**Caching**

**Conceptual**

When we created styler, it was a design principle to make it non-invasive,  
flexible and maintainable. The downside of this is that it is slow. The natural  
solution to this problem is caching. styler does not store any formatted code  
anywhere, it simply hashes the input and checks if this was the output of  
formatting before. If so, it returns the code. the package implements caching on  
two layers. Whole input and by top-level expression. Let’s say you want to style  
the below text:

1 + 1

a <- 3

If it was previously styled, the second time you style it, it will just return  
the input immediately. If you change the code to

1 + 1

a = 3

the first layer of the cache (the whole input) won’t be used, because the text  
input is not the same. However, the first expression 1 + 1 is the same, so it  
will use the cache for this expression and only run a=3 through the full  
processing engine. The bigger the cached expressions, the more this matters. And  
typically, you have

# in package code

f <- function() {

# long function declaration here

}

# in analysis code

data %>%

long() %>%

pipe(expressions) %>%

here(all = chained)

These are two top-level expressions (if you ignore the two top-level comments).  
If you style your code regularly and you do only modify a subset of all  
top-level expressions without touching the majority of them, you will benefit a  
lot from caching. In particular when you run functions like style\_pkg(),  
because all unmodified files will *pass through* very quickly.

**Usage**

Caching is enabled by default and you will be asked once to let the caching  
backend of styler (R.cache) create a cache permanent directory. This will enable  
caching across R sessions, not just within a session. The cache is shared across  
all APIs (style\_text(), style\_file(), style\_dir(), style\_pkg(), Addin).  
You can easily check details of the cache as follows:

cache\_info()

## Size: 0 bytes (0 cached expressions)

## Last modified: NA

## Created: 2020-03-14 14:53:23

## Location: /tmp/RtmpyGP7aD/.Rcache/styler/1.3.2

## Activated: TRUE

There are utilities to deactivate the cache with cache\_deactivate() or clear  
it with cache\_clear(). You can also use use multiple caches, see  
?cache\_activate(). Caches are version and style guide dependent, so when you  
update the styler package in the future, the cache will be rebuilt as you go. We  
wanted to make sure the cache does not grow large on your disk, which is the  
case with the above described approach that does not store any code. We simply  
hash the styled code and create an empty file with the hash as a name. This  
literally takes zero bytes on disk, plus the size of the inode (which is  
negligible).

If you want use styler in a CI/CD workflow or non-interactively, please see  
?caching for details.

***stylerignore***

I am personally not a big fan of this idea, but now you can make styler ignore  
some lines:

blibala= 3

# styler: off

I\_have(good+reasons, to = turn\_off,

styler

)

# styler: on

some\_call()

If you feed the above into styler, it will not change the code between the  
comments. To use something else than # styler: on and # styler: off as  
markers, set the R options styler.ignore\_start and styler.ignore\_stop using  
options(). You can also use the start marker inline on the same line as the  
expression you want to *stylerignore*.

ignore( this) # styler: off

f() # not ignored anymore

You can put markers in arbitrary places:

call(

# styler: off

1 +

# styler: on

3 ,

22

)

Also note that as of styler v1.1.1.9002, we support [alignment  
detection](https://styler.r-lib.org/articles/detect-alignment.html) for function  
calls, so styler won’t modify the below code and *stylerignore* is not  
necessary.

call(

a = 222,

ab = 2

)