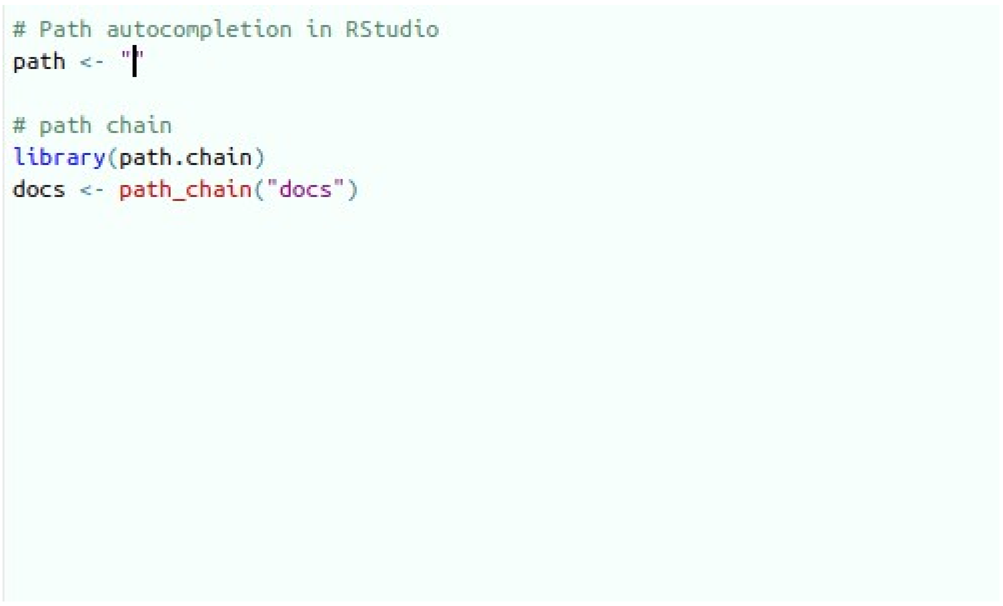
`path.chain` package provides an intuitive and easy-to-use system of nested objects, which represents different levels of some directory's structure in the file system. It allows us to

Look at the `path.chain`

Sometimes one picture can say more, than a thousand words, and this is exactly the case.



```
# Path autocompletion in RStudio
path <- ""

# path chain
library(path.chain)
docs <- path_chain("docs")
```

Motivation
———-

I've been working on the ML project, when we decided to keep strcture of the input data in the **YAML** config. The structure were getting complicated, and at some point on this history ou config get a form like this:

```
default:
  kData:
    kRoot: 'our/super/dir'
    kTerroir:
      kRoot: 'terroir'
      kSoils: 'soils.fst'
      kTemperature: 'temperature.fst'
      kRains: 'rains.fst'
    kWineQuality:
      kChemicalParams: 'chemical_params.fst'
      kContestResults: 'contest_results.fst'
```

For your infomation: the example above is totally fictitious and has nothing to do with the actual project I've been woking on. Moreover, in our project, several times more of paths were defined. As you can imagine, such structure forced us to load data in the following manner:

```
config <- config::get(
  config = "default"
  file = "path/to/config",
  use_parent = FALSE
)

path <- file.path(
  config$kData$kRoot,
  config$kData$kTerroir$kRoot,
  config$kData$kTerroir$kSoils
)

vineyard_soils <- fst::read_fst(path)
```

Doesn’t it look redundant? So, I’ve written a `path.chain` package:
using it we can perform the same action with less code:

```
library(path.chain)

vineyard_soils <- fst::read_fst(
  config$kData$kTerroir$kSoils
)
```

Isn’t it nice for your eyes?

If I would like to modify the config, say, with the following change,

```
default:
  kData:
    kRoot: 'our/super/dir'
    kTerroir:
      kRoot: 'terroir'
      kSoils: 'vineyard_soils.fst' # <- This is the change
      kTemperature: 'temperature.fst'
      kRains: 'rains.fst'
    kWineQuality:
      kChemicalParams: 'chemical_params.fst'
      kContestResults: 'contest_results.fst'
```

the code is still working.

What if we would like to reconfigure our list of paths wthout changing
the code? It may probably break desired behaviour of our scripts, but
with `path.chain` we can easily detect the cause looking into logs.
Simply use `on_path_not_exists` or `on_validate_path`

```
on_validate_path(
  ~ if(tools::file_ext(.x) == '.fst') print("Invalid file")
)

on_path_not_exists(~ log_error("Path {.x} not exists"))…
```