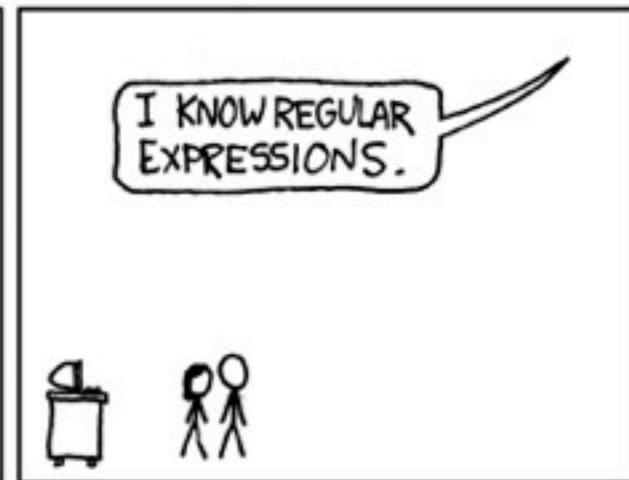
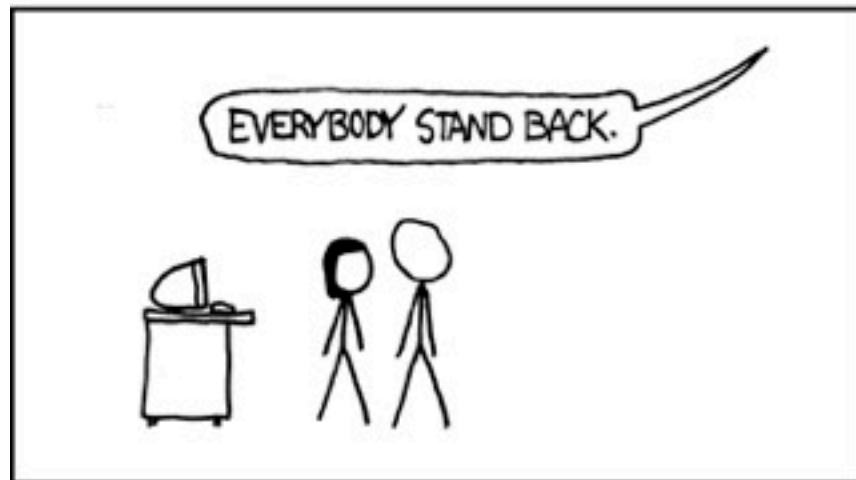


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.



Source: xkcd

# Handling Data using Regular Expressions

Lecture #4 - COMP 364  
January 11, 2010, updated 2012  
Derek Ruths

# Pipelines

---

- The | operator will take the output of a command and send it to another command
  - `curl http://en.wikipedia.org/wiki/Pipeline_(Unix) | head -n 10`
  - `cat /usr/share/dict/words | less`
  - `cat *.fasta | grep AAA`
  - tail and head can be combined together with a pipeline!

# Regular Expressions: Motivation

---

- Create a data set from a subset of a data file
  - *Extract protein interactions for one organism from the STRING database*
- Count specific items in a data set
  - *How many genes code for ribosomal proteins in the human genome?*
- Extract data from a file
  - *Get all the gene locations from an NCBI genome file*

All of these tasks involve selecting a subset of entries from a larger (textual) data set.

# Regular expressions

---

- Purpose: precisely define a class of words/character sequences that have some parts that “look the same”
  - *Words containing the string “ba”*
  - *Words that start with a capital letter*
  - *Character sequences containing only the characters A, C, T, and G*
- We have already seen simple regular expressions: wildcards (\* and ?)

# egrep: selecting lines from files

---

- Select all lines containing a specific regular expression:
  - `egrep "<regular expression>" <file to select lines from>`
- Count the number of lines containing a specific regular expression:
  - `egrep -c "<regular expression>" <file to select lines from>`
- `egrep -c "[[:alnum:]]*[ba][[:alnum:]]*" words.txt` - all words containing "ba"
- `egrep "[ACGT]{5,}" e_coli.txt` - all sequences of ACTG longer than 5 characters

# The process of writing a regular expression

---

- The process has three steps:
  1. Knowing what it is you want to match and how it might appear in the text
  2. Writing a pattern to describe what you want to match.
  3. Testing the pattern to see what it matches

# Regular Expressions (in grep)

---

- **Specifying a specific string:**
  - `egrep "hello" foo.txt` - find all lines containing the word "hello"
- **Specifying a variable position: "." and bracketed expressions ("[...]")**
  - `egrep "hs00." foo.txt` - find all lines containing the string "hs00<anything>"
  - `egrep "hs00[0123456789]" foo.txt` - find all lines containing the string "hs00#"
  - `egrep "hs00[0-9]" foo.txt` - shorter way of writing the above
  - `egrep "hs00[^0-9]" foo.txt` - find all lines containing the string "hs00<anything but a number>"
- **Exercises**
  - Find all lines in `foo.txt` containing the string "hs00<alphabetical character>"
  - Find all lines in `foo.txt` containing the string "hs00<alphanumeric characters>"

# Character classes

---

- Specified by the brackets [ ], a character class is a regular expression that **matches exactly one character**, and the possibilities for that character are specified in the brackets.
- For example, to match “What” **and** “what”, a good regular expression would be “[Ww]hat”, which says that the first character can be **either** ‘W’ or ‘w’ (but not both!) and that the following characters must be **exactly** “hat”
- Other examples: hs00[0-9], [ACGT]+, COMP[34][56]4



# Backslash magic

---

- “.” means “anything”... how do we specify that we want a period?
- “[“ is the beginning of a variable position... how do we specify that we want a left square brace?
- “-” indicates a range of characters... how do we specify that we want a dash?

When in doubt, backslash the character!

# Repetition operators

---

- ? - the preceding item is optional and matched at most once
- \* - the preceding item will be matched zero or more times
- + - the preceding item will be matched one or more times
- {n} - the preceding item is matched exactly n times
- {n,} - the preceding item is matched n or more times
- {n,m} - the preceding item is matched at least n times, but not more than m times.

# Regular expressions: specifying longer variable regions

---

- What if I wanted to find all sequences in which there were several variable positions?
  - Find all lines containing an email address:
    - `egrep "[A-Za-z0-9.-]+@[A-Za-z0-9.-]+\.[A-Za-z]+" emails.txt`
  - Find all lines containing DNA sequences longer than 5 nucleotides:
    - `egrep "[ACTG]{6,}" genome.txt`

# Grouping regular expressions

---

- Parentheses group expressions: repetition operators can act on these groups
  - $(TA)_+A\{3,\}$  = TATA box
- The “|” character indicates an “or” - either expression can match
  - $N[^P](S|T)[^P]$  = N-glycosylation site motif

# Exercises

---

- Write a regular expression for each of the following:
  - A telephone number
  - A telephone number with optional dashes
  - A telephone number with an optional extension
  - A sequence of DNA containing an exon (ensure that the coding component has a correct coding region)

# Useful shortcuts

---

- `[:digit:]` = 0-9
- `[:alnum:]` = A-Za-z0-9
- `[:alpha:]` = A-Za-z
- `[:blank:]` = tab or space
- `[:punct:]` = punctuation symbols
- `[:space:]` = any whitespace
- `[:graph:]` = anything EXCEPT whitespace
- `[:upper:]` = A-Z
- `[:lower:]` = a-z

# Exercises

---

- A telephone number
- A UNIX path
- The scientific name of an organism

# Anchors

---

- `^` = the beginning of a line
- `$` = the end of a line
- What do these regular expressions correspond to?
  - `^(Hello|Greetings) [[:upper:]][[:lower:]]+!`
  - `^(100|[1-9][[:digit:]][[:digit:]]) [[:space:]](T|F)[[:space:]][[:alpha:]]+$`