Motivation

Making an R package

Having it all in one place means

- Easier to install it
- Easier to load it
- Easier to share it
- Easier to version it

Motivation

Making an R package

Having it all in one place means

- Easier to install it
- Easier to load it
- Easier to share it
- Easier to version it

Making it a good one

You probably will want to it to be

- Installable
- Bug-free
- Easily maintainable
- User friendly

Existing tools

- Installable → R CMD check and rcmdcheck
- Bug-free
  - Test your code → testthat
  - Test all (or at least most) of it → covr
- Easily maintainable
  - Keep your functions simple → cyclocomp
  - Keep your code readable → lintr
- User friendly → vignettes and webpages

Existing tools

- Installable → R CMD check and rcmdcheck
- Bug-free
  - Test your code → testthat
  - Test all (or at least most) of it → covr
- Easily maintainable
  - Keep your functions simple → cyclocomp
  - Keep your code readable → lintr
- User friendly → vignettes and webpages

Existing tools

- Installable → R CMD check and **rcmdcheck**
- Bug-free
  - Test your code → **testthat**
  - Test all (or at least most) of it → **covr**
- Easily maintainable
  - Keep your functions simple → **cyclocomp**
  - Keep your code readable → **lintr**
- User friendly → vignettes and webpages

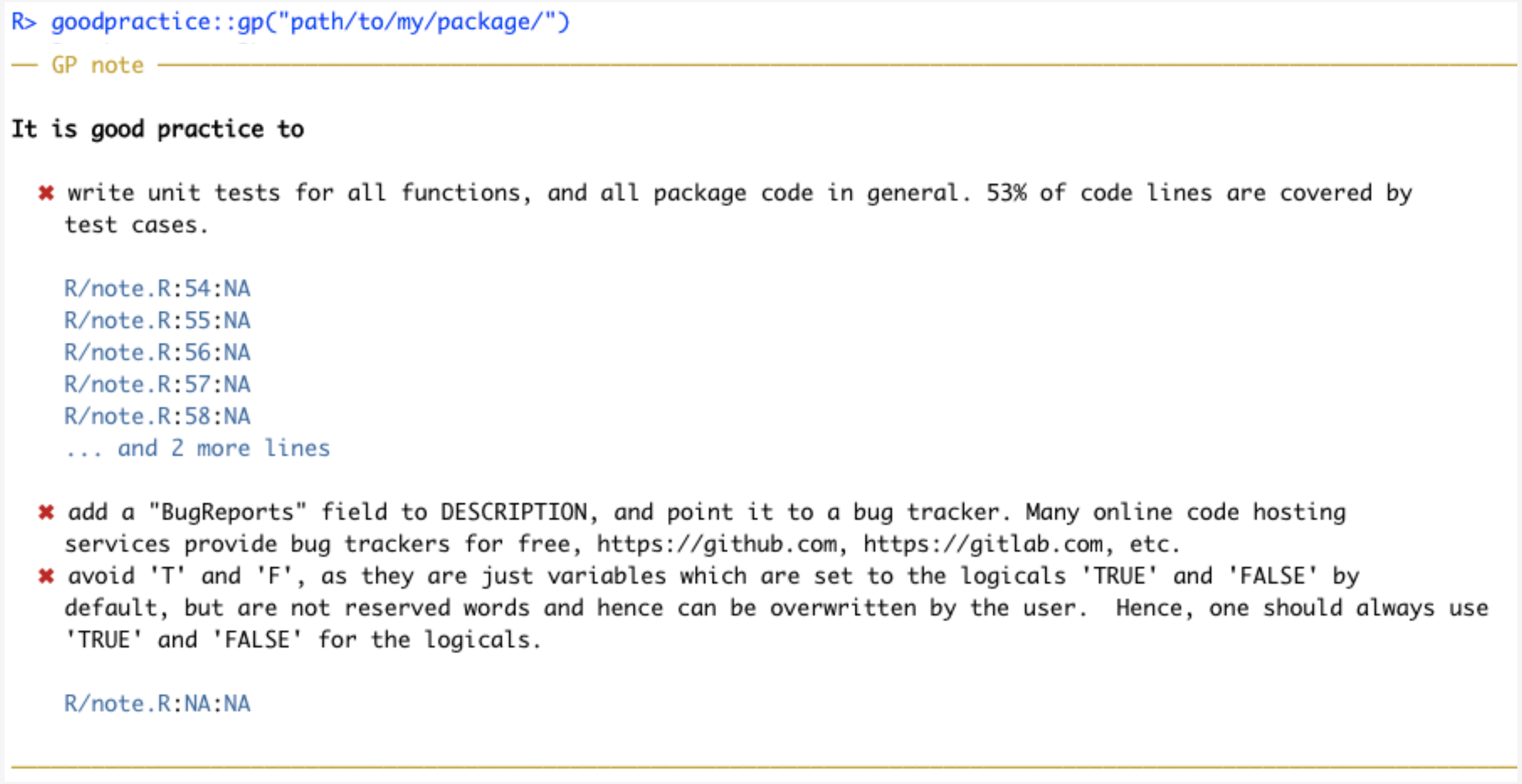
The goodpractice package

- Originally written by Gábor Csárdi
- Performs static code analysis on package source code
- Combines the afore-mentioned tools to carry out 230 checks
- Advice on what (not) to do, why and where
- All in one place which makes it easier to use and harder to forget anything
- Establish and enforce common standards across a team
- Quick check of an unknown package

The goodpractice package

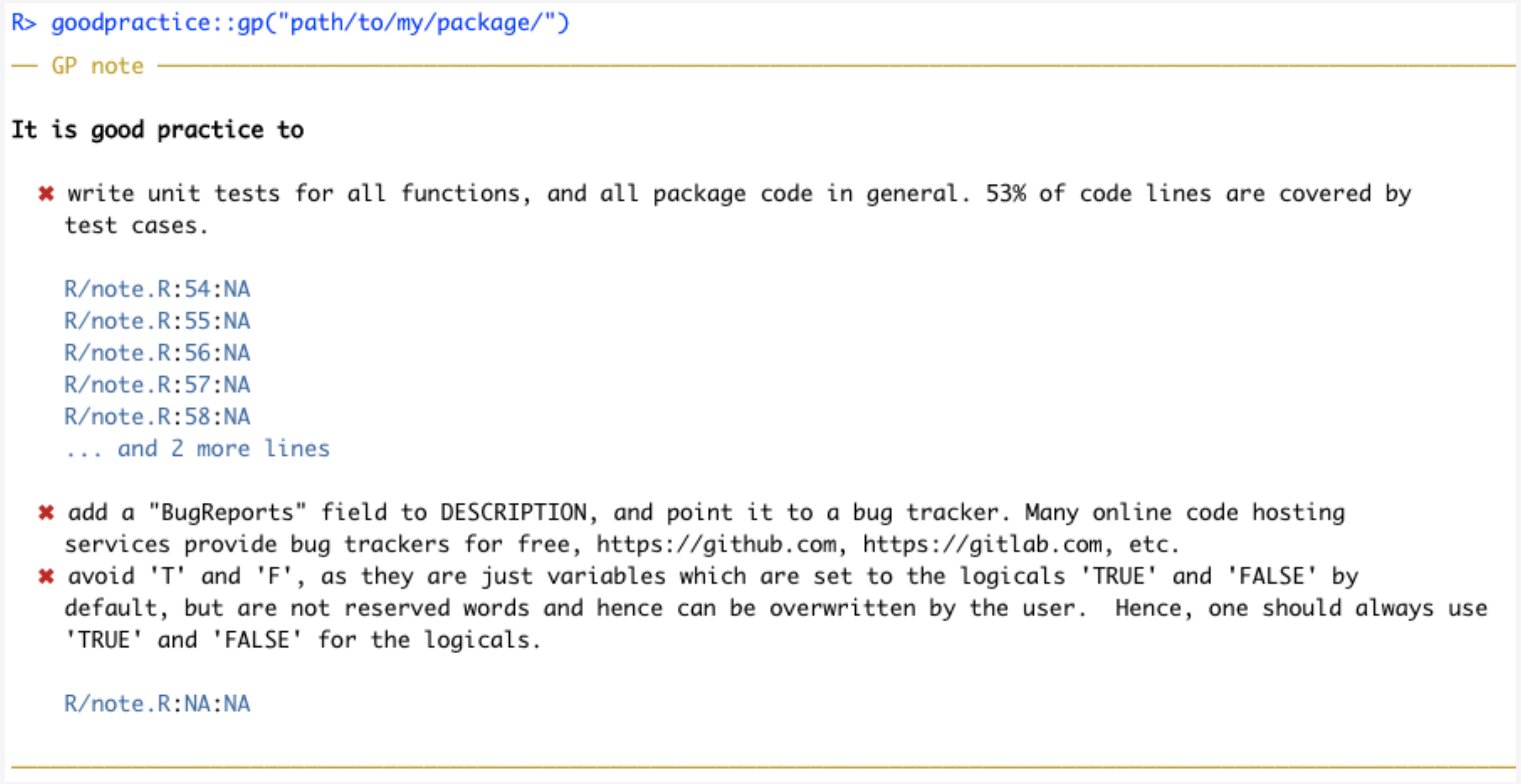
- Originally written by Gábor Csárdi
- Performs static code analysis on package source code
- Combines the afore-mentioned tools to carry out 230 checks
- Advice on what (not) to do, why and where
- All in one place which makes it easier to use and harder to forget anything
- Establish and enforce common standards across a team
- Quick check of an unknown package

goodpractice in practice



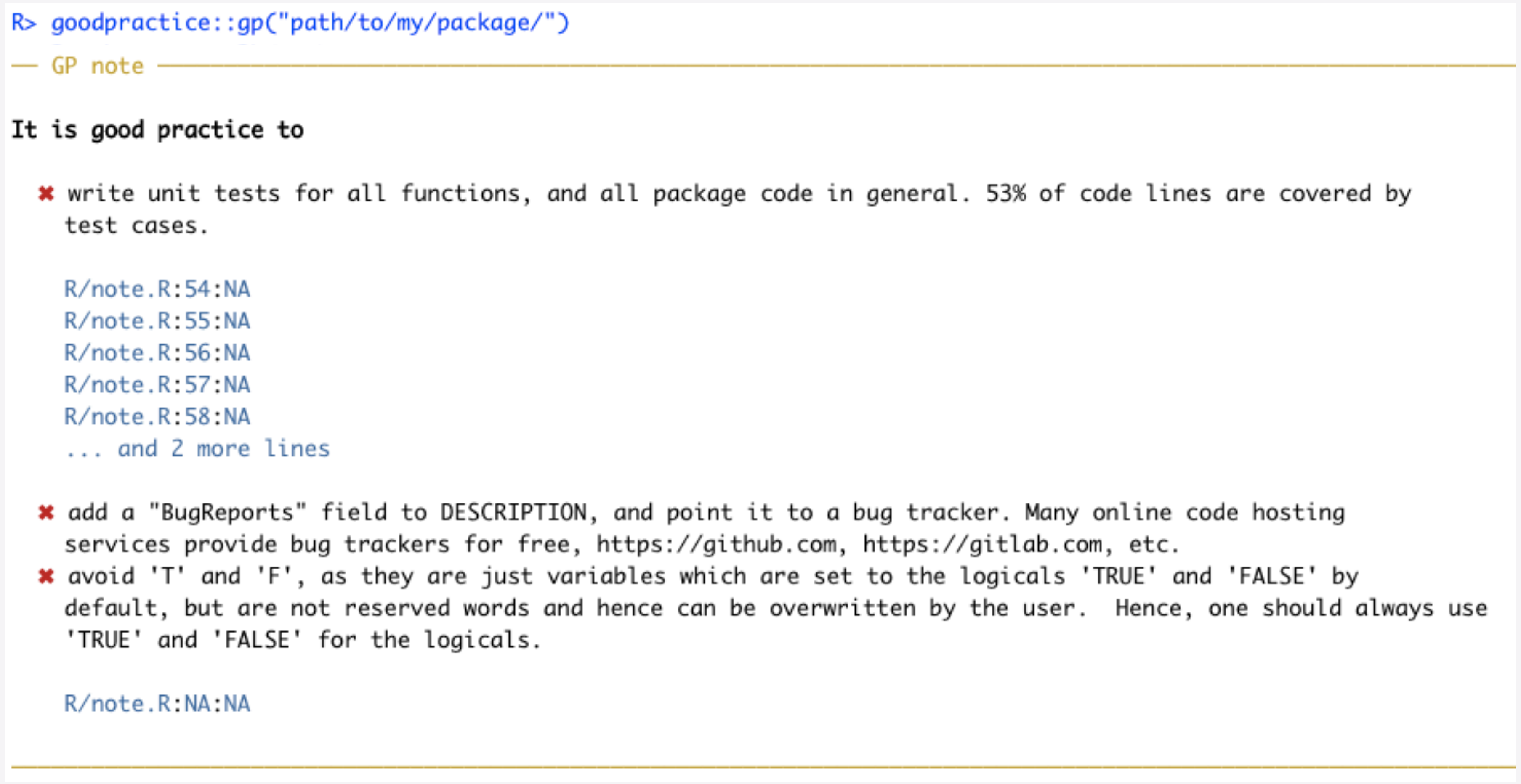
```
R> goodpractice::gp("path/to/my/package/")
── GP note ──────────────────────────────────────────────────────────────────────

It is good practice to

  ✖ write unit tests for all functions, and all package code in general. 53% of code lines are covered by
    test cases.

    R/note.R:54:NA
    R/note.R:55:NA
    R/note.R:56:NA
    R/note.R:57:NA
    R/note.R:58:NA
    ... and 2 more lines

  ✖ add a "BugReports" field to DESCRIPTION, and point it to a bug tracker. Many online code hosting
    services provide bug trackers for free, https://github.com, https://gitlab.com, etc.
  ✖ avoid 'T' and 'F', as they are just variables which are set to the logicals 'TRUE' and 'FALSE' by
    default, but are not reserved words and hence can be overwritten by the user.  Hence, one should always use
    'TRUE' and 'FALSE' for the logicals.

    R/note.R:NA:NA

─────────────────────────────────────────────────────────────────────────────────
```

goodpractice in practice



```
R> goodpractice::gp("path/to/my/package/")
── GP note ──────────────────────────────────────────────────────────────────────

It is good practice to

  ✖ write unit tests for all functions, and all package code in general. 53% of code lines are covered by
    test cases.

    R/note.R:54:NA
    R/note.R:55:NA
    R/note.R:56:NA
    R/note.R:57:NA
    R/note.R:58:NA
    ... and 2 more lines

  ✖ add a "BugReports" field to DESCRIPTION, and point it to a bug tracker. Many online code hosting
    services provide bug trackers for free, https://github.com, https://gitlab.com, etc.
  ✖ avoid 'T' and 'F', as they are just variables which are set to the logicals 'TRUE' and 'FALSE' by
    default, but are not reserved words and hence can be overwritten by the user.  Hence, one should always use
    'TRUE' and 'FALSE' for the logicals.

    R/note.R:NA:NA

─────────────────────────────────────────────────────────────────────────────────
```

goodpractice in practice



```
R> goodpractice::gp("path/to/my/package/")
── GP note ──────────────────────────────────────────────────────────────────

It is good practice to

  ✖ write unit tests for all functions, and all package code in general. 53% of code lines are covered by
    test cases.

    R/note.R:54:NA
    R/note.R:55:NA
    R/note.R:56:NA
    R/note.R:57:NA
    R/note.R:58:NA
    ... and 2 more lines

  ✖ add a "BugReports" field to DESCRIPTION, and point it to a bug tracker. Many online code hosting
    services provide bug trackers for free, https://github.com, https://gitlab.com, etc.
  ✖ avoid 'T' and 'F', as they are just variables which are set to the logicals 'TRUE' and 'FALSE' by
    default, but are not reserved words and hence can be overwritten by the user.  Hence, one should always use
    'TRUE' and 'FALSE' for the logicals.

    R/note.R:NA:NA

─────────────────────────────────────────────────────────────────────────────
```

goodpractice in practice

```
# which checks were carried out?
checks(g_url)
#> [1] "description_url"

# which checks failed?
failed_checks(g)
#> [1] "no_description_depends"
#> [2] "no_description_date"
#> [3] "description_url"
#> [4] "description_bugreports"
#> [5] "lintr_trailing_semicolon_linter"
#> [6] "no_import_package_as_a_whole"
#> [7] "rcmdcheck_package_dependencies_present"
#> [8] "truefalse_not_tf"
```

```
# show the first 5 checks carried out and their results
results(g)[1:5,]
#>                    check result
#> 1                   covr     NA
#> 2              cyclocomp   TRUE
#> 3 no_description_depends  FALSE
#> 4    no_description_date  FALSE
#> 5        description_url  FALSE
```

Customizing goodpractice

- Select checks from what's already implemented in goodpractice
- Add your own checks with `make_check()` and `make_prep()`
- Detailed examples in the *Custom Checks* vignette
- Example: the checkers package for automated checking of best practices for research compendia

Customizing goodpractice

- Select checks from what’s already implemented in goodpractice
- Add your own checks with `make_check()` and `make_prep()`
- Detailed examples in the *Custom Checks* vignette
- Example: the checkers package for automated checking of best practices for research compendia

Summary

- Performs static code analysis on package source code
- Advice on what (not) to do, why and where
- Use as a reminder for yourself or to work to common standards across a team
- Use interactively during package development or programmatically to check multiple packages
- Customise to carry out your own checks

Where to next?

🎁 Install from CRAN

```
install.packages(“goodpractice”)
```

🎈 Code

https://github.com/MangoTheCat/goodpractice

📚 Documentation

https://mangothecat.github.io/goodpractice/