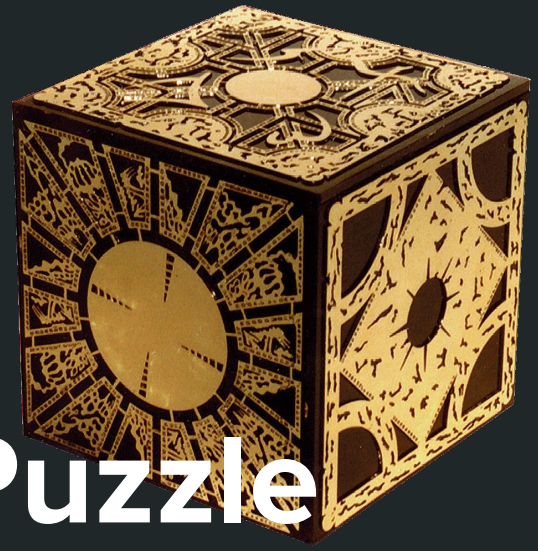


Solving the Regex Puzzle

- or, Finding the Fun in Regular Expressions

Who Am I?

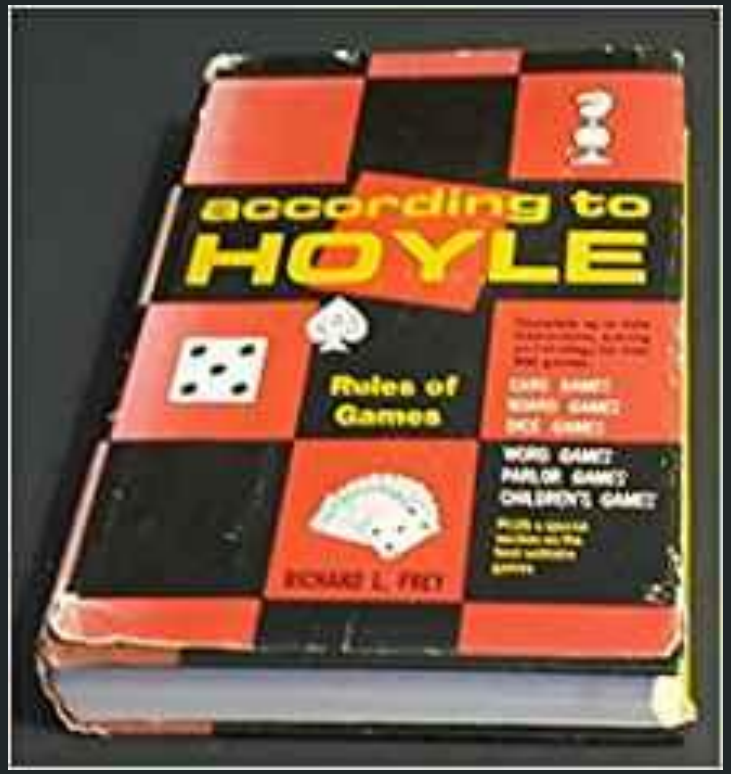
- GNU/Linux hobbyist (occasional zealot) nearly 20 years
- Unix-y professional nearly 10 years
- Language nerd
  - Double majored in German and Theological Languages (Koine Greek, Biblical Hebrew, Latin)
  - Dabble in just about any programming language
- Musician

How Will We Learn to Enjoy Regex?

1) Learn the Rules of the Game
2) Learn Composition Strategies
3) Explore the Unique POSIX Utilities
4) Play Around With Real Examples

Rules of the Game

1) POSIX is enough!
2) Read the fine manual
    a) Basic vs Extended regex
3) Characters
4) Repeaters
5) Anchors
6) Groups



Rules of the Game

POSIX is enough!

POSIX regular expressions are practically ubiquitous

Perl-compatible regular expressions (PCRE) are effectively a superset of POSIX, and a dramatically more complicated superset!

PCRE satisfies a need, but learn to walk before you run

Rules of the Game

Read the Fine Manual

Human Readable Manual:

man grep

Engineer Readable Manual:

man 7 regex

Rules of the Game

Read the Fine Manual

Since 1992, POSIX.2 defines both basic and extended regex. That distinction will be important.

Rules of the Game

Characters

Strictly speaking, we take each atom at a time. In practice, an atom is a character (group), optionally iterated, and optionally bound. Characters can be:

- Explicit, i.e. the character itself
- Any, i.e. .
- Bracketed, i.e. a set of characters
- Classed

Rules of the Game

Characters

Bracketed characters can be explicit sets or ranges. Negation occurs with the carat (^).

For example:

- [A-Za-z0-9]
- [aeiou]
- [^eq8-]

Rules of the Game

Character classes are technically bracketed characters, defined in man 3 wctype.

- Personally, I find them finicky and avoid them unless they'll be a lifesaver.
    - But I've been using basic regex. YMMV.
- Based on locale
- Syntax is [[:space:]] and not [:space:]

Rules of the Game

Notable character classes:

- blank: space or tab
- graph: printable character EXCLUDING space
- print: printable character INCLUDING space
- punct: printable character EXCLUDING alnum
- space: whitespace, including newline variants
- xdigit: hexadecimal

Rules of the Game

Repeaters

Repetition operators, which take effect on preceding atom:

- ?: occurs once or not at all
- *: occurs zero or more times
- +: occurs one or more times
- {n}: occurs n times exactly
- {n,m}: occurs n to m times

Implementations may offer variations. For instance, GNU.

Rules of the Game

Repeaters

In basic mode, some repetition operators* need to be escaped with a backslash. All the more reason to favor extended mode!

\? \+ \{n\} \{n,m\}

Spoilers, parantheses also require the backslash in basic mode:

\(sub-expression\)

Rules of the Game

Anchors align to positions in the line

- ^: beginning of the line
- $: end of the line

Rules of the Game

Groups (or “sub-expressions”) are identified within parantheses, which can be nested. All the aforementioned rules are available within the group, with the addition of the alternation pipe | (or “logical OR” as I remember it)

- (she|he|they)
- ([[:digit:]]{1,3}\.?){4}

Rules of the Game

Groups can be back-referenced positionally. While the manual page recommends against this, it can be a powerful technique.

```
echo "Hadley, Jonathan" | sed -E 's/(.*), (.*)/Hi, \2 \1/'
```

Like all powerful techniques, however, it easily becomes difficult to maintain, so it is wise to use it sparingly.

Composition Strategies

This is where the puzzles begin

- Character by character
- Group by group
- Location! Location! Location!

Composition Strategies

Character by Character

- Consider each character in sequence
- Select the most restrictive character possible
  - Case, character class, explicit set
- Consider possible permutations for the given character
  - Might the first letter be capitalized?
  - Could there be zero padding?
  - Whitespace?

Composition Strategies

Group by Group

- What makes the target unique?
  - Sub-/patterns within the target
  - Regular number of similar characters
- Limited number of variations?
  - Identify least common denominators on those variations

Composition Strategies

(Location!){3}

- Line anchors can be a valuable shortcut
- If the target is challenging to isolate, more regular surrounding text can be leveraged
  - Form data like XML or JSON with name/value pairs or other regular field separation
  - Retrieve the targeted data by back-reference
- Reduce target “surface” by limiting the region
  - Common utilities offer addressing or line ranges

POSIX Utilities

Core tools for regular expression

- grep
- sed
- awk

grep < sed < awk

POSIX Utilities

grep

“Print lines that match patterns”

- The most Unix Way™ of the regex utilities
- Favor egrep (or grep -E) to utilize extended mode

POSIX Utilities

grep

Helpful core options:

- -e additional patterns
- -o print only what matches
- -v invert match

POSIX Utilities

grep

Options to “reduce target ‘surface’”:

-A n print n lines after pattern match

-B n print n lines before pattern match

-C n print n lines circling pattern match

POSIX Utilities

sed

“Anything grep can do, sed can do better”

Equivalent to grep: sed -n ‘/RE/p’

Beyond the text replacement for which sed is (almost exclusively) famous, addresses are a powerful tool, allowing for acutely targeted transformations! One rarely, if ever, needs to pipe grep into sed

POSIX Utilities

sed addressing:

- Line numbers — `sed '5,10 s/aggravated/excited!/'`
- /regex/ — `sed -E '/#/ s/#[[:blank:]]?XXX/#/'`
- first~step — `sed '0~2 a\Add third line pattern'`
- $ (last line) — `sed '$ a\Copyright 2019 -jrh'`

POSIX Utilities

Basic vs. Extended and sed

- All implementations support basic mode
- Likely all modern implementations support some of extended mode with the option -E (GNU certainly does)
- Unless extended mode is enabled, some special characters need backslashes!
  sed ‘s:\([0-9]\{3\}\):\1.\1.\1:’ vs sed -E ‘s:([0-9){3}):\1.\1.\1:’

POSIX Utilities

awk

“Anything sed can do, awk can do… awkwardly?”

Equivalent to grep: awk ‘/RE/{print}’

The two-fold strength of awk is its flexibility with field separators and contextual action. This is useful well beyond rigidly structured data!

It’s really a language in itself, which is its strength and weakness

POSIX Utilities

awk

“Pattern scanning and processing language”

pattern { action }

Patterns can be:

- Regex against whole string ($0, the default target)
- Regex against targeted field
- Explicit expressions, e.g. $3==’Jonathan’

POSIX Utilities

Pattern examples:

```
awk ‘/10\./ {print}’

awk ‘$2~/10\./ {print}’

awk ‘$3==”Susie” {print}’

awk ‘$3!=”Susie” {print}’
```

POSIX Utilities

awk

There is an entire language of actions available, all of which may be appropriate to a given situation.

For our purposes today, the behaviors of {print} will be the focus, because it follows the uses we've been exploring and accessibly introduces the available syntax.

POSIX Utilities

For actions, whitespace is optional - likely helpful to keep code readable, but not required for syntax.

The following both work:

```
awk ‘$2 == “jhadley” { print $3 $4, $6 }’

awk ‘$2==”jhadley”{print$3$4,$6}’
```

```
awk ‘$2==”jhadley”{print$3$4,$6}’
```

Here, the comma indicates an output field separator (OFS), which is a space by default. Without it, all output would follow without any separation. OFS can be defined, as well:

```
awk ‘BEGIN{OFS=”:”}$2==”jhadley”{print$3$4,$6}’
```

POSIX Utilities

Changing the input field separator (FS) grants a lot of flexibility with awk!

It also supports explicit strings and extended regex; which lets us leverage our location/position compositional strategies.

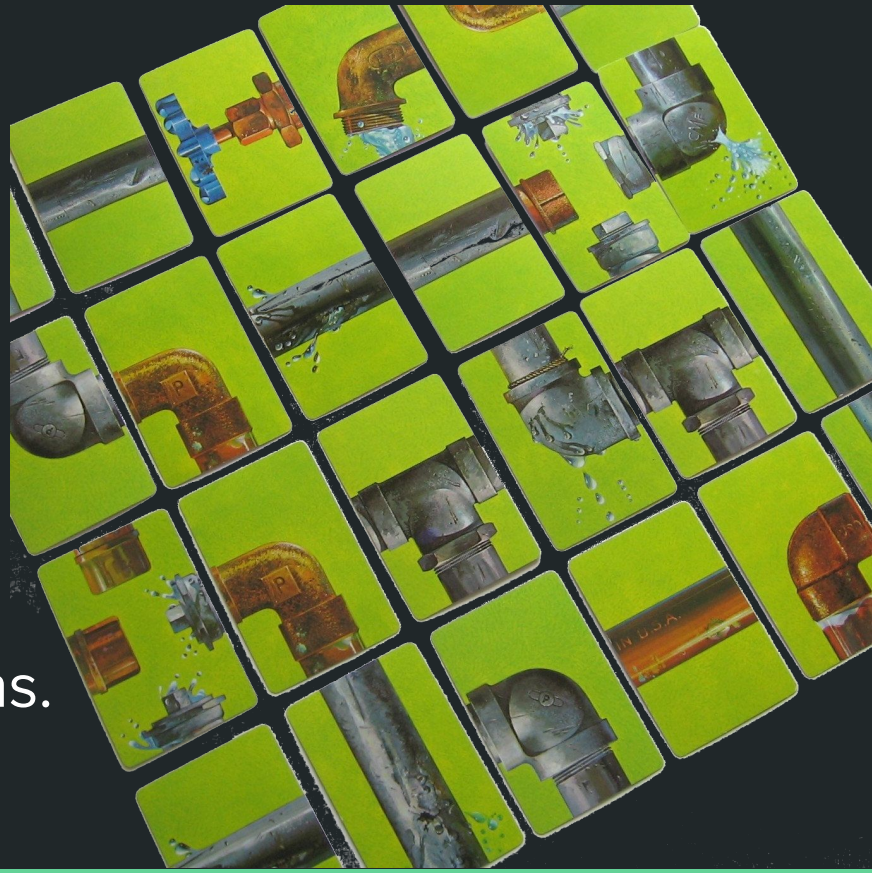
```
awk -F: '$1~/d$/{print$1,$7}' /etc/passwd
awk -F': ?' '{print$2}' some.json
```

Synthesis

Playing with the available utilities, piping them one into another, we have a professional puzzle game we get to play at work!

This is where creativity comes in.

This is where we *engineer* solutions.



Synthesis

Examples - Apache log

10.185.248.71 - - [09/Jan/2015:19:12:06 +0000] 808840 "GET /inventoryService/inventory/purchaseItem?userId=20253471&itemId=23434300 HTTP/1.1" 500 17 "-" "Apache-HttpClient/4.2.6 (java 1.5)"

Synthesis

Examples - ip output

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp4s0f1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 1c:b7:2c:33:44:1f brd ff:ff:ff:ff:ff:ff
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen
1000
    link/ether 40:e2:30:d9:fc:a7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.111/24 brd 192.168.0.255 scope global noprefixroute wlp3s0
       valid_lft forever preferred_lft forever
    inet6 fe80::42e2:30ff:fed9:fca7/64 scope link
       valid_lft forever preferred_lft forever
```