Regular Expressions

Regular expressions (regexes)

- A pattern that matches a set of strings
- Provide a (relatively) standardized way to perform matches on text
- Important to know as ***many*** tools and utilities make use of them
  - `grep`, `sed`, `find` to name a scant few
- Lots of different flavors, but they all encapsulate similar ideas
- You provide a **pattern** that is matched on the text
- The **pattern** can be a simple unassuming string or contain special characters that perform more powerful matching
- For this lecture, we'll be looking at POSIX BRE (basic regex) and ERE (extended regex)
  - `grep` is a utility that searches for patterns in a file or input via regexes
  - By default `grep` will filter out strings that don't **contain** a match
  - Defaults to BRE; `-E` flag (or `egrep`) for ERE
  - `ls /dev | grep tty`: list `/dev` directory, keeping lines that contain "tty"

Resources

- Online regex tester: https://regex101.com/ (one among many)
    - Can provide a breakdown of the regex
    - **Beware of the flavors it supports**
    - `grep` can serve as an offline tester as well
- GNU `grep`'s manual on regular expressions
- Highly detailed website: https://www.regular-expressions.info/

Regex basics

- Patterns are composed of smaller regexes that are concatenated
- The atomic regexes are those that match single characters
- The alphanumeric characters (A-Z, a-z, 0-9) and space act like normal characters
- Some other symbols act like normal characters as well
    - `hello` is a simple pattern that matches "hello"
    - `h`, `e`, `l`, `l`, and `o` are each atomic regexes
    - These are concatenated to form the overall regex `hello`

Regex basics

- There are also special functions denoted by special characters
  - `.` for any single character
  - `|` for an OR
  - `\` for special expressions/escapes
  - Quantifiers: how many to match
  - Brackets: a set of characters to match
  - Anchors: for *positional* matching
  - Backreferences: for matching a previous match
  - `^tty[0-9]+$` is a less simple pattern that matches lines that exactly compose of only "tty" and some numeric digits after it

Misc special characters

- `.` matches *any* single character
  - `...` matches three consecutive characters
- `|` for an OR between regexes
  - `hello|world` matches a string that is "hello" or "world"
- `\` for special expressions/escapes
  - `\b` matches the empty string at the edge of a "word"
  - There's more: check the GNU `grep` manual for the rest
- `(`, `)` enclose a whole expression as a *subexpression*
  - `(Hello|Goodbye) (Sowgandhi|John Paul)` matches:
  - "Hello Sowgandhi"
  - "Hello John Paul"
  - "Goodbye Sowgandhi"
  - "Goodbye John Paul"

Quantifiers

- Specify how many of a preceding regex to match
- `?`: ≤1 time
- `*`: ≥0 times
- `+`: ≥1 times
- `{n}`: *n* times
- `{n,}`: ≥*n* times
- `{,m}`: ≤*m* times
- `{n,m}`: *x* times where $n \leq x \leq m$

Examples

- `a{4}`: matches "aaaa"
- `ba+`: matches "ba", "baa", "baaa"...
- `(hello){3}`: matches "hellohellohello"

Exercise 1

- If you want to test these with `grep`, try using `grep -E`
  - Default `grep` uses BRE, which requires you to `\` escape a lot of things (more on this at the end)
- Write regexes that matches against:
  1. "hello" or "world"
  2. 20 of any character
  3. 3 of any character, "cat", then at least 5 of any character

Brackets

- `[`, `]` enclose a set to match for **one character**
  - `[abc]` matches 'a', 'b', or 'c'

Special things you can put inside them:

- `-`: range
  - `[A-Za-z0-9]`: capital and lowercase numbers and digits
- `^`: not in set
  - `[^ab]`: everything not 'a' or 'b'
- Named classes
  - `[:alnum:]`: alphanumeric characters
  - `[:alpha:]`: alphabetic characters
  - `[:blank:]`: space and tab characters
  - ...and others (see the GNU `grep` manual)
  - Brackets are part of the class name: e.g. `[[:alnum:]]` to match alphanumerics

Exercise 2

- Write regexes that matches against:
    1. 3 English vowels (a, e, i , o, u) in a row
    2. 5 non-numbers in a row
    3. "Odd" and a single digit odd number
    4. "Even" and an even number

Anchors

- Perform *positional* matching
- `^`: match empty string at the beginning of a line
  - i.e. following regex must be at the beginning
  - `^hello`: "hello" must be at the beginning
- `$`: match empty string at the end of a line
  - i.e. preceding regex must be at the end
  - `world$`: "world" must be at the end
- `^hello world$`: **entire line** must be "hello world"
  - Suppose I have a string "hello world!"
  - `hello` would be able to match against the "hello" in "**hello** world!"
  - `^hello$` would be unable to match because "hello" is not at the end of the string
- There are other non-anchor positional matches
  - `\w`, `\b` and others: look up the other `\` regexes

Exercise 3

- Write regexes that matches against:
    1. File names that end in ".txt"
    2. File names that start with "file" with an odd number after and end in ".txt"

Backreferences

- Match previous parenthesized `()` subexpression
- `\n`: match *n* th parenthesized subexpression
    - `(123)testing\1` matches "123testing123"

Q: `<([[:alpha:]][[:alnum:]]*[^>])>.*</\1>`

- Match (simple) HTML/XML tags

Caveats

- GNU `grep` defaults to BRE flavor
  - Use `-E` flag or use `egrep` for ERE flavor
  - In ERE mode, use `[{]` to capture literal '{' for portability
- Other flavors may require escaping certain characters

BRE vs ERE

- In BRE `?`, `+`, `{`, `|`, `(`, and `)` must be escaped with `\`

Any other questions?