Goodcheck

CHEATSHEET

Command line

```
$ goodcheck init        # Generate configuration file
$ vim goodcheck.yml     # Edit configuration file to add rules
$ goodcheck check       # Run check in current directory
```

Configuration file

```
rules:
  - id: css_no_px                  # Name of the rule (required)
    pattern: px                    # Pattern to detect
    message: Don't use `px` units  # Message to show if pattern detected (required)
    glob: "**/*.css"               # Check .css file contents
```

Pattern syntax

String pattern

String pattern matches exact text given to the pattern.

```
pattern:
  - Github
```

Literal pattern

Literal pattern matches exactly to the given pattern. This pattern is suitable when you want to detect exactly one word from the source code.

```
pattern:
  - literal: Github
    case_sensitive: false
    # Optional: true when omitted.
```

Token pattern

Token pattern matches the sequence of given tokens. This pattern is suitable when you want to detect a sequence of specific words from the source code. Spaces can be inserted between tokens.

```
pattern:
  - token: dangerouslySetInnerHTML={
    case_sensitive: false
    # Optional: true when omitted.
```

The pattern matches with the following:

```
<div dangerouslySetInnerHTML={object} />
<div dangerouslySetInnerHTML =
                            {object} />
```

However, does not match with the following:

```
<div dangerouslySetInnerHTML= />
```

Regexp pattern

Regexp pattern allows writing any regular expression. This is the last resort, you should try the literal and token pattern first.

```
pattern:
  - regexp: margin-(left|right)
    case_sensitive: false
    # Optional: true when omitted.
    multiline: true
    # Optional: false when omitted.
```

Here is a list of frequently used regexp meta characters. The regexp of Goodcheck is based on Ruby's regular expression, you can find the detail on the Internet.

`.`	Any single character
`?`	One or zero repeating of the preceding pattern
`*`	Zero or more repeating of the preceding pattern
`()`	Grouping
`|`	Alternation of two patterns
`[123]`	1, 2, or 3
`[A-Z]`	A, B, C, ..., or Z
`\b`	Word boundary
`^` `$`	Beginning / end of line

Examples

```
\bbackground-(color|image)\b
```

background-color or background-image

```
[1-9][0-9]*px
```

One or more digits followed by px

Multiple patterns

You can have an array of pattern objects where it detects any of the given patterns.

```
rules:
  - id: gafa
    pattern:
      - Google
      - Apple
      - Facebook
      - Amazon
    message: Google, Apple, Facebook, and Amazon
```

Writing tests

You can add `pass` and `fail` attributes in each rule to run tests.

```
rules:
  - id: dangerous
    pattern:
      - token: dangerouslySetInnerHTML={
    message: React dangerouslySetInnerHTML
    pass:
      - <div undangerouslySetInnerHTML={object} />
      - <div dangerouslySetInnerHTML=true />
    fail:
      - <div dangerouslySetInnerHTML = {object} />
```

You can check if the `pass` examples don't match the given patterns, and if the `fail` examples match the given patterns.

```
$ goodcheck test
```

Justifications

Some of the rules can't be precise enough. The rules will print a lot of false positives, but you can help the reviewer identify if it's a false positive or not by using the `justification` attribute.

```
rules:
  - id: localStorage
    pattern:
      - token: localStorage
    message: localStorage may cause an exception
    justification:
      - When you catch the errors
```

Not pattern

You can write a rule with the not attribute to detect if a file does not contain a pattern.

```
rules:
  - id: no-encoding
    not:
      pattern:
        - "# coding: euc-jp"
    message: Specify file encoding through magic comment
    glob: "**/*.rb"
```

Variable Token pattern

Easily capture deny and allow list.

Deny list:

A token pattern that does not allow the listed substrings.

```
- token: ${size:number}${unit:word}
  where:
    size: true
    unit:
      - px
      - pt
      - cm
      - mm
      - in
```

Allow list:

A token pattern that only allows the listed substrings.

```
- token: "background-color: ${color:word}"
  where:
    color:
      not:
        - palette.white
        - palette.gray
```

The `${<name>:<type>}` is compiled to a regexp, and the part will be captured and tested during analysis.

YAML syntax

Here are some of the YAML syntax.

```
Hyphen:
  - Means
  - a sequence.

Colon:
  means: a dictionary.

Quote:
  "for YAML special chars"

Pipe: |
  Use pipe for text
  with line breaks.

Right angle bracket: >
  Use right angle bracket when
  new lines are meant to appear
  as spaces.
```

Use quotes (" " or ' ') to use YAML special characters in rules. The most frequently used one is the colon (:).