コーディングスタイルガイド

目次

1. はじめに

Redはコードがデータとして表現される同図象性を持つ言語です。その特性の結果として、どんな形でもデータを取り扱えるよう、言語はほぼ完全なフリーフォームとなっていて、またDSL特有のフォーマッティングのニーズを叶える柔軟性を持っています。以下のコンテンツはRedのコードをフォーマットする 膨大な方法の1つでしかありません。

このドキュメントは *Red*のソースコードの中で使用されているオフィシャルなコーディングスタイルを説明します。従ってGithubのred/redリポジトリにサブミットされる全てのプルリクエストはこのコーディングスタイルに準拠する必要があります。

Red/SystemはRedのダイアレクトであり、Redと同じシンタックス、コーディングスタイルルールを共有します。Red/System固有のルールはそのように記載がされます。

以下のルールの目的は、コメントの必要性を最小にし、スクリーン上に見えるコードの行数を最適に維持することなど、可読性を最大化することです。

2. 1行のコードの長さ

1行のコードに対して最大何文字とするかの厳格な定義はありません。使用されているフォントタイプ（サイズやプロポーショナル、固定長の違いなど）やハイライト効果によっても異なるからです。1080ポイントのモニターの半分の幅で、エディタ上で1行の全てのコード（コメントを含めた）が読めるようにするべきです。私達がRedのコードベースをコーディングしているディスプレイ上では、概ね100文字です。以下の説明の通り、サイズを超えたり長過ぎる 表現は前述の基準に違反するコードサイズの行になるでしょう。

3. インデント

Redのコードベースはソースコードのインデントとして **4** 文字のタブを使用しています。これは小さすぎる値（2文字）や大きすぎる値（8文字）との間のよいトレードオフです。タブを使うことはルールに準拠しつつ（右寄せタブを使うことにだけ注意してください）、エディタ上で自分の好みに合わせて表示を調整できることを意味します。

red/redリポジトリ上の全てのRedのファイルはヘッダに以下のフィールドを含める必要があります。

```
Tabs: 4
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

```
func [
    arg1
    arg2
    ...
][
    print arg1
    ...
]
```

□□□

```
func [
arg1                ;-- □□□□□□□□□
arg2
...
][
       print arg1       ;-- □□□□□□□□□□□
       ...
]
```

4. □□□□□□□□□□

□□□□□□□□□□□□□□□□ `[]` □□□□□□ `()` □□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
a: []
```

□□□□□□□□□□□□□□□□□□

```
[][]
[]()
```

```
[
   hello
][                          ;-- □□□□□□□□□
   world
]
```

[illegible]

```
array: [[] [] [] []]
list:  [ [] [] [] [] ]

either a = 1 [["hello"]][["world"]]
either a = 1 [ ["hello"] ][ ["world"] ]
```

[illegible]

```
b: either a = 1 [a + 1][3]
```

[illegible]

[illegible]

```
b: either a = 1 [
    a + 1
][
    123 + length? mold a
]
```

[illegible]

```
b: either a = 1
    [a + 1][123 + length? mold a]
```

[illegible]*Red*[illegible] `either` [illegible]

[illegible]

```
print either a = 1 ["hello"][
    append mold a "this is a very long expression"
]

while [not tail? series][
    print series/1
    series: next series
]
```

5. [illegible]

[illegible] [illegible] [illegible]
[illegible]Red[illegible]word[illegible]
[illegible][illegible] [illegible]

[illegible] `-`
[illegible]2[illegible]

[illegible]

```
code: 123456
name: "John"
table: [2 6 8 4 3]
lost-items: []

unless tail? list [author: select list index]
```

[illegible]

```
code_for_article: 123456
Mytable: [2 6 8 4 3]
lostItems: []

unless tail? list-of-books [author-property: select list-of-books selected-index]
```

[illegible] [illegible] [illegible]
[illegible]word[illegible]
[illegible] `logic!`
[illegible] `length?`[illegible]
`index?`[illegible]

[illegible]

```
make:   func [...
reduce: func [...
allow:  func [...
crunch: func [...
```

[illegible]

```
length:    func [...
future:    func [...
position:  func [...
blue-fill: func [...       ;-- fill-blue[illegible]
```

[illegible]OS[illegible]Red[illegible]API[illegible]API[illegible]
[illegible]Red[illegible]Red/System[illegible]

```
tagMSG: alias struct! [
    hWnd   [handle!]
    msg    [integer!]
    wParam [integer!]
    lParam [integer!]
    time   [integer!]
    x      [integer!]
    y      [integer!]
 ]

#import [
   "User32.dll" stdcall [
       CreateWindowEx: "CreateWindowExW" [
           dwExStyle    [integer!]
           lpClassName  [c-string!]
           lpWindowName [c-string!]
           dwStyle      [integer!]
           x            [integer!]
           y            [integer!]
           nWidth       [integer!]
           nHeight      [integer!]
           hWndParent   [handle!]
           hMenu        [handle!]
           hInstance    [handle!]
           lpParam      [int-ptr!]
           return:      [handle!]
       ]
   ]
]
```

6. □□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□ GMT □□□□□□□
- □□□□□□□□□□□□Red□□□□□□□□□□□□API□□□□□□□□□□□

7. □□□□Red/System□

Red/System□□ _ □□□□□□□□□□□□□□□□□□□□□□□

□□□□*Red*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

8. [illegible]

[illegible]Red/System[illegible]

[illegible]

```
do-nothing: func [][]
increment: func [n [integer!]][n + 1]

increment: func [n [integer!]][
   n + 1
]

increment: func [
    n [integer!]
][
    n + 1
]
```

[illegible]

```
do-nothing: func [
][
]

do-nothing: func [

][

]

increment: func [
    n [integer!]
][n + 1]
```

[illegible]/local[illegible]word[illegible]

[illegible]

正しい例

```
make-world: func [
    earth   [word!]
    wind    [bitset!]
    fire    [binary!]
    water   [string!]
    /with
        thunder [url!]
    /only
    /into
        space   [block! none!]
    /local
    plants animals men women computers robots
][
    ...
]
```

悪い例

```
make-world: func [
   [throw] earth [word!]        ;-- 属性ブロックが独立した行になっていない
       wind    [bitset!]
       fire [binary!]           ;-- 型の定義が揃っていない
       water   [string!]
       /with
           thunder [url!]
       /only
       /into space [block! none!]  ;-- /withとフォーマットが揃っていない
       /local
           plants animals       ;-- 改行が早すぎる
           men women computers robots
][
    ...
]
```

docstringは、スペックブロックが折り返しする場合のメインのドキュメント（関数自体の説明）は独立した行に記載します。引数とリファインメントのdocstringは対象と同じ行に記載します。docstringは大文字で始め、文末にはピリオドはなくてかまいません（ help 関数は自動的にピリオドを補完します）。

正しい例

```
increment: func ["Add 1 to the argument value" n][n + 1]

make-world: func [
    "Build a new World"
    earth    [word!]      "1st element"
    wind     [bitset!]    "2nd element"
    fire     [binary!]    "3rd element"
    water    [string!]
    /with         "Additional element"
        thunder [url!]
    /only         "Not implemented yet"
    /into         "Provides a container"
        space [unset!]    "The container"
    /local
        plants animals men women computers robots
][
   ...
]
```

悪い例

```
make-world: func ["Build a new World"  ;-- 改行するべき
    earth  [word!]     "1st element"
    wind   [bitset!]     "2nd element" ;-- インデントが多すぎる
    fire   [binary!]
    "3rd element"          ;-- `fire` と同じ行に書くべき
    water  [string!]
    /with          "Additional element"
           thunder [url!]
    /only "Not implemented yet"    ;-- 他のdocstringと列を揃えるべき
    /into
          "Provides a container"      ;-- リファインメントの後に書くべき
           space [unset!]  "The container"
    /local
        plants animals men women computers robots
][
       ...
]
```

9. 関数呼び出し

関数の後の引数は同じ行に書きます。もし行が長くなりすぎる場合、引数は複数行（1つの引数につき1行）にまたがってインデント付きで記載します。

正しい例

```
foo arg1 arg2 arg3 arg4 arg5

process-many
    argument1
    argument2
    argument3
    argument4
    argument5
```

悪い例

```
foo arg1 arg2 arg3
    arg4 arg5

foo
    arg1 arg2 arg3
    arg4 arg5

process-many
    argument1
        argument2
            argument3
                argument4
                    argument5
```

ネストが多く長い式については、それぞれの式の区切りを明示することが難しい場合があります。引数を伴うネストした関数呼び出しをグルーピングするために、丸括弧を使うことは許可されます（ただし、必須ではありません）。

```
head insert (copy/part [1 2 3 4] 2) (length? mold (2 + index? find "Hello" #"o"))

head insert
    copy/part [1 2 3 4] 2
    length? mold (2 + index? find "Hello" #"o")
```

10. コメント

Redのコードベースでは以下のようになっています。

- 視覚的に分かりやすくするため、プレフィックスとして `;--` を使います。
- 1行のコメントは57列目から始まります（平均的にはこれが最も適切な列数です。もしくは53列目を使います）。
- 複数行のコメントは `comment {…}` 形式よりも、1行コメントを複数行使って記載します。

一般的なルールとして、縦のスペースを取りすぎないように、コメントは対応するコードと同じ行に書きます。ただし、コメントがコードと分離している場合、新しい行に書いても問題ありません。

11. 文字列のシンタックス

1行の文字列は "" を使います。 {} 形式は複数行文字列のために使います。このルールを遵守することで、以下のことが保証できます。

- コードをloadする前後におけるソースの表示の一貫性
- 意味の分かりやすさ

このルールの例外は1行の文字列の中身に " を含む場合です。この場合、 {} 形式を使うことで、エスケープクォート ^" を使わずにすむため、可読性が向上します。

12. 改行の使い方

未定