

# うわっ...私のコマンド、遅すぎ...？

## Rust再実装コマンドでターミナルをもっと便利に

ぱうえる（甲本健太）

Github : [@kentakom1213](https://github.com/kentakom1213)

# Rustとは

- C/C++並に速く、安全性が高い言語
- 主な用途
  - コマンドラインツール
  - Webアプリケーション
  - OSの開発

公式：[www.rust-lang.org/ja](http://www.rust-lang.org/ja)



# 今回紹介するコマンドたち

前	後	コマンドの意味
find	fd	ファイルを探す
grep	ripgrep	ファイルの中身を検索する
ls	exa	ディレクトリの中を見る
cat	bat	ファイルの中を見る

`find` → `fd`

ファイルを探す

- 高速！

`README.md` (7569個) を計測

- `find` : 8.0 sec
- `fd` : 0.75 sec

- デフォルトで色付けあり
- デフォルトで `.gitignore` を無視

公式 : [github.com/sharkdp/fd](https://github.com/sharkdp/fd)

[illegible][illegible]

`grep` → `ripgrep`

ファイルの中身を検索する

- **100倍程度高速！**

1434ファイルの中から `"アルゴリズム"` という文字列を検索

- `grep` : 3.8 sec
- `ripgrep` : 0.062 sec

- **みやすい！**

公式：[github.com/BurntSushi/ripgrep](https://github.com/BurntSushi/ripgrep)

# grep コマンド

```
> grep -r "アルゴリズム" .
./abc_training/C/C_IncreasingSequense.py:# 0(n^2) のアルゴリズム、部分点はもらえる
./Sakumon/Product_of_Range/Product_of_Range_ans.md:ある数列の連続する部分列の総和を高速に処理
./Sakumon/Product_of_Range/Product_of_Range_ans.md:累積和アルゴリズムと同様に、前処理が  $O(N)$ 
./algorithm/repeated_squares.ipynb: "# 高速累乗計算アルゴリズム\n",
./algorithm/repeated_squares.ipynb: "0(log(n))のアルゴリズムは強い\n",
./algorithm/LCS.py:### アルゴリズム ###
./algorithm/binary_tree.ipynb: "- Pythonによるアルゴリズム入門 (酒井和哉)"
./algorithm/README.md:# アルゴリズムレパートリー
./algorithm/README.md:競プロでこれまでに勉強したアルゴリズムと、これから挑戦したいもの
./algorithm/README.md:一部アルゴリズムと呼べるかあやしいものもあります。
./algorithm/README.md:参考: [競技プログラミングでの典型アルゴリズムとデータ構造](https://alg
./algorithm/README.md:## アルゴリズム
./algorithm/README.md:### ソートアルゴリズム
./algorithm/README.md:### 探索アルゴリズム
./algorithm/README.md:- [ ] カラツバ法 (高速乗算アルゴリズム)
./algorithm/rust_binary_tree/README.md:- [二分木 - Rustでは始めるデータ構造とアルゴリズム (第
./algorithm/PartitionFunction.ipynb: "# 分割数を求めるアルゴリズム\n",
./paiza/jack/in.txt:競プロ経験者の方だけでなく、「やってみたいけど敷居が高い…」とか「アルゴ
./paiza/jack/in.txt:内容は「最近勉強したアルゴリズム」「こだわりのエディタ」「コンテスト前の
./ac-library/document_ja/string.html:<p>文字列アルゴリズム詰め合わせです。
./ac-library/document_ja/string.html:文字列に関する様々なアルゴリズムが入っています。</p>
./ac-library/document_ja/appendix.html:<p>C++初心者には難しいかもしれない機能を表すマークです
が該当します。</p>
./ac-library/document_ja/modint.html:<p>自動でmodを取る構造体です。AC Libraryはmodintを使わな
>
./ac-library/document_ja/math.html:<p>数論的アルゴリズム詰め合わせです。</p>
./notebooks/graph_algorithms.ipynb: "# アルゴ式 グラフアルゴリズム"
```

# ripgrep コマンド

```
> rg "アルゴリズム"
notebooks/graph_algorithms.ipynb
7:      "# アルゴ式 グラフアルゴリズム"

algorithm/PartitionFunction.ipynb
74:      "# 分割数を求めるアルゴリズム\n",

algorithm/rust_binary_tree/README.md
4:- [二分木 - Rustではじめるデータ構造とアルゴリズム (第1回)](https://laysakura.github.io/2019/08/24/rust-binary-tree/)

algorithm/README.md
1:# アルゴリズムレパートリー
2:競プロでこれまでに勉強したアルゴリズムと、これから挑戦したいもの
3:一部アルゴリズムと呼べるかあやしいものもあります。
5:参考:[競技プログラミングでの典型アルゴリズムとデータ構造](https://algo-logic.info/competitive-programming/)
8:## アルゴリズム
10:### ソートアルゴリズム
18:### 探索アルゴリズム
67:- [ ] カラツバ法 (高速乗算アルゴリズム)

algorithm/LCS.py
11:### アルゴリズム ###

algorithm/binary_tree.ipynb
11:      "- Pythonによるアルゴリズム入門 (酒井和哉)"

algorithm/repeated_squares.ipynb
7:      "# 高速累乗計算アルゴリズム\n",
345:      "O(log(n))のアルゴリズムは強い\n"
```



ls → exa

ディレクトリの中を表示する

- デフォルトで色付けあり
- 絵文字を設定できる 😊
- tree形式でも表示できる

公式：[github.com/ogham/exa](https://github.com/ogham/exa)

```
> ls -1
Dockerfile.django
LOG.md
README.md
blog
db.sqlite3
django_blog
docker-compose.yml
manage.py
requirements.txt
```

```
> exa -1 --icons
📁 blog
📄 db.sqlite3
📁 django_blog
🚢 docker-compose.yml
📄 Dockerfile.django
📄 LOG.md
🐍 manage.py
📄 README.md
📄 requirements.txt
```

cat → bat

ファイルの中身を表示する

- まるでエディタのようにファイルを見れるコマンド
- デフォルトでシンタックスハイライトあり

公式：[github.com/sharkdp/bat](https://github.com/sharkdp/bat)

> cat solver.py

```
def solve(arr, cell):
    """単純なdfsで解く"""

    # 求まったとき
    if cell == 81:
        yield arr[:]
        return

    i, j = cell//9, cell%9
    if arr[i][j] == 0:
        # 順に代入する
        for n in range(1, 10):

            # --- 行 ---
            if n in arr[i]:
                continue

            # --- 列 ---
            is_in_col = False
            for r in range(9):
                is_in_col |= arr[r][j] == n
            if is_in_col:
                continue

            # --- ブロック ---
            is_in_block = False
            for r in range(i//3*3, i//3*3+3):
                for c in range(j//3*3, j//3*3+3):
                    is_in_block |= arr[r][c] == n
            if is_in_block:
                continue

            # 条件をみたしていたとき
            arr[i][j] = n
            yield from solve(arr, cell+1)
            arr[i][j] = 0
        else:
            yield from solve(arr, cell+1)

def main():
    arr = [list(map(int, input().split())) for _ in range(9)]

    for ans in solve(arr, 0):
        print()
        for row in ans:
            print(*row)

    solve(arr, 0)

if __name__ == "__main__":
    main()
```

> bat solver.py

File: solver.py	
1	
2	def solve(arr, cell):
3	"""単純なdfsで解く"""
4	
5	# 求まったとき
6	if cell == 81:
7	yield arr[:]
8	return
9	
10	i, j = cell//9, cell%9
11	if arr[i][j] == 0:
12	# 順に代入する
13	for n in range(1, 10):
14	
15	# --- 行 ---
16	if n in arr[i]:
17	continue
18	
19	# --- 列 ---
20	is_in_col = False
21	for r in range(9):
22	is_in_col  = arr[r][j] == n
23	if is_in_col:
24	continue
25	
26	# --- ブロック ---
27	is_in_block = False
28	for r in range(i//3*3, i//3*3+3):
29	for c in range(j//3*3, j//3*3+3):
30	is_in_block  = arr[r][c] == n
31	if is_in_block:
32	continue
33	
34	# 条件をみたしていたとき
35	arr[i][j] = n
36	yield from solve(arr, cell+1)
37	arr[i][j] = 0
38	else:
39	yield from solve(arr, cell+1)
40	
41	
42	def main():
43	arr = [list(map(int, input().split())) for _ in range(9)]
44	
45	for ans in solve(arr, 0):
46	print()
47	for row in ans:
48	print(*row)
49	
50	solve(arr, 0)
51	
52	if __name__ == "__main__":
53	main()
54	~

# その他おすすめツール

- [xcp](#) : `cp` コマンドの再実装
- [delta](#) : `diff` コマンドの再実装
- [Starship](#) : コマンドラインを可愛くできるツール

## 参考

- [awesome-alternatievs-in-rust](#)