誰にでも書ける #! /bin/sed -f 講座
第 2 回
「タルィフの転逆」

安岡孝一

*yasuoka* ： root さん、 root さん。
*root* ： 何だい？
*yasuoka* ： 強力な命令。
*root* ： ああ、そうだったね。じゃまずは、次のファイルを~/bin の中に作ってくれるかい？ ファイル名は yasuoka だ。

```
#! /bin/sed -f
s/yasuoka/YASUOKA/g
```

（間）

*yasuoka* ： できました。
*root* ： じゃ、 who の結果をその yasuoka に繋いでごらん。
*yasuoka* ： はい。

```
/home/yasuoka/bin% who | yasuoka （ぽこ）
YASUOKA  console Mar 10 14:09
fujii    ttyp0   Mar 10 14:11   (daikaku)
takahash ttyp2   Mar  9 09:21   (kinkaku:0.0)
/home/yasuoka/bin% ■
```

お？ yasuoka が大文字になっちゃった。そうすると

```
/home/yasuoka/bin% pwd | yasuoka （ぽこ）
/home/YASUOKA/bin
/home/yasuoka/bin% ■
```

あ、やっぱり。
*root* ： s/yasuoka/YASUOKA/gってのは、バッファの中の yasuoka を全部 YASUOKA に置き換える、っていう命令なんだ。

```
s/正規表現/文字列/
```

バッファ中で、最初に正規表現とマッチした部分を、文字列に置き換える。

```
s/正規表現/文字列/g
```

バッファ中で、正規表現にマッチした部分を、全て文字列に置き換える。
いずれも、文字列中に正規表現の \1 … \9 を用いてもよい。なお、文字列中の文字として /、 \、 & を用いる際には、それぞれ \/、 \\、 \& とすること。

*root* ： で、このスクリプトは前回にも言ったように

```
#! /bin/sed -f
s/yasuoka/YASUOKA/g
p
d
```

と同じだから、置き換えられた後のバッファが出力されるわけだ。
*yasuoka* ： バッファの中に yasuoka がなかったときには、どうなるんですか？
*root* ： そのときには s/yasuoka/YASUOKA/g は、何もしない。
*yasuoka* ： ふーん。この s がそんなに強力なんですか？
*root* ： 強力も強力。特に \( \) が、あまりにも強力。~/bin の中に次のファイルを作ってくれるかい？ ファイル名は rev だ。

```
#! /bin/sed -f
h
G
:loop
s/^\(.*\).\(\n\)\(.\)/\3\1\2/
/\n./bloop
s/\n//
```

（間）

*yasuoka* ： できました。どういうコマンドなんですか？
*root* ： 試しに rev rev を実行してごらん。
*yasuoka* ： はい。

```
/home/yasuoka/bin% rev rev （ぽこ）
f- des/nib/ !#
h
G
pool:
```

```
/2\1\3\/)\.(\)\n\(\.)\*.(\^/s
poolb/.n\/
//n\/s
/home/yasuoka/bin% 
```

ん？ どういうことですか？

*root* ： 各行が逆になってるだろ。

*yasuoka* ： え？ ああ、ああ、わかりました。

```
/home/yasuoka/bin% cat rev （ぽこ）
#! /bin/sed -f
h
G
:loop
s/^\(.*\).\(\n\)\(.\)/\3\1\2/
/\n./bloop
s/\n//
/home/yasuoka/bin% 
```

でも、いったいどういう仕掛けなんですか？

*root* ： まあ s/^\(.*\).\(\n\)\(.\)/\3\1\2/ が全てだね。でもその前に、4 行目の: と 6 行目の b を説明しておこう。

| |
|---|
| bラベル<br>　　「:ラベル」にジャンプする。<br>:ラベル<br>　　「bラベル」のジャンプ先。何もしない。 |

*yasuoka* ： ジャンプ命令もあるんですね。

*root* ： うん。さて、それじゃ例として

```
/home/yasuoka/bin% echo live | rev （ぽこ）
evil
/home/yasuoka/bin% 
```

を実行した時の、 rev の動作を説明しよう。

*yasuoka* ： はい。

*root* ： まず「live」がバッファに読み込まれた状態で、実行開始。次の h と G で、バッファは「live改live」になる。

*yasuoka* ： ええと、 h でコピーバッファに写して、 G でそれをバッファの末尾に追加だから、はいそうですね。

*root* ： で、:loop では何もせず、次の s でバッファは「lliv改ive」になる。

*yasuoka* ： そこがわかりません。

*root* ： うん、じゃまずは \( \) を無視して考えよう。 /^.*.\n./っていう正規表現は、「live改live」っていうバッファにマッチするかい？

*yasuoka* ： .って何でしたっけ？

*root* ： 任意の 1 文字。

*yasuoka* ： すると.* は、任意の 0 文字以上の文字列って意味ですね。 \n は？

*root* ： 改行コード。前回の正規表現のところに書いてあったろ？

*yasuoka* ： そうでしたっけ？ うーん、ま、とにかくそれならマッチします。

*root* ： どういう風に？

*yasuoka* ： ええと、最初の.* がバッファの先頭の liv に、次の. がその直後の e に、その後に改行コード、最後の. がその後の l に。 ive は残ります。

*root* ： よくわかったね。で、 /^\(.*\).\(\n\)\(.\)/ だと、最初の.* にマッチした文字列が \1 に、次の次の \n が \2 に、最後の. にマッチした文字が \3 に格納されるわけだ。

*yasuoka* ： あ、するとここでは \1 には liv、 \2 には改行コード、 \3 には l が格納されるんですね。

*root* ： そう。そしてそれを \3\1\2 の順に並べると？

*yasuoka* ： lliv改、になります。

*root* ： つまり「live改live」のうち、 live改l、の部分が、 lliv改、に置き換わるわけだ。これでバッファは「lliv改ive」になる。

*yasuoka* ： でもこれにどういう意味があるんですか？

*root* ： この s/^\(.*\).\(\n\)\(.\)/\3\1\2/ はようするに、改行コードの直前の 1 文字を削除すると同時に、改行コードの直後の 1 文字をバッファの先頭にもってくる、っていう意味なんだよ。

*yasuoka* ： はい。

*root* ： 動作の説明を続けようか。次の /\n./bloop は、改行コードの後に文字があるなら:loop にジャンプ、っていう意味だ。バッファは今「lliv改ive」だから:loop にジャンプする。

*yasuoka* ： loop の名のとおり、繰り返すわけですね。

*root* ： で、また s を実行して、バッファは「illi改ve」になる。また:loop にジャンプして、 s でバッファは「vill改e」に。もう一度:loop にジャンプして、 s で「evil改」。次の /\n./ は真にならないから、もうジャン

プは起こらない。最後の s/\n// で改行コードが削られて、バッファは「evil」になるわけだ。

*yasuoka* : その後、見えない p で改行コード付きで出力されて、さらに d を実行するわけですね。うーん、強力。

*root* : ま、そうやって、入力を１行ずつ処理していく。

*yasuoka* : でもこんな s/^\(.*\).\(\n\)\(.\)/\3\1\2/ なんて、ちょっと思いつかないんですけど。

*root* : その辺は慣れだね。

*yasuoka* : そうですか。

*root* : s の親戚に y っていう命令があるんだけど、今度はそれを使った例を示そうか。ファイル名は number だ。

```
#! /bin/sed -f
x
1s/^.*$/     1/
G
h
y/ 0123456789/11234567890/
G
s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/
x
s/\n/  /
```

（間）

*yasuoka* : できました。それにしても長い s ですね。

*root* : 大したことはやってないんだけどね。さて、この前の例はあるかな。

```
/home/yasuoka/bin% cat bow （ぽこ）
#! /bin/csh
# "bow" Version 2.0

if ($#argv) then
  set TIME=$argv[1]
else
  set TIME=3
endif
while ($TIME > 0)
  echo BOWWOW
  @ TIME--
end

exit (0)
/home/yasuoka/bin%
```

お、まだあった。じゃ number bow を実行してみてごらん。

*yasuoka* : number bow ですか。

```
/home/yasuoka/bin% number bow （ぽこ）
     1  #! /bin/csh
     2  # "bow" Version 2.0
     3
     4  if ($#argv) then
     5    set TIME=$argv[1]
     6  else
     7    set TIME=3
     8  endif
     9
    10  while ($TIME > 0)
    11    echo BOWWOW
    12    @ TIME--
    13  end
    14
    15  exit (0)
/home/yasuoka/bin%
```

あ、行番号が付くんですね。どういう仕掛けなんですか？

*root* : これはなかなか変態的なスクリプトなんだけど。

```
/home/yasuoka/bin% cat number （ぽこ）
#! /bin/sed -f
x
1s/^.*$/     1/
G
h
```

```
y/ 0123456789/11234567890/
G
s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/
x
s/\n/  /
/home/yasuoka/bin% 
```

まずは 2 行目の x を教えておこう。

| x<br>バッファの内容とコピーバッファの内容を交換する。 |
|---|

*root* : で、問題の 6 行目の y だけど、これは Unix の tr みたいな命令なんだ。

| y/文字列/文字列/<br>バッファの中にある文字のうち、前の文字列に含まれる文字を、後の文字列の同じ順番のところにある文字に置き換える。なお、2 つの文字列は同じ長さでなければならない。 |
|---|

*yasuoka* : 実際にはどういう動作になるんですか？

*root* : さっきの number bow の例でいくと、まず「#! /bin/csh」をバッファに読み込んだ状態で、実行開始。x でバッファとコピーバッファの内容を交換して、1s/^.*$/　　1/ でバッファを強制的に「　　1」にする。

*yasuoka* : /^.*$/ は、バッファ全体にマッチしますからね。

*root* : それから G でバッファは「　　1改#! /bin/csh」になって、h でコピーバッファにコピー。次の y/ 0123456789/11234567890/ で、バッファは「111112改#!1/bin/csh」になる。

*yasuoka* : バッファの先頭の空白は 5 個だったんですね。

*root* : うん。次の G で「111112改#!1/bin/csh改　　1改#! /bin/csh」、さらに次の s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/ でバッファは「　　2」になる。

*yasuoka* : え？

*root* : つまり /^.*[^0]0*\n.*\n.*[^9]9*\n.*$/ を考えると、.* には 11111 が、[^0] には 2 が、 0* には長さ 0 の文字列が、改行コードがあって、.* には #!1/bin/csh が、また改行コードがあって、.* には空白 5 個が、 [^9] には 1 が、 9* には長さ 0 の文字列が、さらにまた改行コードがあって、.* には #! /bin/csh が、それぞれマッチングするからね。 \1 には 2 が、 \2 には 5 個の空白が、それぞれ格納される。

*yasuoka* : [^0] って何でしたっけ？

*root* : 0 以外の 1 文字。

*yasuoka* : それでも、何が何だかわかりません。

*root* : うーん。ようするに「111112改#!1/bin/csh改　　1改#! /bin/csh」のうち、改#!1/bin/csh改、のところと、改#! /bin/csh、のところは削除されるんだ。

*yasuoka* : はい。

*root* : 最初の 111112 のところは、最後の 1 文字が \1 に格納される。ただし最後の 1 文字が 0 だったなら、 0 でない文字まで戻って、そこから後を全部 \1 に格納する。それから　　1 のところは、最後の 1 文字以外が \2 に格納される。ただし最後の 1 文字が 9 だったなら、 9 でない文字まで戻って、そこから前を \2 に格納する。

*yasuoka* : うーん、わかったような、わからないような。

*root* : じゃ、続きを説明しよう。次の x で、バッファは「　　1改#! /bin/csh」に、コピーバッファは「　　2」になる。さらに s/\n/  / で、バッファは「　　1  #! /bin/csh」になって、見えない p で出力、見えない d で次の「# "bow" Version 2.0」がバッファに読み込まれる。

*yasuoka* : はい。

*root* : 最初の x でバッファとコピーバッファを交換し、 1s は入力がもう 2 行目だから実行されなくて、 G でバッファは「　　2改# "bow" Version 2.0」になる。さらに h でコピー。

*yasuoka* : はい。

*root* : y でバッファは「111113改#1"bow"1Version13.1」になって、次の G で「111113改#1"bow"1Version13.1改　　2改# "bow" Version 2.0」、さらに s でバッファは「　　3」になる。

*yasuoka* : ああ、そうやって 1 つずつ増やしてるんですね。

*root* : そうそう。で、 x と s でバッファは「　　2  # "bow" Version 2.0」に、コピーバッファは「　　3」になる。

*yasuoka* : ははーん。わかってきました。でも桁上がりはどうなってるんですか？

*root* : これで実行が 9 行目まで進んで、コピーバッファが「　　9」になった時を考えよう。

*yasuoka* : はい。

*root* : bow の 9 行目は空行だから「」がバッファに読み込まれて、スクリプトの最初に戻る。

*yasuoka* : 見えない d ですね。

*root* : x G h で、バッファとコピーバッファは両方とも「　　9改」になる。
*yasuoka* : x でバッファは「　　9」、コピーバッファは「」だから、はい。
*root* : y でバッファは「111110改」になり、次の G で「111110改改　　9改」になる。次の s では、 \1 には 10 が、 \2 には 4 個の空白が格納されるから、バッファは「　　10」となる。
*yasuoka* : \1 には、 111110 の最後の 1 文字が格納されるんじゃなかったですか？
*root* : さっきも言ったろ。最後の 1 文字が 0 だったなら、 0 でない文字まで戻るって。
*yasuoka* : ああ、そうでしたね。
*root* : 細かく言うと .*\([^0]0*\) と 111110 のマッチングだから、 .* には 1111 が、 [^0] には 1 が、 0* には 0 が、それぞれマッチングして、 \1 には 10 が格納されるわけだ。 \(.*\)[^9]9* と　　9 のマッチングは逆に、末尾の 9 とその直前の 1 文字が削除されて、 \2 に 4 個の空白が格納される。
*yasuoka* : すごーい。ちょっと考えつきませんね。
*root* : まあね。これを繰り返して、最終行まで行番号を付けるわけだ。
*yasuoka* : 999999 まで行くんですか？
*root* : うん。でもその後は崩れちゃうな。
*yasuoka* : ふーん。

```
/home/yasuoka/bin% number number （ぽこ）
     1  #! /bin/sed -f
     2  x
     3  1s/^.*$/     1/
     4  G
     5  h
     6  y/ 0123456789/11234567890/
     7  G
     8  s/^.*\([^0]0*\)\n.*\n\(.*\)[^9]9*\n.*$/\2\1/
     9  x
    10  s/\n/  /
/home/yasuoka/bin% 
```

6 行目の y/ 0123456789/11234567890/ のところは、 y/ 0-9/11-90/ とかいう風には書けないんですか？
*root* : 書けない。
*yasuoka* : じゃあ、小文字の大文字変換だと、 52 文字全部書かなきゃいけませんね。
*root* : まあ、そうかな。
*yasuoka* : それは大変だなぁ。

```
/home/yasuoka/bin% number rev （ぽこ）
     1  #! /bin/sed -f
     2  h
     3  G
     4  :loop
     5  s/^\(.*\).\(\n\)\(.\)/\3\1\2/
     6  /\n./bloop
     7  s/\n//
/home/yasuoka/bin% 
```

さっきの rev に戻りますけど、 5 行目では、どうして \n を \( \) でくくってるんですか？
*root* : って言うと？
*yasuoka* : 別にわざわざ \2 に格納しなくても、 s/^\(.*\).\n\(.\)/\2\1\n/ じゃ、ダメなんでしょうか？
*root* : s の後の文字列には、 \n は書けないからね。
*yasuoka* : え？ じゃあ、バッファに改行コードを挿入したい時は？
*root* : G とか H とかを巧みに使うか、その時点でバッファに入ってる改行コードを増やすしかないね。
*yasuoka* : そうなんですか。
*root* : ただバッファには普通、改行コードは読み込まれないから、 G とかで挿入した改行コードを区切り記号だとみなすのが、正規表現をうまく使うコツなんだ。そういう点で sed には、改行コードをバッファの区切りだとみなす命令も、いくつかある。
*yasuoka* : 他にもまだ命令があるんですか？
*root* : うん。でも今日はもう時間がないから、それはまた今度にしよう。
*yasuoka* : はい。どうもありがとうございました。