

A *GitHub* Action to check your code with *diKTat*

Table of Contents

Features.....	1
Usage	2
Configuration.....	2
<code>config</code> : custom configuration file	2
<code>reporter</code> : requesting a custom reporter	3
<code>input-paths</code> : custom source sets	5
<code>java-distribution</code> and <code>java-version</code> : running <i>diKTat</i> using a custom JVM	6
<code>fail-on-error</code> : suppressing lint errors	6
<code>relative-paths</code> : relative or absolute paths	7
<code>color</code> : coloring the plain-text output	7
<code>debug</code> : enabling debug logging	7
Outputs	8
Using the command-line client	8

License MIT

[[GitHub release](#)] | <https://badgen.net/github/release/saveourtool/benedikt/latest?color=green>

[Ubuntu Linux] |

`https://badgen.net/badge/icon/Ubuntu?icon=terminal&label&color=green`

[macOS] | `https://badgen.net/badge/icon/macOS?icon=apple&label&color=green`

[Windows] |

`https://badgen.net/badge/icon/Windows?icon=windows&label&color=green`

□ | An always updated version of this document is available [here](#) as a PDF e-book.

Features

- Customizable `diktat-analysis.yml` [location](#). You can use the rule set configuration with an alternate name or at a non-default location.
- Customizable JVM [vendor and version](#). You can run *diKTat* using a default JVM, or you can set up your own one.
- Customizable [reporters](#) (*SARIF*, *JSON*, *Checkstyle XML*).
- Allows multiple [input paths](#). If you have a multi-module project and only wish to check certain

directories or modules, you can configure the action accordingly.

- The last line of the log output reported to the summary page:

diKTat Check summary

✗ *diKTat* exited with code 1

17:54:11.978 [main] DEBUG com.pinterest.ktlint.internal.KtlintCommandLine - 3914ms / 14 file(s) / 185 error(s)

- *diktat*: 1.2.3
- *ktlint*: 0.46.1
- *java*: 11.0.16

Job summary generated at run-time

Usage

In the simplest scenario, the action can be used without input parameters; you just need to check out your code first, using [actions/checkout](#):

```
jobs:
  diktat_check:
    name: 'diKTat Check'
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - uses: saveourtool/benedikt@v1
```

Configuration

config: custom configuration file

- Default: **diktat-analysis.yml**
- Required: **no**

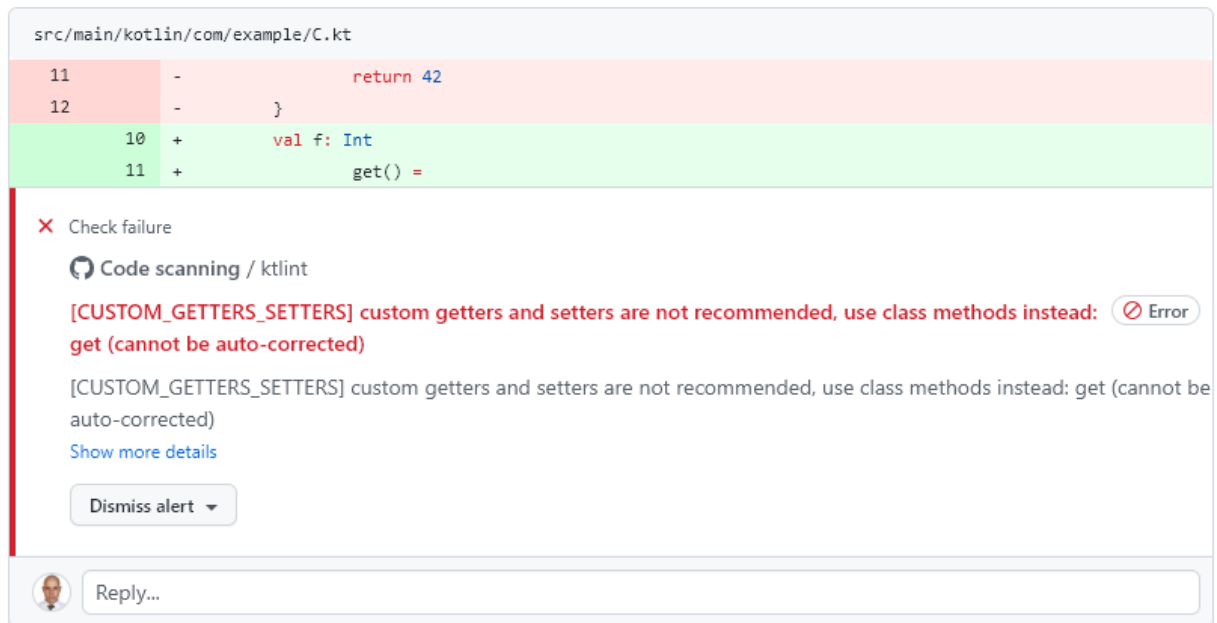
You can override the name or the path of your YAML configuration file using the **config** input parameter, e. g.:

```
- uses: saveourtool/benedikt@v1
  with:
    config: path/to/diktat-analysis-custom.yml
```

reporter: requesting a custom reporter

If you wish, you can report errors in a custom format.


- Default: `sarif`
- Required: `no`
- Possible values: any of the following.
 - `sarif`: report errors in the [SARIF](#) format. The output file will be named `report.sarif` and automatically uploaded to *GitHub* using the `upload-sarif` action. This will enable the check results to be shown as annotations in the pull request:



as well as in the *Code scanning alerts* section of your repository:

[MAGIC_NUMBER] avoid using magic numbers, instead define constants with clear names describing what the magic number means: 42 (cannot be auto-corrected)

Dismiss alert ▾

 Open in master 8 minutes ago

src/main/kotlin/com/example/C.kt:19

```

16      */
17      val g: Int
18          get() =
19              42


```

[MAGIC_NUMBER] avoid using magic numbers, instead define constants with clear names describing what the magic number means: 42 (cannot be auto-corrected)



ktlint

20 }

Severity

 Error


Affected branches


 master 

Tool Rule ID


ktlint diktat-ruleset:magic-number

No rule help available for this alert.

 First detected in commit 8 minutes ago

 Initiate a new code scanning procedure ✖ 5511f17

src/main/kotlin/com/example/C.kt:19 on branch master

 Appeared in branch master 8 minutes ago

✖ diKTat Check, Ubuntu Linux #62: Commit 5511f177


- **checkstyle**: this is the reporter of choice if you ever encounter issues with the **sarif** reporter. Errors are reported in the **Checkstyle-XML** format to the file named **checkstyle-report.xml**. The report is then consumed by the **reviewdog** tool and uploaded to **GitHub**, resulting in code annotations similar to those produced by the **sarif** reporter:

```

2
3  + /**
4  +  * This is a simple class covered by both unit and integration tests.
5  +  */
6  object C {

```

✖ Check failure on line 6 in src/main/kotlin/com/example/C.kt

 GitHub Actions / diKTat errors reported by reviewdog

[diKTat errors reported by reviewdog] src/main/kotlin/com/example/C.kt#L6 <diktat-ruleset:identifier-naming>

[OBJECT_NAME_INCORRECT] object structure name should be in PascalCase and should contain only latin (ASCII) letters or numbers

[Raw output](#)

- **plain**: report errors in the plain-text format, e. g.:

```

C.kt:1:1: [MISSING_KDOC_TOP_LEVEL] all public and internal top-level classes and
functions should have Kdoc: C (cannot be auto-corrected) (diktat-ruleset:kdoc-
comments)
C.kt:1:1: [FILE_NAME_INCORRECT] file name is incorrect - it should end with .kt
extension and be in PascalCase: C.kt (diktat-ruleset:file-naming)
C.kt:1:1: [PACKAGE_NAME_MISSING] no package name declared in a file: C.kt

```

```
(diktat-ruleset:package-naming)
C.kt:1:7: [CLASS_NAME_INCORRECT] class/enum/interface name should be in
PascalCase and should contain only latin (ASCII) letters or numbers: C (diktat-
ruleset:identifier-naming)
C.kt:1:7: [IDENTIFIER_LENGTH] identifier's length is incorrect, it should be in
range of [2, 64] symbols: C (cannot be auto-corrected) (diktat-
ruleset:identifier-naming)
```

The errors, if any, are printed on the standard output.

- **plain?group_by_file**: same as above, but group errors by file, e. g.:

```
C.kt
1:1 [MISSING_KDOC_TOP_LEVEL] all public and internal top-level classes and
functions should have Kdoc: C (cannot be auto-corrected)
1:1 [FILE_NAME_INCORRECT] file name is incorrect - it should end with .kt
extension and be in PascalCase: C.kt
1:1 [PACKAGE_NAME_MISSING] no package name declared in a file: C.kt
1:7 [CLASS_NAME_INCORRECT] class/enum/interface name should be in PascalCase
and should contain only latin (ASCII) letters or numbers: C
1:7 [IDENTIFIER_LENGTH] identifier's length is incorrect, it should be in
range of [2, 64] symbols: C (cannot be auto-corrected)
```

- **json**: report errors in the JSON format to the file named **report.json**.
- **html**: report errors in the HTML format to the file named **report.html**.

input-paths: custom source sets

- Default: none
- Required: **no**

One or more patterns which indicate the files or directories to check. Use a multiline string to specify multiple inputs.

- If an input is a path to a file, it is passed to *diKTat* as-is:

```
- uses: saveourtool/benedikt@v1
  with:
    input-paths: |
      path/to/file.kt
```

- If an input is a path to a directory, the directory is recursively traversed, and all ***.kt** and ***.kts** files are passed to *diKTat*.

```
- uses: saveourtool/benedikt@v1
  with:
```

```
input-paths: |
  src/main/kotlin
  src/test/kotlin
```

- If an input is an [Ant-style path pattern](#) (such as `**/*.kt`), *diKTat* expands it into the list of files that match the path pattern. Path patterns may be negated, e. g.: `!src/**/*.Test.kt` or `!src/**/*.generated/**`.

```
- uses: saveourtool/benedikt@v1
  with:
    input-paths: |
      **/*.kt
      **/*.kts
      !**/.generated/**
```

If this input parameter is not specified, this is equivalent to setting it to `.`, meaning *diKTat* will check all `*.kt` and `*.kts` files in the project directory unless configured otherwise.

java-distribution and java-version: running *diKTat* using a custom JVM

It's possible to run *diKTat* with a custom JVM using the `actions/setup-java` action. The following input parameters may be specified:

- **java-distribution**: the Java distribution, see the [list of supported distributions](#).
 - Default: `temurin`
 - Required: **no**
- **java-version**: the Java version to set up. Takes a whole or semver Java version. See [examples of supported syntax](#).
 - Default: `none`
 - Required: **no**

NOTE

Setting just the **java-distribution** property in order to use a custom JDK is not sufficient: you'll need to set **both** **java-distribution** **and** **java-version**:

```
- uses: saveourtool/benedikt@v1
  with:
    java-distribution: 'temurin'
    java-version: 17
```

fail-on-error: suppressing lint errors

- Default: `true`

- Required: **no**

If **false**, the errors are still reported, but the step completes successfully. If **true** (the default), then lint errors reported by *diKTat* are considered fatal (i.e. the current step terminates with a failure):

```
- uses: saveourtool/benedikt@v1
  with:
    fail-on-error: true
```

NOTE

This flag only affects the case when *diKTat* exits with code **1**. Higher [exit codes](#) are *always* fatal.

relative-paths: relative or absolute paths

- Default: **true**
- Required: **no**

If **true**, file paths get relativized with respect to the project directory. Otherwise, absolute file paths get reported. Example:

```
- uses: saveourtool/benedikt@v1
  with:
    relative-paths: true
```

NOTE

When *SARIF* [reporter](#) is used, this flag has no effect: in *SARIF* mode, paths reported are always absolute.

color: colorizing the plain-text output

- Default: **true**
- Required: **no**

Setting this flag enables the console output to be colorized. This is only useful if the [reporter](#) is set to **plain** or **plain?group_by_file**:

```
- uses: saveourtool/benedikt@v1
  with:
    reporter: plain
    color: true
```

debug: enabling debug logging

- Default: **false**

- Required: **no**

Debug logging can be enabled by setting the **debug** input parameter to **true**:

```
- uses: saveourtool/benedikt@v1
  with:
    debug: true
```

Outputs

The action returns the exit code of the command-line client using the **exit-code** output parameter, e. g.:

```
jobs:
  diktat_check:
    name: 'diKTat Check'
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - id: diktat
        uses: saveourtool/benedikt@v1

      - name: 'Read the exit code of diKTat'
        if: ${{ always() }}
        run: echo "diKTat exited with code ${ steps.diktat.outputs.exit-code }"
        shell: bash
```

The exit codes are documented [here](#).

Using the command-line client

Alternatively, if you wish to run *diKTat* locally (e. g.: as a *Vim* plug-in), or you're using a different CI/CD server, you can try the [command-line client](#).