What are Regular Expressions?

Regex is a language for describing patterns in strings.

What are Regular Expressions?

Regex is a language for describing patterns in strings.
Use regex for:

- Finding needles in haystacks.
- Changing one string to another.
- Pulling data out of strings.

Looking for stuff with `grep`

- `grep`: Global Regular Expression Print

Looking for stuff with `grep`

- `grep`: Global Regular Expression Print
- `grep 'REGEX' FILES`: Search `FILES` for `REGEX` and print matches.
- If you don't specify `FILES`, `grep` will read `STDIN` (so you can pipe stuff into it).

Looking for stuff with `grep`

- `grep`: Global Regular Expression Print
- `grep 'REGEX' FILES`: Search `FILES` for `REGEX` and print matches.
- If you don't specify `FILES`, `grep` will read `STDIN` (so you can pipe stuff into it).
- `-C LINES` gives `LINES` lines of context around the match.
- `-v` prints every line that doesn't match (invert).
- `-i` Ignore case when matching.
- `-P` Use Perl-style regular expressions.
- `-o` Only print the part of the line the regex matches.

Basic Patterns

- `.` Matches one of any character.
- `\w` Matches a word character (letters, numbers, and _).
- `\W` Matches everything `\w` doesn't.
- `\d` Matches a digit.
- `\D` Matches anything that isn't a digit.
- `\s` Matches whitespace (space, tab, newline, carriage return, etc.).
- `\S` Matches non-whitespace (everything `\s` doesn't match).
- `\` is also the escape character.

Variable-length Patterns

- `{n}` matches n of the previous character.
- `{n,m}` matches between n and m of the previous character (inclusive).
- `{n,}` matches at least n of the previous character.

Variable-length Patterns

- `{n}` matches *n* of the previous character.
- `{n,m}` matches between *n* and *m* of the previous character (inclusive).
- `{n,}` matches at least *n* of the previous character.
- `*` matches 0 or more of the previous character (`{0,}`).
- `+` matches 1 or more of the previous character (`{1,}`).
- `?` matches 0 or 1 of the previous character (`{0,1}`).

DIY character classes

- `[abc\d]` matches a character that is either a, b, c, or a digit.
- `[a-z]` matches characters between a and z.
- `^` negates a character class: `[^abc]` matches everthing except a, b, and c.

Anchors

- `^` forces the pattern to start matching at the beginning of the line.
- `$` forces the pattern to finish matching at the end of the line.
- `\b` forces the next character to be a word boundary.
- `\B` forces the next character to not be a word boundary.

Groups

- `(ab|c)` matches either ‘ab’ or ‘c’.
- You can use length modifiers on groups, too: `(abc)+` matches one or more ‘abc’
- The real power of grouping is backreferences. You can refer to the thing matched by the 1st group, etc.
- For example, `(ab|cd)\1` matches ‘abab’ or ‘cdcd’ but not ‘abcd’ or ‘cdab’.

Greedy vs. Polite matching

- Regular expressions are greedy by default: they match as much as they possibly can.
- Usually this is what you want, but sometimes it isn't.
- You can make a variable-length match non-greedy by putting a `?` after it.
- For example: `.+\.` vs. `.+?\.`.

Sed: Editing with regex

- `sed` is a stream editor-use it for editing files or STDIN.
- It uses regular expressions to perform edits to text.
- `-r` enables extended regular expressions.
- `-n` makes `sed` only print the lines it matches.

The Print Command

- `sed -n '/regex/ p'` works pretty much exactly like `grep`.
- Use this to make sure your regexes are matching what you want them to.
- (You can also use `p` in conjunction with `s`, which we'll talk about immediately.)

The Substitute Command

- `s/PATTERN/REPLACEMENT/` replaces the thing matched by PATTERN with REPLACEMENT.
- Patterns can be any regular expression that we've talked about so far.
- Replacements can be plain text and/or backreferences!

The Substitute Command

- `s/PATTERN/REPLACEMENT/` replaces the thing matched by PATTERN with REPLACEMENT.
- Patterns can be any regular expression that we've talked about so far.
- Replacements can be plain text and/or backreferences!
- `s/ / /g` makes the substitution global (every match on each line).
- `s/ / /i` makes the match case-insensitive.