carriage return, and space |
| [:upper:] | Upper-case letters |
| [:xdigit:] | Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f |

## Alternation and Grouping

- Or – **|**
  - **`gray|grey`** → gray, grey
- Grouping – parentheses
  - **`gr(a|e)y`** → gray, grey

## Quantification

- `e?`  0 or 1 occurrence of `e`
  - `colou?r` → color , colour
- `e*`  0 or more occurrence of `e`
  - `go*gle` → ggle, gogle, google, gooogle …
- `e+`  1 or more occurrence of `e`
  - `go+gle` → gogle, google … but NOT ggle
- `e{n}`  `n` occurrences of `e`
- `e{n,}`  `n` or more occurrences of `e`
- `e{n,m}`  `n-m` occurrences of `e`

## Which Regex?

- Vowels
- No letters
- Either a or b, 1 or more times
  - b, abba, baaaba ….
- 5 consecutive lower-case letters
- All English terms for an ancestor
  - father, mother, grand father, grand mother, great grand father, great grand mother, great great grand father …

## Others

- `.`  matches any character
- `^`  matches the start of a line
- `$`  matches the end of a line
- `\< \>`  matches the beginning and the end of a word
- `\`  escapes any special characters, i.e. if you actually want to match `.`, must match `\.`

## Which Regex?

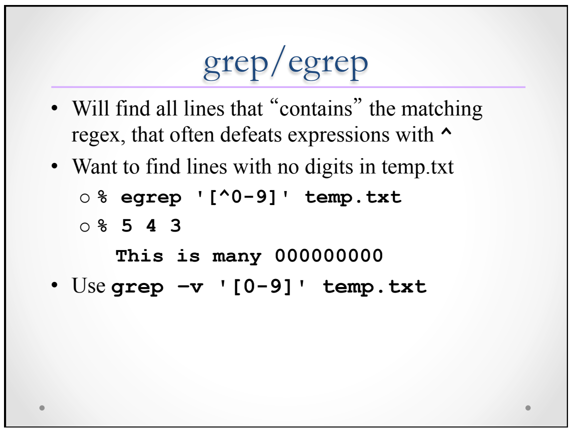
- 3 letter string that ends with "at"
- 3 letter string that ends with "at", except for "bat"
- "hat" or "cat", but only if first thing on a line
- words with no vowels
- Floating point number

## Back Reference

- `\n` matches the expression previously matched by the `n`th parenthesized subexpression
- Find all matching html title tags, h1, h2 … h6 (i.e. <h1> text </h1>)
  - `<h[1-6]>.*</h[1-6]>`
  - `<\(h[1-6]\)>.*</\1>`
  - `n` is indexed from `1`

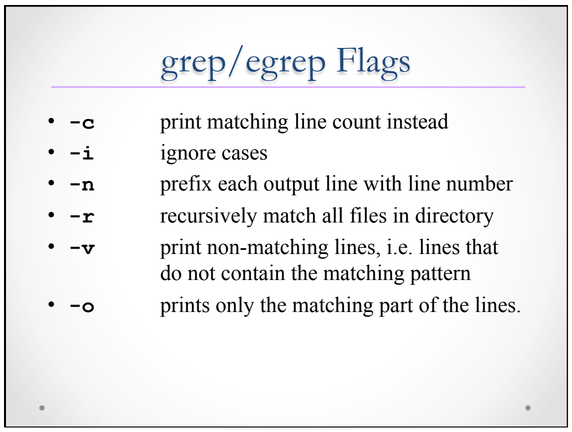
## grep, egrep and regex

- grep supports traditional Unix regex, while egrep supports full posix extended regex, and is therefore more powerful.
- grep –e is equivalent to egrep
- When giving regex at command line, must quote entire expression so that the shell will not try to parse and interpret the expression
- Use single quotes instead of double quotes



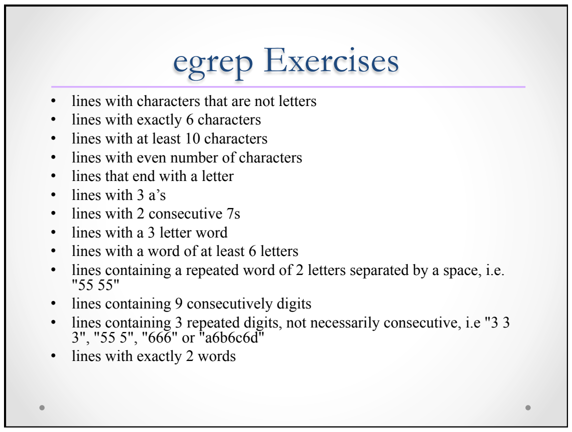
## grep/egrep

- Will find all lines that “contains” the matching regex, that often defeats expressions with **^**
- Want to find lines with no digits in temp.txt
  - `% egrep '[^0-9]' temp.txt`
  - `% 5 4 3`
    `This is many 000000000`
- Use `grep -v '[0-9]' temp.txt`



## grep/egrep Flags

- `-c` print matching line count instead
- `-i` ignore cases
- `-n` prefix each output line with line number
- `-r` recursively match all files in directory
- `-v` print non-matching lines, i.e. lines that do not contain the matching pattern
- `-o` prints only the matching part of the lines.



## egrep Exercises

- lines with characters that are not letters
- lines with exactly 6 characters
- lines with at least 10 characters
- lines with even number of characters
- lines that end with a letter
- lines with 3 a’s
- lines with 2 consecutive 7s
- lines with a 3 letter word
- lines with a word of at least 6 letters
- lines containing a repeated word of 2 letters separated by a space, i.e. "55 55"
- lines containing 9 consecutively digits
- lines containing 3 repeated digits, not necessarily consecutive, i.e "3 3 3", "55 5", "666" or "a6b6c6d"
- lines with exactly 2 words