エンジニアには希望がある、哲学がある。Geekでポジティブな面白い仲間がいっぱい。

ISSN 2187-9664

USP MAGAZINE

Vol.5 summer 1000Yen

よりぬき版

for the sophisticated shell scripters

シェルスクリプトに何でもおまかせ

Open usp Tukubai シンプルレシピ

Part1・テキスト編

TechLION再録

まつもとゆきひろ

10年先も勝負できるプログラマーとは

清水浩先生に訊く、技術者哲学

好評連載

原色美女図鑑　第4回　ともちゃ

後藤大地　漢のUNIX

大岩元　今私たちは何を学ぶべきか　第5回

シェルスクリプト大喜利　他

From Editor

「〜に訊く、技術者哲学」と題したインタビュー連載を始めました。

今回インタビューに応じてくださったのは、“ 次世代の車 ” と呼ぶべき存在になる車を目指し、電気自動車を研究開発されている清水浩先生。インタビューの冒頭、「本物の技術者を追求したくて取材に参りました」と申し上げると「それは取材する相手を間違えましたね」と苦笑い。

確かにおっしゃる通りかもしれません。それは人々が心に思い描くだけの理想の人物像であって、「完璧な人間」同様、この世に実在しないのかもしれないと思いました。

しかしそのように思い描く人物像の片鱗を、実在の人物に感じる瞬間があります。普通の人とは比べ物にならない程の積極性や独創性があったり、巧いなーと感心してしまうような絶妙な技術センスを持っていたり、時にあっけにとられるほど楽観的だったり情熱的だったり……。今回私は、書店で出会った清水先生の本の中にその片鱗を見ました。

素直に凄いと思う言動は勿論、一見常人離れして見える言動であっても、どれもその人が築いた技術者哲学に基づき、確信的にやっているのだろう。そんな技術者哲学を色々な人から聞き、独自の哲学を確立していけば、誰しも本物の技術者に近づけるのではないだろうか。

新たに始めたこの連載にはそんな意図があります。TechLION 再録等の他の記事も、切り口は違えど、同じテーマを持っています。

日本の技術者がもっと幸せになればと、強く思います。

USP MAGAZINE 編集部

Contents

※ 薄字で記載している記事は、よりぬき晩には収録されておりません。また、本誌とは掲載順序が若干異なります。

特集1

シェルスクリプトに何でもおまかせ

Open usp Tukubai シンプルレシピ Part1・テキスト編

USPマガジン編集部

Tukubai、
それはシェルスクリプトの使い易さを加速させるコマンド群。
前号では、そのオープン版 “Open usp Tukubai” を紹介したが、
やり残したことがあった。豊富なレシピの紹介である。

そこで今号と次号の二回に渡り、
Tukubaiの、特に実際のコーディングスタイルが分かり易い
レシピを特集する。

「シェルスクリプトとは、僅かなコマンドの追加で、こんなにも応用の効く言語になるのか!」と、あなたはきっと驚くことだろう。

準備
Tukubaiを知り、そして始める。

レシピ1
帳票を作る。

レシピ2
ルービックキューブを作る。

準備　Tukubaiを知り、そして始める。

Tukubaiは、シェルスクリプトをプログラム開発言語として本気で使うべく作られたコマンド集だ。これを制作したUSP研究所は実際に、Tukubaiを用いて様々な業務システムの開発や開発支援を行っている。

2012年2月、このうち使用頻度が高いもの中心に選ばれ、オープンソースとして公開された。それがOpen usp Tukubaiである。

シェルスクリプトで開発するのはなぜか

なぜシェルスクリプト？

意外に思うかもしれないが、シェルスクリプトは上手に使えば、C言語並のハイパフォーマンスなプログラムを、C言語より遥かにラクに作れる。

ヒミツは、Unixシステムコールやライブラリーの殆ど素の能力を、Unixシェル自身やコマンド（cat, sed, awk, grep等々……の馴染みのもの）から直接使える点にある。Unixシステムコールやライブラリーはカーネルに直結しており、コンピューターの本気の性能や堅牢性の恩恵に預かれるのだ。Unixシェルはいわばその上に被さる薄い皮。だから、使わない手はない。

なぜTukubai？

しかし、実際の業務アプリケーションやWebアプリケーションを作ろうとすると、そこで頻出する処理を簡単にやってくれるコマンドがなかなか無い。一般的にシェルスクリプトが開発言語として利用されない理由もここにある。そこを補うべく作られたものがTukubaiというわけだ。

本特集では、Tukubaiを使うと、どんなことができて、それはどんなふうに書けるかということがわかるレシピを紹介しよう。

Open usp Tukubaiをインストールする

インストールは簡単。アーカイブをダウンロードして解凍し、make installするだけ。だが、もう少し詳しく説明してみよう。

事前に必要なもの

1. Python

Open版であるOpen usp Tukubaiのコマンドは Python（2.4以上の2.x系用）で出来ている。よって事前にインストールしておこう。Pythonの導入についての詳細は、Pythonサイト（http://www.python.org/）や、日本Pythonユーザ会さんのサイト（http://www.python.jp/）を参照して欲しい。

2. Bash（本記事のレシピで必要）

この後に紹介するレシピのコードを実行するにはBashが必要である。Linux以外のBashが導入されていない環境では導入しておくこと。その際、各コードの先頭にあるシバン(#! /bin/bash)のパスは変更が必要だ。

Open usp Tukubaiの導入

Open usp Tukubaiのダウンロードは Tukubaiに関するポータルサイトである「UEC」（usp engineers' community、https://uec.usp-lab.com/、→**写真1**）にて可能である。

このサイトの上部メニューから、

→“Tukubai”

→“Open usp Tukubai ダウンロード”

と進むと、そこで最新版が公開されている。wgetやfetchコマンド等を使って取得した後、次のようにして解凍、およびインストールする。

```
> tar jxf open-usp-ukubai-yyyymmdd.tar
.bz2
> sudo make install
```

レンタルサーバー等、root権限が与えられておらず、ホームディレクトリー等にインストール先を変更したい場合、次のようにmakeに与えるLOCALBASE変数でそのディレクトリー指定できる。

```
> tar jxf open-usp-ukubai-yyyymmdd.tar
.bz2
> make LOCALBASE=/home/USERNAME/tukuba
i install
```

この場合は、環境変数$PATHにTukubaiコマンドのパスを追加すること。上記例の場合、/home/USERNAME/tukubai/binとなる。

尚、アンインストールしたい場合はmake installの代わりにmake deinstallとすればよい。

写真1. Tukubaiのポータルサイト“UEC”

FreeBSDでの導入は更に簡単！

FreeBSDではportsコレクションにOpen usp Tukubaiが登録されている。従って、Open usp Tukubaiアーカイブファイルのダウンロード作業も、事前のPythonインストール作業も不要だ（これらは自動的に行われる）。

具体的にはrootになって下記のコマンドを打ち込むだけである。

```
# cd /usr/ports/devel/open-usp-tukubai
# make install clean
```

記事中のプログラムはUECサイトにてダウンロード可

さあ、これで準備完了だ。この後紹介するレシピで、Tukubaiの雰囲気を掴みとって欲しい。

しかしながら、それらのレシピを手で打ち込むのは大変な作業だ。そこで、本記事で紹介したレシピのプログラム、及びサンプルデータを、先程紹介したポータルサイトUECにてダウンロードできるようにしてある。是非併せて活用してもらいたい。

レシピ1　帳票を作る。

問題

シェルスクリプトで業務システムを作れるというが、それならシェルスクリプトやTukubaiを使って帳票を作ることはできるのか？できるなら例を見せてもらいたい。帳票というからには数値計算だけではなく、きちんとした見た目にレイアウトするところまで面倒を見てもらわないと困る。

解答

勿論できる。テキストファイル（スペース区切り）の各種テーブルが用意されていれば[*1]、Tukubaiコマンドを使って既存のリレーショナルデータベース（RDB）でできるデータの加工は概ねできる。加工されたデータはテキストデータになっているため、引き続き、スペース調整や罫線挿入を施すことで、レイアウトを整えることまでできる。

サンプルプログラム

全国のスーパーから集めた売上に関するデータテキストと、その他４つのマスターデータ（商品所属部門対応表、部門名表、店舗所在地域対応表、地域名表）テーブル群のテキストファイルを元データとし、ある一週間における、地域別・商品部門別の商品販売数の帳票を作成してみる。

元データは次の通り、いずれもスペース区切りのテキストデータである。

*1　本レシピでは割愛するが、スペース区切りになっていない生データ（例えばログファイル等）をそのように加工するのもTukubaiで行っている。

File1．売上データ「URE」

```
0001 0000007 20071201 117 8335 -145
0001 0000007 20071202 100 7821 -130
0001 0000007 20071203 103 6012 -115
0001 0000007 20071204 104 6341 -120
0001 0000007 20071205 102 6165 -135
0001 0000007 20071206 106 7201 -125
0001 0000007 20071207 108 6033 -115
0001 0000007 20071208 132 6106 -100
                  ⋮
                  ⋮
```

列構造（左から順に）
1. 店コード 2. 商品コード 3. 日付
4. 売数 5. 売上高 6. 値引高

File2．商品所属部門対応表「BUMON」

```
0000007 001
0000017 001
0000021 002
0000025 002
0000027 001
0000030 001
0000043 001
0000045 002
0000047 001
                  ⋮
                  ⋮
```

列構造（左から順に）
1. 商品コード 2. 部門コード

File3．部門名表「BNAME」

```
001 野菜
002 果物
003 魚介類
004 肉類
005 調味料
006 乾物その他
007 米
008 和風冷蔵
009 菓子
010 飲料
011 酒類
```

列構造（左から順に）
1. 部門コード 2. 部門名

File4．店舗所在地域対応表「TENAREA」

```
0001 1003
0002 1005
0003 1003
0004 1007
0005 1002
0006 1005
0007 1003
0008 1005
0009 1003
                  ⋮
                  ⋮
```

列構造（左から順に）
1. 商品コード 2. 部門コード

File5．地域名表「AREANAME」

```
1001 北海道
1002 東北
1003 関東
1004 中部
1005 近畿
1006 中四国
1007 九州
```

列構造（左から順に）
1. 地域コード 2. 地域名

リスト１．BUMONTREND：商品部門・地域別の売数推移の帳票を作成するプログラム

```
 1 #!/bin/sh
 2 #
 3 # BUMONTREND : 部門別売数推移(地域別内訳)
 4 #
 5 # Usage: BUMONTREND
 6 #
 7 # Written by N.Tounaka(tounaka@usp-lab.com) / Date : 12 Jun.2012
 8 # Arranged by USP MAGAZINE(mag@usp-lab.com) / Date : 12 Jun.2012
 9
10 # 変数の定義
11 tmp=tmp
12 start=20071203
13 end=20071209
14
15 ###################################################################
16 # 販売データの集計
17 #  URE       1.店コード 2.商品コード 3.日付 4.売数 5.売上高 6.値引高
18 #  BUMON     1.商品コード 2.部門コード
19 #  TENAREA   1.店コード 2.エリアコード
20 #  AREANAME  1.エリアコード 2.エリア名
21 #  BNAME     1.部門コード 2.部門名
22 ###################################################################
```

Tukubaiを使ったコーディングの作法では、いつ、誰が書いたのかを冒頭に記す。（コード内容が分からない時の手がかりとなる）

一時ファイルの置き場所定義

帳票での対象期間定義

変数を設定する場合は、なるべく冒頭で行う。

加工の対象となる元データ各々の列構造も、コメントとして記しておく。

次ページへ続く→

```
awk '$3>='$start'&&$3<='$end'{print}' URE
join1 key=2 BUMON
sort -k1,1
join1 key=1 TENAREA
# [LAYOUT] 1.店コード 2.エリアコード 3.商品コード 4:部門コード
#          5.日付 6.売数 7.売上高 8.値引高  #
self 4 2 5 6
# [LAYOUT] 1.部門コード 2.エリアコード 3.日付 4.売数
sort -k1,3
sm2 1 3 4 4
sort -k2,2
join1 key=2 AREANAME
sort -k1,1
join1 key=1 BNAME
# [LAYOUT] 1.部門コード 2.部門名 3.エリアコード 4.エリア名 5.日付 6.売数
awk '{print $1":"$2,$3":"$4,$5,$6}'
# [LAYOUT] 1.部門コード:部門名 2.エリアコード:エリア名 3.日付 4.売数
yobi -j 3
awk '{$3=$3"("$4")";print}'
delf 4
# [LAYOUT] 1.部門コード:部門名 2.エリアコード:エリア名 3.日付(曜日) 4.売数
# map : 部門/エリア単位に売数を日付横展開   #
map num=2
# awk : ヘッダー行を月/日(曜日)の形に加工   #
awk 'NR==1{for(i=3;i<=NF;i++){$i=substr($i,5,2)"/"substr($i,7)};print}
     NR>1{print}'
# ysum : 期間合計を付加する                 #
ysum +h num=2
# sm4 : 部門ごとの合計行を挿入する          #
sm4 +h 1 1 2 2 3 NF
awk '{if($2~/@/){$2="9999:全国計"};print}'
# awk : ヘッダー行の編集                    #
awk 'NR==1{$1="部門";$2="エリア";$NF="期間合計";print}
     NR>1{print}'
comma +h 3/NF
# awk : 部門ごとに罫線(横)を挿入する        #
awk 'BEGIN{k=sprintf("%0118d",0);gsub("0","=",k);}
     NR<=2{print;if(NR==1){print k;gsub("=","-",k);f=$1;}}
     NR>2{if($1!=f){f=$1;print k;}else if($1==f){f=$1;$1="@";}print;}
     END{print k;}'
keta -16 -12 10xNF-3 11
tr '@' ' '                                  > $tmp-body

# ヘッダーをつくる
y1=$(echo $start | cut -b 1-4)
m1=$(echo $start | cut -b 5-6)
d1=$(echo $start | cut -b 7-8)
w1=$(yobi -dj $start)
y2=$(echo $end | cut -b 1-4)
m2=$(echo $end | cut -b 5-6)
d2=$(echo $end | cut -b 7-8)
w2=$(yobi -dj $end)
```

- 売上データを本流データとし、これを最初にパイプに流す（他のマスターデータは、あくまでこの本流に付加する支流データ）
- 売上データのうち、処理対象期間を抽出する（≒ where 句）

店コードをキーにして、店舗所在地表と内部結合（≒ inner join 句）

「パイプのこの位置を流れているデータの列構造はどうなっているか」について要所要所で記しておくとよい。

指定列のみ抽出する“SELect Field”（≒ select 句）

- 1 列目（部門名）～ 3 列目（日数）までの列が完全に同一の行について、それぞれ 4 列目（売数）～ 4 列目（同）を縦に合計する（≒ group by 句）
- join 系 ,sm 系コマンドは、使用前にソートが必要

3 列目（日付）に基づき、その次の列との手前に曜日文字列を挿入（元データは一列増えることになる）

指定列のみ削除する“DELete Field”（≒ select 句）

縦並びの表を横に展開する。（後述）

2列目まで(部門コード,エリアコード)をキーとし、その次の列以降の値を横に加算。最終列の後ろに付け足す。

- sm2 の類似コマンド
- 1 列目（部門名）～ 1 列目（同）をキーとして扱い、3 列目～最終列（各曜日の売数）をそれぞれ縦に合計するのは同じだが、サブキーとして残したい列（今回は 2 列目（地域名）～ 2 列目（同））がある場合に使用。

可読性向上のため、3 列目以降（全て売数）の値にカンマを振る。(1000 ⇒ 1,000)

パイプの出口付近では、値の計算ではなく、見た目の調整を行っている。

comma や keta といったコマンドは、整形用途に分類される。

全行通して同じ列が縦に並ぶように、各行の各列間スペースを調節する。

パイプの出力を一旦、一時ファイルへ保存

帳票の上につける書類のタイトルは、先程のパイプとは別に作り、これも一時ファイルに保存

yobi コマンドは、標準入力のみならず、引数で加工元データを指定できる。

次ページへ続く→

Tukubai の名コマンドたち　その 1 ―map

map コマンドは縦並びのデータを横並びにしてくれる。しかしながら、この一言で表すのは難しい。下の図を見ながら動作を理解してもらいたい。

引数 num で数値 n を指定すると、1~n 列目までを縦（レコード）キーとし、n+1 列目を横（カラム）キーとして表を作り、そこに n+2 列目の値を拾って当てはめてくれる。主キー、サブキーと、キーが 2 階層あるようなデータの整理に活躍するコマンドなのである。

支店A	1F売数	435241
支店A	2F売数	302114
支店A	3F売数	260141
支店B	1F売数	312498
支店B	2F売数	221387
支店B	3F売数	179162

map num=1

*	1F売数	2F売数	3F売数
支店A	435241	302114	260141
支店B	312498	221387	179162

Tukubai の名コマンドたち　その 2 ―sm2 と sm4

sm2,sm4 の sm はサムアップの意であり、どちらも列合計を求める（＝縦に足す）計算をするためのものである。両者の違いは、主キーの他にサブキーを維持するか否かである。sm2 は維持しないためサブキー列を消し、列合計した行のみ出力する。一方 sm4 は、維持するため、列合計した行が増える形になる。

支店A	1F売数	435241
支店A	2F売数	302114
支店A	3F売数	260141
支店B	1F売数	312498
支店B	2F売数	221387
支店B	3F売数	179162

sm4 1 1 2 2 3 3

支店A	1F売数	435241
支店A	2F売数	302114
支店A	3F売数	260141
支店A	@@@@@@	997496
支店B	1F売数	312498
支店B	2F売数	221387
支店B	3F売数	179162
支店B	@@@@@@	713047

sm2 1 1 3 3

支店A	997496
支店B	713047

```
75 cat << END > $tmp-header
76
77   [ 部門別売数推移(エリア内訳) ]                    出力 : $(date)
78                                                    ユーザー : $USER
79
80   集計期間 : ${y1}年${m1}月${d1}日(${w1}) - ${y2}年${m2}月${d2}日(${w2})
81
82 END
83
84 # ヘッダーとデータを合わせて出力
85 cat $tmp-header $tmp-body ―――― タイトルと表本体を順番に出力する。
86
87 # 終了
88 rm -f $tmp-* ―――――――――― 作った一時ファイルを片付ける。
89 exit 0 ―――――――――――― 最後に必ず exit 0 を書くのも Tukubai のコーディング作法である。
```

このシェルスクリプトによって次の帳票テキストデータが生成される。

```
[ 部門別売数推移(エリア内訳) ]                    出力 : 2012年  6月 12日 火曜日 11:37:12 JST
                                                  ユーザー : usp

集計期間 : 2007年12月03日(月) - 2007年12月09日(日)

部門          エリア          12/03(月)   12/04(火)   12/05(水)   12/06(木)   12/07(金)   12/08(土)   12/09(日)     期間合計
==========================================================================================================================
001:野菜      1001:北海道       219,060     200,540     188,700     466,140     767,240     307,880     256,260   2,405,820
              1002:東北         801,520     770,380     776,260   1,779,080   3,103,740   1,406,820   1,440,400  10,078,200
              1003:関東       1,508,682   1,435,524   1,527,662   3,657,046   6,151,328   2,386,452   2,358,427  19,025,121
              1004:中部         925,600     901,900   1,010,100   2,357,820   3,876,960   1,328,460   1,401,480  11,802,320
              1005:近畿         538,160     528,840     627,180   1,479,700   2,331,060     839,360   1,066,000   7,410,300
              1006:中四国       793,080     771,740     657,420   1,948,460   2,840,420   1,402,460   1,198,920   9,612,500
              1007:九州       1,022,780     969,840     910,300   2,454,180   3,829,140   1,364,320   1,475,900  12,026,460
              9999:全国計     5,808,882   5,578,764   5,697,622  14,142,426  22,899,888   9,035,752   9,197,387  72,360,721
--------------------------------------------------------------------------------------------------------------------------
002:果物      1001:北海道       135,560     176,400     178,560     479,140     758,660     334,340     145,080   2,207,740
              1002:東北         645,560     528,240     672,000   1,480,240   2,100,880   1,020,220     997,440   7,444,580
              1003:関東       1,120,240   1,292,420   1,045,520   3,115,860   4,604,960   1,869,160   2,074,980  15,123,140
              1004:中部         744,940     783,120     731,860   1,876,680   2,983,840   1,048,560   1,242,560   9,411,560
              1005:近畿         433,920     478,880     444,260   1,169,600   1,659,360     879,900     769,900   5,835,820
              1006:中四国       593,540     640,560     547,280   1,720,500   2,111,420     878,580     854,960   7,346,840
              1007:九州         738,640     732,820     753,080   1,744,780   2,786,160   1,140,840   1,294,920   9,191,240
              9999:全国計     4,412,400   4,632,440   4,372,560  11,586,800  17,005,280   7,171,600   7,379,840  56,560,920
--------------------------------------------------------------------------------------------------------------------------
003:魚介類    1001:北海道       236,760     234,060     210,440     459,940     814,420     302,080     231,320   2,489,020
              1002:東北         853,480     881,020     755,400   1,945,400   2,971,220   1,015,780   1,209,820   9,632,120
              1003:関東       1,707,760   1,784,320   1,617,980   4,014,920   5,572,080   2,568,960   2,637,820  19,903,840
              1004:中部       1,071,700     961,360   1,121,160   2,410,680   3,779,520   1,544,360   1,777,260  12,666,040
              1005:近畿         619,860     607,840     587,200   1,584,200   2,362,300   1,035,640     972,340   7,769,380
              1006:中四国       782,640     878,100     709,480   1,905,380   3,044,100   1,591,460   1,350,780  10,261,940
              1007:九州       1,026,360   1,102,000     996,720   2,642,520   3,517,860   1,518,280   1,676,320  12,480,060
              9999:全国計     6,298,560   6,448,700   5,998,380  14,963,040  22,061,500   9,576,560   9,855,660  75,202,400
                                                          :
                                                          :
```

解説

このサンプルプログラムは、Tukubai を使って帳票を作るシェルスクリプトとしては非常に典型的な姿をしている。最大の特長は、何十段にも及ぶパイプの接続である。（途中いくつかコメントや AWK コードで遮られているものの）パイプ記号が綺麗に縦に並べられ、このシェルスクリプトにおいては実に 25 段にも及ぶ。

１本のパイプがもたらす効果

Tukubai コマンドを使えば、このような長いパイプ１本に元データを流し込むだけで帳票（きちんと整形された姿）を完成させてしまうことができる。途中通過するコマンドは、SQL の句を１つ１つ切り出したようなコマンドが多くあるあたりがまた面白い。

さて処理全体を１本のパイプに収めてしまうことには１つ大きなメリットがある。パイプを構成しているコマンドは全て同時起動し、工場の流れ作業のような並列処理が自然と実現されることとなるからだ。

プレーンテキストの帳票

帳票を作るというと、Office アプリや HTML といった手段で表現することを考えがちだが、このプレーンテキストの表を見て意外に様になっているようには感じないだろうか。これで事足りる場面が意外に多いのではないかと言いたいのである。（紙にしたければこのテキストファイルをプリンターデバイスに送れば良いだけだ）

勿論、Office アプリ向け (例えば CSV 形式) にして取り込ませたり、HTML タグを付けて装飾させたりも可能だ。しかし、そのために新たなコストを支払うこと鑑みれば、これで完成としてしまうことも十分ありなのではないだろうか。

このように、レイアウトという帳票作りの最終工程を簡単に済ませられるのも、テキストベースの手法をとっているメリットの一つと言えるだろう。

レシピ2　ルービックキューブを作る。

問題

Tukubai はシェルスクリプトをデータベース言語化させるためだけのコマンドセットではないというが、それならもっと一般的な例は無いのか。

数理的な操作の様子が伺えるコードが見てみたい。例えば、何らかの科学技術計算を行っているプログラムであれば、それがわかりやすいと思うのだが。

解答

実際に科学技術計算にも利用しているのだが、そのような例は専門分野の知識から解説しなければならない。そこでここでは専門分野知識の解説が不要なゲームのレシピを紹介しよう。数学パズル的要素が強いゲームであれば、数理的な操作も見え易いからだ。

サンプルプログラム

そこでルービックキューブ（3 × 3 のオーソドックスなもの）を作ってみた。

ルービックキューブは、立方体の一面が一つの行列に対応した 6 つの行列であると見なせる。キューブを動かすと、その行列のうちの一つで± 90°回転が発生し、それに伴って隣接する 4 つの面に相当する行列の一部要素が、別の行列に移動する。

この動きを、シミュレートしようというものである。

リスト 2．rubik-cube：ルービックキューブプログラム（ファイル 1・メイン）

```
#!/bin/bash
#
# 3 x 3ルービックキューブ
# Written by N.Tounaka(tounaka@usp-lab.com)
#                         Date : 5 Jun.2012
# Arranged by USP MAGAZINE(mag@usp-lab.com)
#                         Date : 6 Jun.2012
#                             U
# 立方体(6面体)の展開図＝L F R B
#                             D
# F:fore B:back U:up D:down R:right L:left
#
# 関数f:面Fを時計回り(右90度回転)
# 関数F:面Fを反時計回り(左90度回転)

# 変数定義(一時ファイルパス/着色データ)
tmp=tmp
F=`printf '¥033[43mF¥033[00m'`
U=`printf '¥033[46mU¥033[00m'`
R=`printf '¥033[41mR¥033[00m'`
B=`printf '¥033[45mB¥033[00m'`
D=`printf '¥033[44mD¥033[00m'`
L=`printf '¥033[42mL¥033[00m'`

# 関数のインクルード
source rubik-cube.func

# 初期化/シャッフル/答え作り(シャッフルの逆操作を記録)
init
rand ${1-=10} 2> $tmp-ans
ans=$(tac $tmp-ans | yarr -d | tr a-zA-Z A-Za-z)

# ゲームの開始
n=1
while true; do

  echo " Fore    Up      Right   Back    Down    Left"
  echo " ------------------------------------------"
  ycat -3 F U R B D L
  ycat /dev/null -
  sed -e "s/F/$F/g; s/U/$U/g; s/R/$R/g" ¥
      -e "s/B/$B/g; s/D/$D/g; s/L/$L/g"
  echo "<-f F-> <-u U-> <-r R-> <-b B-> <-d D-> <-l L->"
  echo -n "${ans}[$n]> "
  read -n 1 scr
  echo
  [ "$scr" = "q" ] && break
  eval ${scr} 2> /dev/null
  n=$((n+1))

done

rm F B U D R L $tmp-*
exit 0
```

ルービックキューブの面要素に色を付けるためのエスケープシーケンス定義

関数だけ別定義した下記のプログラムリストをインクルード

ここで文字列を逆順 (abc → cba) に並べようとしている。これをやりたいなら、対象文字列を一旦 tarr 等で縦並びにしておけば tac で縦の逆順にできるから。これを yarr で横並びにすればよい。

/dev/null と ycat することで行頭にスペースを挿れている。

- ここの F~L はキューブ各面の現在の要素の配列状況を記憶するための一時ファイル。(3×3行列)
- 各面の 3 × 3 の状況を横に並べて表示するために ycat を使っている。

キューブ 1 回操作につき 1 周の無限ループ。①現在状況表示、②操作指示待ち、③指示に基づく実際の操作、の繰り返し。

リスト 3．rubik-cube.func：ルービックキューブプログラム（ファイル 2・関数定義）

```
# RUBIK-CUBE エンジン
#
# Written by N.Tounaka(tounaka@usp-lab.com)
#                         Date : 5 Jun.2012
# Arranged by USP MAGAZINE(mag@usp-lab.com)
#                         Date : 6 Jun.2012

init ()    # 色が揃ったキューブをセットする
{
  for surf in F U B D R L; do
    yes "$surf" |
    head -9     |
    yarr -3     > $surf
  done
}

r ()  # R面(Right)の90度時計回り
{
  self 3 F | cat           > $tmp-f
  self 3 U | tac           > $tmp-u
  self 1 B | cat           > $tmp-b
  self 1 D | tac           > $tmp-d
  delf 3 F | ycat - $tmp-d > $tmp-w; mv $tmp-w F
  delf 3 U | ycat - $tmp-f > $tmp-w; mv $tmp-w U
  delf 1 B | ycat $tmp-u - > $tmp-w; mv $tmp-w B
  delf 1 D | ycat $tmp-b - > $tmp-w; mv $tmp-w D
  tateyoko R | self 3 2 1 > $tmp-w; mv $tmp-w R
}
R ()  # R面(Right)の90度反時計回り
{
  self 3 F | tac           > $tmp-f
  self 3 U | cat           > $tmp-u
  self 1 B | tac           > $tmp-b
  self 1 D | cat           > $tmp-d
  delf 3 F | ycat - $tmp-u > $tmp-w; mv $tmp-w F
```

FFF ←というような内容の文字列を作っている。最初 "F" を縦に 9 個作り、yarr を使って横 3 列に広げて敷き詰めている。

メインループで入力された 1 文字がこれら 12 通りの関数名のいずれかになっていた時、eval によりその関数が呼び出される。

tateyoko は転置行列を求めるコマンド。でも何と直感的なネーミングなことか。（但し日本人専用）

取り出す列番号を逆順に指定すれば、テーブルを左右反転させられるという self の応用例

次ページへ続く→

```
  delf 3 U | ycat - $tmp-b > $tmp-w; mv $tmp-w U
  delf 1 B | ycat $tmp-d - > $tmp-w; mv $tmp-w B
  delf 1 D | ycat $tmp-f - > $tmp-w; mv $tmp-w D
  tateyoko R | tac         > $tmp-w; mv $tmp-w R
}

l ()  # L面(Left)の９０度時計回り
{
  self 1 F | tac           > $tmp-f
  self 1 U | cat           > $tmp-u
  self 3 B | tac           > $tmp-b
  self 3 D | cat           > $tmp-d
  delf 1 F | ycat $tmp-u - > $tmp-w; mv $tmp-w F
  delf 1 U | ycat $tmp-b - > $tmp-w; mv $tmp-w U
  delf 3 B | ycat - $tmp-d > $tmp-w; mv $tmp-w B
  delf 3 D | ycat - $tmp-f > $tmp-w; mv $tmp-w D
  tateyoko L | self 3 2 1  > $tmp-w; mv $tmp-w L
}

L ()  # L面(Left)の９０度反時計回り
{
  self 1 F | cat           > $tmp-f
  self 1 U | tac           > $tmp-u
  self 3 B | cat           > $tmp-b
  self 3 D | tac           > $tmp-d
  delf 1 F | ycat $tmp-d - > $tmp-w; mv $tmp-w F
  delf 1 U | ycat $tmp-f - > $tmp-w; mv $tmp-w U
  delf 3 B | ycat - $tmp-u > $tmp-w; mv $tmp-w B
  delf 3 D | ycat - $tmp-b > $tmp-w; mv $tmp-w D
  tateyoko L | tac         > $tmp-w; mv $tmp-w L
}

f ()  # F面(Front)の９０度時計回り
{
  tail -1 U | tarr              > $tmp-u
  self  1 R | yarr              > $tmp-r
  tail -1 D | tarr | tac        > $tmp-d
  self  3 L | yarr | self 3 2 1 > $tmp-l
  ctail -1 U | cat - $tmp-l     > $tmp-w; mv $tmp-w U
  delf 1 R   | ycat $tmp-u -    > $tmp-w; mv $tmp-w R
  ctail -1 D | cat - $tmp-r     > $tmp-w; mv $tmp-w D
  delf 3 L   | ycat - $tmp-d    > $tmp-w; mv $tmp-w L
  tateyoko F | self 3 2 1       > $tmp-w; mv $tmp-w F
}
```

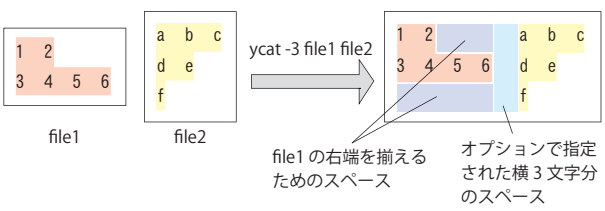
ctail は cut tail(尻尾を切る) の意味。
指定された行数だけ末行を削除する。

```
F ()  # F面(Front)の９０度反時計回り
{
  tail -1 U | tarr | tac        > $tmp-u
  self  1 R | yarr              > $tmp-r
  tail -1 D | tarr              > $tmp-d
  self  3 L | yarr | self 3 2 1 > $tmp-l
  ctail -1 U | cat - $tmp-r     > $tmp-w; mv $tmp-w U
  delf 1 R   | ycat $tmp-d -    > $tmp-w; mv $tmp-w R
  ctail -1 D | cat - $tmp-l     > $tmp-w; mv $tmp-w D
  delf 3 L   | ycat - $tmp-u    > $tmp-w; mv $tmp-w L
  tateyoko F | tac              > $tmp-w; mv $tmp-w F
}

b ()  # B面(Back)の９０度時計回り
{
  head -1 U | tarr | tac        > $tmp-u
  self 1 L  | yarr | self 3 2 1 > $tmp-l
  head -1 D | tarr              > $tmp-d
  self 3 R  | yarr              > $tmp-r
  tail -n +2 U | cat $tmp-r -   > $tmp-w; mv $tmp-w U
  delf 1 L     | ycat $tmp-u -  > $tmp-w; mv $tmp-w L
  tail -n +2 D | cat $tmp-l -   > $tmp-w; mv $tmp-w D
  delf 3 R     | ycat - $tmp-d  > $tmp-w; mv $tmp-w R
  tateyoko B | self 3 2 1       > $tmp-w; mv $tmp-w B
}

B ()  # B面(Back)の９０度反時計回り
{
  head -1 U | tarr              > $tmp-u
  self 1 L  | yarr | self 3 2 1 > $tmp-l
  head -1 D | tarr | tac        > $tmp-d
  self 3 R  | yarr              > $tmp-r
  tail -n +2 U | cat $tmp-l -   > $tmp-w; mv $tmp-w U
  delf 1 L     | ycat $tmp-d -  > $tmp-w; mv $tmp-w L
  tail -n +2 D | cat $tmp-r -   > $tmp-w; mv $tmp-w D
  delf 3 R     | ycat - $tmp-u  > $tmp-w; mv $tmp-w R
  tateyoko B | tac              > $tmp-w; mv $tmp-w B
}

u ()  # U面(under)の９０度時計回り
{
  head -1 R                     > $tmp-r
  head -1 F                     > $tmp-f
  head -1 L                     > $tmp-l
  head -1 B                     > $tmp-b
```

次ページへ続く→

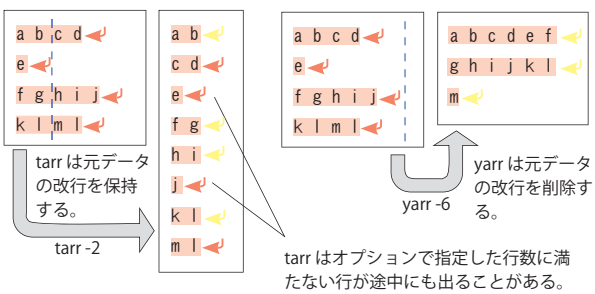
Tukubai の名コマンドたち　その３―ycat

ycat は「横 cat」の意であり、cat の横バージョンである。cat で複数のファイルを指定すれば、先に指定されたファイルの最終行の下に次のファイルを結合するが、ycat の場合、先に指定されたファイルの最右端の先に結合する。先のファイルの右端位置が各行バラバラであるなら適宜スペースを追加して崩れないようにする。
更に ycat は、結合する際に結合面にスペースを挿れるようになっている（デフォルトは１文字）。もし挿れたくなければオプションとして -0 を指定すればよいし、もっと多くしたければ大きい数字を指定する。



Tukubai の名コマンドたち　その４―tarr,yarr

tarr は「縦アレイ」、yarr は「横アレイ」の意。帳票作成向けの使い方もあるが、ここでは単純な列数調整コマンドとしての動きを紹介する。
両者共に一行あたりの列数を、オプションで指定された数に調整するためのコマンドであるが、tarr にはなるべく縦伸ばししようという性格が、yarr にはなるべく横伸ばししようという性格が与えられており、それによって動作仕様が若干異なる。
一つはオプションで列数が指定されなかった際のデフォルト値。tarr は１であり、無指定なら全要素を縦一列に並べる。一方 yarr は無限大。よって無指定なら全要素を横一行に並べる。もう一つは、改行コードの扱い。tarr は縦伸ばししようとするので元データの改行コードを消さない。よってオプションで指定した列数に満たない行が途中でも生じ得る (右図)。一方 yarr は消すので、そのようなことはない。



```
  tail -n +2 R | cat $tmp-b - > $tmp-w; mv $tmp-w R
  tail -n +2 F | cat $tmp-r - > $tmp-w; mv $tmp-w F
  tail -n +2 L | cat $tmp-f - > $tmp-w; mv $tmp-w L
  tail -n +2 B | cat $tmp-l - > $tmp-w; mv $tmp-w B
  tateyoko U | self 3 2 1     > $tmp-w; mv $tmp-w U
}

U ()  # U面(under)の９０度反時計回り
{
  head -1 R                   > $tmp-r
  head -1 F                   > $tmp-f
  head -1 L                   > $tmp-l
  head -1 B                   > $tmp-b
  tail -n +2 R | cat $tmp-f - > $tmp-w; mv $tmp-w R
  tail -n +2 F | cat $tmp-l - > $tmp-w; mv $tmp-w F
  tail -n +2 L | cat $tmp-b - > $tmp-w; mv $tmp-w L
  tail -n +2 B | cat $tmp-r - > $tmp-w; mv $tmp-w B
  tateyoko U | tac            > $tmp-w; mv $tmp-w U
}

d ()  # D面(down)の９０度時計回り
{
  tail -1 R                   > $tmp-r
  tail -1 F                   > $tmp-f
  tail -1 L                   > $tmp-l
  tail -1 B                   > $tmp-b
  ctail -1 R | cat - $tmp-f > $tmp-w; mv $tmp-w R
  ctail -1 F | cat - $tmp-l > $tmp-w; mv $tmp-w F
  ctail -1 L | cat - $tmp-b > $tmp-w; mv $tmp-w L
  ctail -1 B | cat - $tmp-r > $tmp-w; mv $tmp-w B
  tateyoko D | self 3 2 1 > $tmp-w; mv $tmp-w D
}

D ()  # D面(down)の９０度反時計回り
{
  tail -1 R                   > $tmp-r
  tail -1 F                   > $tmp-f
  tail -1 L                   > $tmp-l
  tail -1 B                   > $tmp-b
  ctail -1 R | cat - $tmp-b > $tmp-w; mv $tmp-w R
  ctail -1 F | cat - $tmp-r > $tmp-w; mv $tmp-w F
  ctail -1 L | cat - $tmp-f > $tmp-w; mv $tmp-w L
  ctail -1 B | cat - $tmp-l > $tmp-w; mv $tmp-w B
  tateyoko D | tac            > $tmp-w; mv $tmp-w D
}

# ランダムに指定回数キューブを回転させる
rand ()
{
  func=(f F b B u U d D r R l L)
  for((i=0; i<$1; i++)); do
    proc=${func[$(($RANDOM%12))]}
    echo $proc 1>&2   # 手続きを保存する
    eval $proc        # 関数の実行
  done
}
```

12 通りの回転関数名の配列を作り、それらをランダムに呼び出すことでキューブの並びを崩している。

このシェルスクリプトを実行すると**写真２**のような画面になる。キューブの全６面が横一列に表示される。左から、正面、上面、右面、背面、底面、左面の順番である。

そしてキー入力待ちになっているが、ここでそれぞれの頭文字（F,U,R,B,D,L）をタイプするとその面が 90°回転する。ただし小文字（f,u,r,b,d,l）の時は時計回り、大文字（F,U,R,B,D,L）の時は反時計回りだ。回転すれば勿論、隣接する他の４つの面の状態も変化する。ターミナルという制約上、立体的に表示することが出来ず、イメージが掴めるまでの間大変なのだが、ちょっと遊んでみよう。

しかし実は、入力プロンプトのところに解答が記してある（写真２では“RldDLRLFFuB”）。よって、この通りにタイプすれば誰でも簡単に全色を揃えることができてしまうのだが……。

尚、終了したい時は “q” とタイプする。

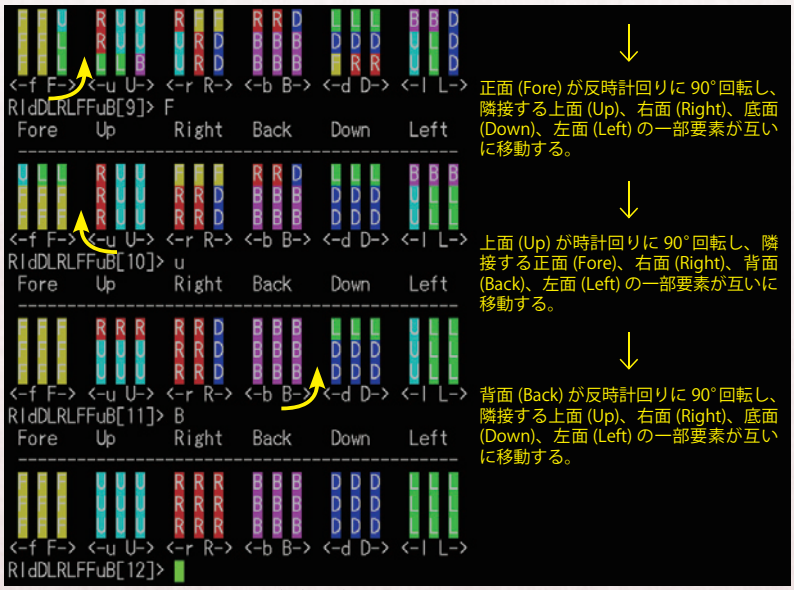


写真２．ルービックキューブプログラムの実行画面

解説

プログラムリスト（特に後者の 12 通りの回転関数）を観察してみると、self や delf 等、帳票作成でも使用したコマンドが頻出している。先のレシピでは使わなかったものの本レシピで登場した tarr や yarr というコマンドも帳票作成でよく使うコマンドである。

先のレシピ解説において「Tukubai は SQL の句を一つ一つ切り出したようなコマンドが多い」と述べたわけだが、それが行列の操作に多用されているとことが興味深い。これはつまり、データベース操作も行列操作も、数理的にはほぼ同じ概念の操作であるということを意味しているのではないだろうか。

Tukubai は、シェルスクリプトをデータベース言語化するための拡張コマンドセットというわけではない、主張する理由はここにある。多くが単機能のコマンドとして切り出されているおかげで汎用性を兼ね備えるようになったのだ。

人は情報を整理する時によく紙と鉛筆を使う。それと同様にしてコンピューターに対し、情報整理をテキストデータ・テキストファイルで行わせようというのが Open usp Tukubai をリリースした USP 研究所の提案であり、そのための要素技術が Tukubai を加えた UNIX コマンドセットと言えよう。

TechLION

For Independent Engineer

まつもとゆきひろ（Rubyアソシエーション理事長）が語る

10年先も勝負できるプログラマーとは

6回目を数えるTechLION。
100名を優に超えるであろう参加者・関係者が、
東京・六本木のイベントスペースSuperDeluxeに集った。

第一部では、オブジェクト指向スクリプト言語
「Ruby」の開発者、Matzこと
まつもとゆきひろ氏が登壇。
Rubyが
全世界で愛用されるプログラミング言語に
なりえたのはなぜか。
コンピューターとRubyの未来は。
プログラマーが
食べていくためにすべきことは
――これらの問いに持論を展開した。

お相手は、
おなじみ法林浩之氏（日本UNIXユーザ会）に、
今回から技術評論社の馮富久氏もMCに加わり対談を盛り上げた。
エンジニアである自らの経験に即した
まつもと氏のメッセージをお伝えしたい。
（聞き手：法林浩之、馮富久、まとめ：柏崎吉一）

法林：こんにちは、今日のゲストはRubyのパパ、まつもとゆきひろさんです。

まつもと：こんにちは。本日はよろしくお願いします。

法林：あ、乾杯する指令を、忘れた。いまドリンクもってきますね（ビール到着）。では、カンパーイ。いろいろおめでとうございます。肩書き、増えましたね。

まつもと：Ruby Associationは去年、一般財団法人になりました。それから、グルーヴノーツのチーフアーキテクトになりました。発表はちょうど昨日（4月11日）です。

法林：ちなみにグルーヴノーツの広報のお姉さん、ドラ娘で有名なショウジユウコさんという方です（会場：おー）。そして、まつもとさん、FSF（Free Software foundation）AWARD2011での受賞！（会場：拍手）。おめでとうございます。

まつもと：ありがとうございます。FSFのThe Award for the Advancement of Free Softwareという賞を最初にもらったのは、ラリー・ウォール。Perlの開発者ですね。それから、Pythonを作ったグイド・ヴァンロッサムも受賞しています。

法林：つまりRubyが、Perl、Pythonに並んだ！ PHPの人は、もらっていないの？

まつもと：もらっていないですね。

法林：じゃあ、PHPには、勝った（会場：笑）。

まつもと：ちなみに、なぜかこの賞をリーナスはもらっていない。それはともかく、私はFSFの受賞時に、思わず涙が出そうになりました。松江市名誉市民に選ばれた時と、喜びの質がまったく違った。FSFの選考者はRubyやフリーソフトウェアが何であり、コードとは何かを理解しています。そういう人達が「君はフリーソフトウェアで頑張って

いるね」と評価してくれた。ストールマンとLISPについて語る、なんて話は他の賞では絶対出てこない（笑）。

IT業界にとって10年先は永遠の未来である

まつもと：雑誌の連載がたまったので今度、本を出します。「コードの世界」の続編でタイトルは、「コードの未来」です。今日しゃべるネタは、それに関係します。さて本日のお題は、「10年先も勝負できるプログラマーとは」。10年先というと、IT業界にとっては永遠の未来のような気がしませんか？　予想は難しい。そこで、次のような前提を導入します。「現在から未来への変化は連続的である」「技術動向は歴史の中で比較的予測しやすい」「eXtremeに考える」です。

まず、「未来への変化は連続的である」。10年前を振り返ってみましょう。2002年といえば、RubyはRails以前です。Ruby on Railsが発表されたのが、2004年。Railsを作ったデンマークの若者は、炎上しているところにガソリン持って飛び込むような議論好きで、それによってRubyは話題にはなりましたが、当時誰もビジネスで使えると思ってなかった。『開発生産性はJavaの10倍なのか論争』があった。

法林：その頃のLLイベントで、LLを仕事で使いたいか、とアンケートを尋ねると、ヘビープログラマーほど仕事で使えていないことが分かった（会場：笑）。

馮：Rubyはレンタルサーバーでも、まだ浸透していなかったですよね。そもそもサーバーにRubyがプリインストールされてなかった。

まつもと：PerlとPHPはあったんですが。あの頃、「Rubyを使えるレンタルサーバー一覧」といった記事があったと記憶しています。「さくらはインストールしないと使えない」とか。ただRubyの根本的なところは今も10年前と変わっていないんです。一方、我々を取り巻くITインフラもそう。メールやWeb。ツイッターはないけどチャットはあった。基本的な道具立ては揃っていた。そう振り返ると、コンピューターの世界では年々劇的な変化が減っているように思います。

とはいえ時々、不連続なイノベーションは起こります。1903年にライト兄弟が動力付きの有人飛行機を300m飛ばした3年後にはもう世界の空を飛行機がガンガン飛んでいた。ITの世界でも、今から5年後に量子コンピューターが登場すれば、今あるプログラムが全部パーになるかもしれません（笑）。でも今日の予測では、あくまで連続的な変化しか起きなかったと仮定します。

次に、「技術動向は歴史の中で比較的予測しやすい」という前提について。気体分子一個一個の動きは予測できなくても、全体としての振る舞いは予想できる。しかし外乱の影響を避けるために、閉じた空間での観察に限定しなければ正確な予測が困難です。歴史や人間社会も同じように、誰かが未来を予想できると、筋書きが変わってしまうため、正確な予測は不可能になる。子供のころ読んだSFにそう書いてあり、期待していた分、少しがっかりした記憶があります。そもそも歴史においては個人の影響は不可避で、徳川家康が出てこないと歴史は進まない。ただ誰がいつ何をするかを前もって知るのは難しい。しかし、コンピューターを含む技術全体については、俯瞰すると10年先を予測できる可能性がある。例えば、ムーアの法則です。

法林：半導体の集積密度は、2年で倍になるというヤツですね。

まつもと：発表されたのが1965年、東京オリンピックの頃です。半導体でできた電子回路が微細化し、チップが小型化する。処理性能、記憶できる容量、ネットワーク帯域なども指数関数的に増える。自動車で、リッター走行距離10kmが2年後に20km、さらに2年後に40kmなんてあり得ない。陰で技術者の人達がムーアの法則を維持させようと頑張ってきたから、過去40年少なくとも予言通りになっている。永遠に続かなくても、まだしばらくは当たり続けるかもしれない。eXtreme Programmingという開発手法があります。自分が書いたコードを他人にレビューしてもらうと思わぬ発見や気付きを得やすいという考え方ですが、未来予測についても常識に囚われず大胆に、eXtremeにやりましょう。

ソフトウェア開発の主戦場が変わる

まつもと：中学の時に手にした両面倍密度の320KBフロッピーディスク。記憶できる空間は無限に広く感じられ、「この一枚で一生やっていけるかも」とさえ思った。でも、今日320GBのハードディスクは珍しくない。30年間で百万倍以上。メモリもさらに大容量化しています。320GBの次世代フラッシュメモリが何年か先に出るかもしれない。主記憶上にデータがいつまでも揮発せずに残っていて、フラットにアクセスできるとか。

法林：ありえそうですね。

まつもと：CPUとDRAMの処理速度の差を緩和するためキャッシュを持たせる構造が考案され、プログラマーもそれを意識してコードを書いていた。しかし、ソフトウェアアーキテクチャが変われば不要になります。

馮：プログラマーには大きな変革ですね。

まつもと：コンピューターは性能とネットワーク帯域の広さにおけるバランスの変化で、集中処理と分散処理を振り子のように行き来します。オンライン処理、C/Sシステム、サーバー同士をつないだWeb、クラウドができた。揺り戻しは5年くらいの割と短いスパンで起きる。帯域と性能のバランスの中で、どういうデザインが最適かとプログラマーは考える必要があります。

同時に気になるのが、データの入り口がどこになるのかということ。口にするのも気恥ずかしいんですけど、ビッグデータというBuzzワードがありますね。ビッグデータは自動的に生まれてこない。どこかの生データを取り込んである種のDBに放り込む。Hadoopで解析するデータをどこから取ってくるのか。ユーザーの行動履歴などのログ、

車や家電で使われる組み込み系経由のデータかもしれません。その辺りが今後のソフトウェア開発における主戦場の一つになっていくと思います。

“70 万コア”を使いこなすプログラミング言語が必要

まつもと：メインストリーム PC の価格推移に比較して、マイクロコントローラが低価格化しています。これが組み込まれて、家電のフリをしたコンピューターがどんどん出ていますね。テレビ、音楽プレイヤーや電話、車の形をしたコンピューターがある。開発言語はアセンブラではなくて高級言語。OS は Linux。つまり、デバイスの PC 化です。小さなコンピューターが低価格化し、そこで使われるソフトの割合が増えてくれば、高い開発生産性がいままで以上に必要とされます。

法林：Ruby で書いています、とか。

まつもと：Ruby の採用はまだないですが、今後使ってもらえるように頑張っています。

法林：10 年前のレンタルサーバーみたいな感じですね。

まつもと：PC のスパコン化にも注目です。すでに 20 年前のスパコンと同程度の性能をもったノート PC が手元にあります。この調子で進むと 10 年後の電気屋さんで買えるコンピューターは、1024 コアになってもおかしくない。神戸にあるスーパーコンピューター「京」は、8 万 CPU です。1CPU が 8 コアなので、全部で 64 万コアくらい。その類推でいうと 10 年後のノート PC に 70 万コアくらい搭載される可能性がある。1024 コアは笑い話ではありません。ただ、膨大なマルチ / メニーコア向けのプログラミング言語の開発はこれからです。CPU1 個にやらせている仕事をいかに分散するか。4 コアの CPU であれば、コア 1 にこの仕事、コア 2 にあの仕事……と割り振りできますが、数十万コアになると「この仕事は、13522 番目のコアに」などキリがない。抽象化レベルがもう一段高くなる必要がある。Linux の内部では「コア ID は 84722 番」とかコア単位での指定が可能ですが、開発言語レベルでは人間が理解できないと使い物にならない。また、ノード数がこれだけ膨大になるともはや人手によるリソースの管理は不可能です。言語と同様に、ハードや I/O を隠蔽するレイヤーが必須。コンセントから得ている電力エネルギーがどこで発電されたものかが利用者の目に見えないように、クラウドも抽象化されていくと思います。

10 年先も通用するサイエンスを身につける

まつもと：僕の未来予測はだいたいこんな感じです。そういう中で、プログラマーが生き延びるにはどうするか。「10 年先でも変わらないであろう、重要なことを押える」のが大切だと思います。たとえば、数学はあと 10 年で反故になることはない。数千年前から続く、長い歴史があります。やっておいて損はない。そう言いつつ僕は、数学が苦手なんです。高校の時、10 段階評価で 1 をもらったくらいですから。

法林：得意科目は何でしたか？

まつもと：国語と英語です。

法林：やっぱり言語が得意なんだ。

まつもと：でも、MIT でやった 2002 年の LL では英語が喋れずひどい目に会った。

法林：僕も英語は苦手です。プロレス用語を除けば。

まつもと：もう一つ押えておきたいのが、コンピューターサイエンスです。この分野は未成熟ながらここ何十年か続いているので、それなりの歴史がある。この辺は押えた方がいいでしょう。数学にしてもコンピューターサイエンスにしても、要するに「サイエンスで物事を考えよう」ということです。事実に基づいて、推論し、それを検証する。この考え方は 10 年先も通用するでしょう。その時々のテクノロジーのトレンドに適用することで、地に足のついたエンジニアになれる。と同時に、自分の持っているものに満足しないで、新しいものにチャレンジできる柔軟な精神を下支えしてくれます。

法林：年をとると、いろいろと億劫になりますからね。

まつもと：苦手を克服するなら、得意な部分を伸ばした方がいい、という人もいます。その辺のバランスは人それぞれだと思います。狭い部分に突っ込んでいけるのも長所です。かといって新しいトレンドを敬遠するのはどうか。大人げない大人になろう、ということです。分別のあるふりをしていると、いつしか頭ががんじがらめになってしまいます。

法林：中学生コミッターには負けないとか（会場：笑）。

まつもと：その彼も、もう中学卒業です（笑）。また、つい先日も「まつもとに負けない、勝つ」といったツイートに思わず熱くなったら、至るところで「大人げないまつもと」というツッコミを頂きました（会場：笑）。

法林：ちゃんと実践しているんだ、自分の理論を（笑）。

まつもと：分別云々と言っていますが、何でも手を出せばいいという訳でもない。私の場合、いまさら言語以外の分野に首を突っ込んでも仕方ない。とはいえ、プログラマーとして振る舞う時は、皆さんぜひ大人げないプログラマーになりましょう。社会生活では別ですけど。

コミュニケーション能力も結構大事です。コンピューターをやらないうちの奥さんが『あなた、いつも機械に向かって』とボヤいている。見た目はたしかにそうです。でも、誰かの書いた意見を読んだり、それに返事したり、時に議論したり、講演の準備をしたり。要するに人間相手の仕事をしている。

法林：機械相手ではなく。

まつもと：Ruby を設計する時も、ユーザーはどう感じるか、自分はどう感じるか、と機械を通じて、人とやりとりしている。だが、奥さんには機械とやりとりしているようにしか見えない。リア充礼賛ではありません。そもそも私は非リア充の典型のような人間で

す。よく結婚できたと思っている。しかし、ある種のコミュニケーション能力、人とやりとりしたり、誰かのことを考えたりするのは面白い。人間的な活動に対する興味を持つことがプログラマーにとって重要かつ非常に有効な能力と思います。もし『コンピューターをどう制御するしか関心がありません』みたいな感じだと、プログラマーとしての伸び代が小さくなってしまうと思うんです。

普通の人が普通じゃない人になる

まつもと：これからは、個人の持っている評価が重要になると思います。そもそもオープンソース、フリーウェア、というのは名誉経済ですね。「伽藍とバザール」を読まれた人はご存知かもしれないけど、オープンソースの世界は、○○に勤めているから偉いとか、その人の所属なんて通用しない。『まつもとという奴は、何とかというソフトウェアを作ったから、プログラミング能力があるに違いない』と思われて、仕事を任せてもらえたり、チャンスをもらえたり、有利なポジションを得たり、といった場面に巡り合える。エンジニア個人で勝負できる。俗っぽくいうと「名声 w」。自分の考え、感じたことをブログやツイッターに書く中で、あの人は面白い、と思われれば外部の評価が高まる。自分の作ったソフトウェアを公開すれば、『あの人は、こういう Ruby ソフトウェアを作れる人だ』と見てもらえる。積み重ねです。

法林：まつもとさんも長いですよね。

まつもと：Ruby は 20 年くらい。その前に emacs を 4 年ほど。ほかのも通算すると 30 年弱くらいやっている。2001 年頃からは日本や海外で年 20 回くらい講演していた。そのおかげで、今のポジションに至っていると思っています。個人的な能力が人より 100 倍優れているわけではない。天才プログラマーだと思わない。でも、普通の人がとる普通の行動が、継続していく中で外部の評価につながり、「普通じゃない人」になれる。普通じゃないのがよいのかどうか、という意見もありますが、私は普通じゃない方がいいと思う。小学一年生の時、担任の先生から「どうしてあなたは他の子と同じことが出来ないの？」と叱られたことがあります。

法林：僕は小学生の時に「法林は、人がやらないことをやる人だ」と言われました。思い返すと、大人になってから JUS の活動とか、みんながやりたがらないことをやっている（会場：笑）。

まつもと：外部評価を高めることをこれから 10 年続けると、10 年後もプログラマーとして食べていけると思います。私は 20 年 Ruby やってきて、一度も嫌だと思ったことはない。自分でいうのもなんですが、私は異常なほどのプログラミング言語オタクです。好きでやっている。努力というより、継続的な行動です。その源泉にあるのがモチベーション。自分のモチベーションを 10 年、20 年保てる人は、先々勝利できるのではないでしょうか。モチベーションの出どころは何か。私の場合は、「楽しさ」です。ものすごく楽しい。なぜか。プログラミングに万能感があるからです。

法林：そういうコンセプトが Ruby にも反映されている。

まつもと：5 歳くらいの子供は万能感にあふれています。「やればできるもん」と言う。小学生くらいになると、少し挫折を知る。我が家に小学 2 年生の子供がいるのですが、周りの子供を見て段々と万能感が失われていくのが分かる（笑）。しかし、プログラミングに年齢は関係ない。コンピューターを使って在りし日の万能感を取り戻せます。モチベーションが高まり、楽しくなって、継続しているうちに、ひとかどの人物になれる日が来るかもしれない。IT というのは、ソフトがないと動かない。プログラマーの皆さんは世界を変える力を持っている。プログラミングは、世界改革の源泉です。ぜひ皆さんに世界を変えて頂きたい。

法林：それでは最後に、質問を受け付けたいと思います。どうぞ。

松浦（本誌編集長）：優れた OSS は世の中に数多く出ていますが、その中でユーザーが実際に使ってくれるレベルに到達しているソフトは限られています。OSS 開発者の多くは、どうやったら使ってもらえるかと悩んでいます。Ruby は今や世界中で使われる OSS になりましたが、何が成功の要因だったと思いますか？

まつもと：私もこう幾度も言われました。『Ruby ？ ああ知っていますよ。使っていませんが』。もし、そこで開発をやめていれば Ruby は終わっていたでしょう。でもやめなかった。Ruby って、愛され言語なんですよ。Ruby を知っている人、イコール好きな人が多い。好きだと言ってくれる人がいるので、仕事を続けていたら人が増えてきた。95 年に一般公開してから 10 年間続けていた結果がビジネスでの利用拡大でした。愛されるプロダクトを作って継続することが、成功の秘訣だと思います。僕自身はプロモーションを全然していません。

質問者 A：まつもとさんは、Ruby を全世界の人に使ってほしいと思って作っていたのでしょうか。それとも自分のために開発したのでしょうか。

まつもと：僕は、色々な言語を使う中で、自分が欲しい言語、使って満足のできる言語が欲しくて Ruby を作り始めました。

法林：質問は尽きないですが、この辺で時間が参りました。まつもとさん、本日はありがとうございました。

まつもと：こちらこそ、ありがとうございました。

TechLION はメインゲストの他にも、とても魅力的なゲストをお呼びしています。Vol.6 では後藤大地さんや、角俊和さん、閑歳孝子さん。ブログや写真などの情報は、WEB サイトにありますので覗いてみてください。そして、可能であればライブにも是非遊びに来てください。とても楽しいですよ・・・。（TechLION 事務局スタッフより）

http://techlion.jp/

シェルスクリプト大喜利 第五回

司会：『もっと吹く』編集長・みかん

皆様こんにちは、はじめまして。3か月のご無沙汰、シェルスクリプト大喜利（略してsh大喜利）のコーナーのお時間です。あたくし司会進行役のもっと吹く編集長みかんです。他ページでは結構脂っこいなこと言ってたりして「人格の使い分け大丈夫？」とかひやかされとりますが、大丈夫。あぁ……この変態力作コード、絶対**ウシシとか言いながら書いてるこしたな。**と、投稿作品を見てると自動的にこの人格に変わりますから。いゃいゃ、読者参加コーナー面白いわ。

ということで、まずは本コーナーのシステムをご説明！

シェルスクリプト大喜利とは

シェルスクリプト大喜利特有のルール

一、sh大喜利はクイズやテストではありませぬ。なので決まった答えというものはないのです。あえて言うなら面白いスクリプトが正解！

二、面白いスクリプトとは例えば、こんなもの。

イ、人が考えつかない意外性がある

ロ、美しい or 芸術的 or 記述がシンプル・短い or 高速

ハ、アイデア・こだわりが光る

ニ、ネタになるようなバカバカしさ、くだらなさがある

などなど、ただし最後のは段位強制返還の恐れありよ。:-)

三、スクリプト動作環境はLinuxとします。そして、特に断りなき場合は、Linux JM(http://linuxjm.sourceforge.jp/)に記載されているコマンド及び機能のみ使用可能とします。これは多くの人が楽しめるようにするためなのです。(但しJMに載っているので、**Cシェル系での回答もOK！**)

四、sh大喜利はシェルスクリプトを披露する場なので、**PerlやRuby、Pythonなどは使っちゃダメ**です。そもそもJMにも載っていません。逆にシェルスクリプトにとって不可欠なawkやsed等はOKです。JMにもありますし。でも、よっぽど面白ければ、なきにしもあらず？

ルールもおさらいしたところで、それじゃイッてみよう。

五、今回からOpen usp Tukubai(http://uec.usp-lab.com/)も使用OKとしました。ただし、それなりに美しく書かれていないと採用はキビシいですよ〜。

本番開始

<第一問>

foldという、指定文字数に達したら改行するコマンド。これ見てふと思いました。

「fold コマンドで商と余りを求められないかな？」

というわけで、２つの引数を受け取り、fold コマンドを使って商と余りを求めるスクリプトを書いてください。

こんなコマンドがあるのをアタクシ前回初めて知りまして。なので、まだこのコマンドにどんな実用性があるのか想像できず、「割り算の商と余りを求めるのに使えるかな？」とか馬鹿馬鹿しいこと思いついたんです。それで大喜利のお題にしてみたんですが、投稿者の皆さん。こんな馬鹿馬鹿しい問題に付き合ってくれてありがとうございます、ホント。

◎gori.shさんの回答

```
1 #!/bin/bash
2 seq -s- 0 $1|tr -d 0-9|fold -w$2|awk ´END{b=(l=leng
  th)!=´$2´;print NR-b,b*l}´
```

添付コメントによれば、文字数を極力削る方針で書いたそうで、AWKのlength関数はAWKで唯一カッコを省けるということを本誌前号で知ってそこも削るこだわり様。

んー、本誌を見て勉強したってところがエラい。(←贔屓)よし、**二段進呈**しちゃおう！これで四段だ。

◎emasakaさんの回答

```
1 #!/bin/sh
2 printf "%0$1s¥n" | fold -w $2 | sed "¥$s/.*/echo -n
  ´&´|wc -c/e;s/$2/¥¥
3 0/" | sed -n ´$!=;$p´ | tail -n 2
```

「GNU sedじゃないとだめかも」ということで、確かにその通りですよ。でも、注目すべきはAWKを使ってないところ。この問題、AWKを使えば簡単だろーなーと思ってたので、AWKを使わない回答を実は待ってたのですよ。

エラい！さてはアタクシの心を読んでウシシとかいいながらコードを書いていたな！よし、これも**二段進呈**。

◎浅井 陽一さんの回答

```
1 #!/bin/bash
2
3 if [ $# -ne 2 ]; then
4     echo "このシェルスクリプトは２つの引数を取ります。"
5     echo "１つめの引数は対象の数です"
6     echo "２つめの引数は割る数です"
7     exit
8 fi
9 for i in `cat /dev/urandom|tr -cd "[:alnum:]"|head -c $1|fold -w $2`
10 do
11     AMARI=`echo -n $i|wc -m`
12     if [ ${AMARI} -eq $2 ];then
13         LINE=`expr ${LINE} + 1`
14         AMARI=0
15     fi
16 done
17 echo "計算結果 : "${1}" / "${2}" = "${LINE}" ... 余り"${AMARI}
```

う〜ん、他の投稿と比べると行数は長いんですけど、AWK を使っていないということで確かにこれも密かに求めていたコードになってるのです。

シェルプログラミングの達人目指して頑張ってもらいたいなという応援も兼ねて、**初段授与**。

◇ ◇ ◇

今日は make 祭りが控えてるので、はい次のお題行きます。

＜第二問＞

本誌前号で rs というコマンド出てきました。でもこれ *BSD 特有のコマンドです。そこで rs コマンド相当品作ってください。ただし、次の簡易仕様とします。

- **ターミナルの縦横サイズの自動検出はしなくてよい**
- **データは標準入力からのみ受け取る**
- **-t オプション指定時の動作（行と列を転置）とする**
- **引数は rows と cols、２つの数値だけを必ず貰える**

前回の大喜利では Linux にあって *BSD には無い tac コマンド相当の汎用品を作ってもらいましたが、今回は逆です。それでこの rs コマンドというのはどういう動きをするものかというと ls コマンドで説明すると分かり易いですよ。ls でファイルを表示すると、アルファベット順にソートされた後、それが左の列から縦に並びますな。ファイルが 9 個、それが横に 3 列に表示されると、左の列に最初の 1~3 番目、中央列に 4~6 番目、右の列に 7~9 番目に並ぶというアレをやってもらいたいわけです。簡易仕様で。（ソートも無しで）

◎emasaka さんの回答

```
1 #!/bin/sh
2 sed "1s/^/paste <(echo '/;$1~$1{N;s/¥¥
3 /') <(echo '/}" | sed "¥$s/¥$/')/" | bash
```

2 問続けての採用。そしてこちらも「GNU sed じゃないとだめかも」とのことで、その通りでした。

う〜〜〜ん、一緒に添付してくれたサンプルデータだと確かに正常なんだけど……。8 文字（恐らくタブ文字数）以上の長さのデータを食わせると動かないのが残念なところ。おしい！……でも、サービスして**一段授与**。これで三段だ。

◎a_shima さんの回答

```
1 #! /bin/sh
2 mkdir /tmp/left
3 nl | head -n $(($1*$2)) | awk '{printf("/tmp/left/!%05d!%s¥n",$1,$2)}' | xargs touch
4 w=`ls -1 | awk '{l=length($0);m=(l>m)?l:m}NR%'$2'==0{m=(l>m)?l:m;w+=m+2;m=0}END{if(m>0)w+=m+2;print w-1}'`
5 ls -w $w -C /tmp/left | expand | sed -e 's/^![0-9]¥{5¥}!//' -e 's/  ![0-9]¥{5¥}!/  /g'
6 rm -rf /tmp/left
```

添付コメント「ls コマンドの動きがそれですよということだったので、ls（bash 版オンリー）にやらせてみました。大抵の場合動きます。Copyright に対する Copyleft の如く、rs に対する ls ですね」ということで、何と ls を無理矢理使った解答が届きましたよ。

いやぁ、そう来たか……。敢えて狙ったんだろうけど、すっごく無理矢理感が伝わってくるぞ（Copyright のくだりもね）。ls ってファイルになっているものしか表示しないから同名のファイルを作って、さらに ls はソートをかけるからファイル名の手前に番号を振ることでソートを回避けてるし……。意地でも ls にやらせようと、よくもまあここまで執念を燃やしたものだねぇ。グレイト！**三段授与**だ！！

◎さいたま帝国さんの回答

```
1 #! /bin/sh
2 head -$(($1*$2))                                    |
3 awk '{print}END{for(i=NR;i%'$1'>0;i++)print"@"}'    |
4 yarr -$1                                            |
5 tateyoko                                            |
6 sed 's/ @$/  /'                                     |
7 keta
```

待ってました。Tukubai コマンドを使った投稿ですね。ありがとうございます。「Tukubai 使ってみました。tateyoko を使うんだろうなというとこまでは容易に想像つきましたが、後が結構難しかったです。こんな感じでよろしいでしょうか」とのコメントが付いてましたが、はい、結構です。

慣れない中、使ってくれてありがとう。指定された行数×列数になるように元データの行数を揃えて、yarr で行列化した後で tatetoko 使って転置。最後に keta で表示を揃えるというわけね。バーティカルバー（パイプ記号、"|"）を縦に並べる独特の記述にもしてくれていて完璧だ。初 Tukubai 採用作品でもあるし、記念に**二段授与**しよう。

◇ ◇ ◇

さてさて、次は make のお題。力作と予想の斜め上を行く作品と、**お約束の作品が多数**届きましたよ。

<第三問>
make コマンドで「普通そんなことしねーよ」というちょっと変わった使い方を披露してください。BSD make でも GNU make でも OK。

さあさあ、Maker の皆様、おまちどうさまでした。本日一番のお楽しみ、make コマンドのお時間がやってきました。

シェルスクリプトで何でもこなすという熱狂的シェルスクリプティストがいるのと同様に、世の中には make コマンドで何でもこなすという熱狂的メイカーがいるのです。例えばシェルスクリプトだと "&" を使ったり、或いはパイプを繋げたりして並列処理をさせられますが、make コマンドには -j オプションというのがあって、コイツをうまく使えばやっぱり並列処理ができるなど、全然負けず劣らずなのです。

ではまず、お約束というヤツからいきましょうかね。

◎F-joy さん、さいたま帝国さん、t_kawai さん、うみねこさん、doslov さんの回答

```
> make love
make: don't know how to make love.  Stop

> make love
Not war.
```

これは BSD 版 make コマンドの挙動ですね。このジョークの意味を理解するにはまず、make love の意味を知っておく必要があるわけですが、各自**英和辞典で調べておくよーに**。

それで引数に示されたターゲットが定義されていない場合、BSD make は基本的に "don't know how to make ~" ってエラーメッセージを出すわけですが、そこに出てくる make love という熟語を見て「ウブなヤツめ」とほくそ笑むのがお約束。ところが最近のバージョンで、それに対抗して "Not war." とか返してくるわけです。"Make love, Not war." についても検索すれば意味わかります。

というわけで投稿ありがとう。来ると思ってたよ！！しかもこんなに多数。有名なネタだから**各自一段だけ授与**ね。よってさいたま帝国さんが三段、他の皆さん初段ね。

次は力作いってみましょう。

◎K さんの回答

```
Makefile:
 1 PS2PDF?=    ps2pdf14
 2
 3 INPUTSIZE?= A4
 4 MEDIASIZE?= A3
 5 POSTERSIZE?=    A0
 6 POSTER?=    poster -v -i${INPUTSIZE} -m${MEDIASIZE
   } -p${POSTERSIZE} -o
 7 #"-o" option should be located at the tail.
 8
 9 TGIF?=  tgif -a4 -print -eps -reqcolor -one_file_p
   er_page -page
10 #"-page" option should be located at the tail.
11
12 #<file name>-<page number><l for Large Format Prin
   ter>.pdf
13 TARGETS2=   poster-1.pdf poster-2.pdf
14 TARGETS?=   ${TARGETS2} poster-1l.pdf poster-2l.pd
   f
15
16 TMPMAKEFILE?=   Makefile.tmp
17
18 all:
19     -@rm ${TMPMAKEFILE}
20     @echo ".SUFFIXES:   .pdf l.pdf .ps .eps" > ${T
   MPMAKEFILE}
21     @echo >> ${TMPMAKEFILE}
22     @echo ".epsl.pdf:" >> ${TMPMAKEFILE}
23     @echo "  ${PS2PDF} ¥$${.IMPSRC} ¥$${.TARGET}" >
   > ${TMPMAKEFILE}
24     @echo >> ${TMPMAKEFILE}
25     @echo ".ps.pdf:" >> ${TMPMAKEFILE}
26     @echo "  ${PS2PDF} ¥$${.IMPSRC} ¥$${.TARGET}" >
   > ${TMPMAKEFILE}
27     @echo >> ${TMPMAKEFILE}
28     @echo ".eps.ps:" >> ${TMPMAKEFILE}
29     @echo "  ${POSTER} ¥$${.TARGET} ¥$${.IMPSRC}" >
   > ${TMPMAKEFILE}
30     @echo >> ${TMPMAKEFILE}
31     @echo "all: ${TARGETS}" >> ${TMPMAKEFILE}
32     @echo >> ${TMPMAKEFILE}
33 .for i in ${TARGETS2}
34     @echo "${i:C/pdf/eps/}: ${i:C/-[1-9][0-9]*.pdf
   /.obj/}" >> ${TMPMAKEFILE}
35     @echo "  ${TGIF} ${i:C/.*-([1-9][0-9]*).pdf/¥1/
   } ¥$${.ALLSRC}" ¥
36         >> ${TMPMAKEFILE}
37     @echo >> ${TMPMAKEFILE}
38 .endfor
39     make -f Makefile.tmp
40 #   -@rm ${TMPMAKEFILE}
41
42 clean:
43     -@rm *.eps *.ps *.pdf ${TMPMAKEFILE}

poster.obj:
 1 %TGIF 4.1.45-QPL
 2 state(0,37,100.000,0,0,0,16,1,9,1,1,0,0,1,0,1,0,´C
   ourier´,0,80640,0,0,0,10,0,0,1,1,0,16,0,0,2,2,1,1,
   1056,1497,1,0,2880,0).
 3 %
 4 % @(#)$Header$
 5 % %W%
 6 %
 7 unit("1 pixel/pixel").
 8 color_info(11,65535,0,[
 9     "magenta", 65535, 0, 65535, 65535, 0, 65535, 1
   ,
10     "red", 65535, 0, 0, 65535, 0, 0, 1,
11     "green", 0, 65535, 0, 0, 65535, 0, 1,
12     "blue", 0, 0, 65535, 0, 0, 65535, 1,
13     "yellow", 65535, 65535, 0, 65535, 65535, 0, 1,
14     "pink", 65535, 49344, 52171, 65535, 49344, 521
   71, 1,
15     "cyan", 0, 65535, 65535, 0, 65535, 65535, 1,
16     "CadetBlue", 24415, 40606, 41120, 24415, 40606
```

```
   , 41120, 1,
17      "white", 65535, 65535, 65535, 65535, 65535, 65
   535, 1,
18      "black", 0, 0, 0, 0, 0, 0, 1,
19      "DarkSlateGray", 12079, 20303, 20303, 12079, 2
   0303, 20303, 1
20 ]).
21 script_frac("0.6").
22 fg_bg_colors(´black´,´white´).
23 dont_reencode("FFDingbests:ZapfDingbats").
24 objshadow_info(´#c0c0c0´,2,2).
25 page(1,"",1,´´).
26 box(´black´,´´,64,64,384,320,0,1,1,0,0,0,0,0,0,´1´
   ,0,[
27 ]).
28 page(2,"",1,´´).
29 oval(´black´,´´,64,64,384,320,0,1,1,1,0,0,0,0,0,´1
   ´,0,[
30 ]).
```

動作には tgif、poster コマンド、Ghostscript が必要。その環境で上記 2 つのファイルを置いて make するとトンボや図形の描かれた PDF が 3 種類できます。

一発ネタというわけではなく、Makefile の文法を巧く活用した実用的なものなのだけど、力作だったので採用。コメントによればもっとエレガントに書きたかったとのことなんだけど、make を使うこと自体が結構エレガントだ。ありがとう、**初段授与**。そのうち、make VS. Bash とかやりたいねぇ。

◎ゆきとさんの回答

```
Makefile:
 1 cat:
 2   @awk ´BEGIN{ORS=""} {printf(""); print $0; print
   f("¥n")}´
 3
 4 cat^2:
 5   @printf "make:¥n"
 6   @awk ´BEGIN{ORS=""} {printf("¥t@printf ¥""); pri
   nt $0; printf("¥134n¥"¥n")}´
 7
 8 cat^3:
 9   @printf "make:¥n"
10   @printf "¥t@printf ¥"make:¥134n¥"¥n"
11   @awk ´BEGIN{ORS=""} {printf("¥t@printf ¥"¥134t@p
   rintf ¥134¥""); print $0; printf("¥134134n¥134¥"¥1
   34n¥"¥n")}´

使い方その1(cat一段):
> cat hoge | make cat

使い方その2(cat二段):
> cat hoge | make cat^2 | make -f -

使い方その3(cat三段):
> cat hoge | make cat^3 | make -f - | make -f -
```

これはまた一発ネタですよ。添付コメントによれば「make コマンド自体も標準入力を Makefile として受け取れるので、Makefile を作る Makefile を作る Makefile……、というのを多段化しました。やっていることは単なる cat です。ただしダブルクォーテーションを含むテキストは処理できませんが」とのことですよ。

いやぁ〜、よくやるねぇ……。途中途中で吐き出される Makefile を眺めると、printf を作る printf を作る printf……になっていてこれが 1 つの make を通るたびに外れていくという動きになってるな。その馬鹿馬鹿しさに敬意を表して**一段授与**。これで二段だ。

◇ ◇ ◇

というところで本日の大喜利はこれにてお開き！読者の皆さん、投稿してくれた皆さん、ありがとうございました。

投稿大募集!!

次回のお題

一、「幸運数」と呼ばれる自然数があります。100 以下の幸運数を全て求めるシェルスクリプトを書いてください。答えを出すのが早いか、または見応えあるソースコードで。

二、9 行 9 列、各列がスペースで区切られた九九のテキスト表ファイルがあります。この表を標準入力から読み込み、x 行目 y 列目にあるセルを含めてその下のセル全てを、上に z 行詰めるシェルスクリプトを書いてください。
その際、詰めたことでできた空セルは "@" としてください。尚、x,y,z はこのシェルスクリプトの引数として渡ってくることが約束された整数とし、かつ値の範囲も 1 ≦ x,y ≦ 9、0 ≦ z < x であることは保証されているとします。

三、cat コマンドの作者が遺言を残しました。
「私の人生における最大の後悔は、dog コマンドを作り忘れてしまったことだ。あぁ誰か作ってくれぇー。ガクッ」
さぁ、一体どんなコマンド？

投稿のしかた

お題への回答は、お名前（ペンネーム）、回答したいお題番号、回答スクリプト、簡単な補足の四点セットで下記の宛先へ！一人何問でも何個でも回答可です。尚、次回締め切りは**9月3日（月）午前 0 時**とします。しかもその間は何度でも回答の修正を受け付けます。

お題もどしどし送ってくださーい

それからお題も大募集。考えてくれた方にも段位を授与します。自分で出題して回答するのも、今のところ可！

投稿先

どちらも投稿先は、mag@usp-lab.com です。**前回からアドレスが替わってます**んでご注意！締切前にこの記事を見てしまった皆さん、自慢の奥義をここに披露してください。

天地概況① ―奈須螢路―

この本の読者の皆さんは技術系で、かつ管理者の責にある方が多いのではと想像しております。今やＩＴは会社の合理的運営に不可欠となり、技術職は経営に直結する重要なポジションとなっています。そこでこのコラムでは技術と経済の視点から、世相を観測し未来を考えていこうと思います。どうぞお付き合いください。

先日、日産で「イチロー・ニッサン」キャンペーンを仕掛けた元営業のＩ氏に伺ったお話です。まだ無名だったイチローの起用に会社は反対。当時もっと売れていたバンチ佐藤の方を起用しようとしたそうです。Ｉ氏は「彼は今流行りの選手のスタイルではないが、近々にも必ずスター選手になる」と信念をもって反論したそうです。結果はご存じの通り。「かわらなきゃ」キャンペーンで、日産はどん底から浮上していきます。

価値感が相対化し、嗜好が細分化していく中でこそ、自分の考えに信念を持って行動することが大切です。最初は人々の理解を得られなくても後に高い評価を得たという事例は昔から世に存在します。自動車業界の古典的事例としてはフランス、シトロエン社の大衆車第一号「２ＣＶ」の誕生物語があります。

大戦前の１９３５年、南仏郊外で人力頼みの農民の生活を見た経営者は「需要はここにある」と車を大衆に供する事を思いつき、過酷な生活を変える安価な超小型車こそ、破綻した会社を救う目玉商品になると確信したのです。ただ、その課題は前例が無く、「普通車の半額くらいで、50キログラムのじゃが芋と夫婦を乗せて60キロ出せ、荒れた未舗装路を疾走しても生卵が割れない快適さで、１Ｌで30キロ位走れ、冠婚葬祭用の服や、山高帽でも乗車可能な広さで、女性も簡単に運転できる」というものでした。それでもナチスの妨害をも掻い潜り、技術陣はみごとこれに応える小型車を完成、ついに48年仏最大の自動車ショーで発表したのです。

ところが、待っていたのは大衆の「嘲笑」でした。人々は２ＣＶを見て「醜いアヒルの子」と言い、マスコミは缶切りを付けて売れと罵り、詩人も芸術家もこき下ろしました。

しかしそれが後にはどうでしょう。名車どころかフランス自体を表す象徴となり、世界の人々に親しまれる国民車となります。日本では映画「ルパン三世」に登場、ヒロインの少女が山道を２ＣＶで疾走し逃げ回る名シーンが生まれました。宮崎駿監督も自社名を「２馬力＝２ＣＶ」とする程の愛好家です。

徹底的に需要分析し生まれたデザインと技術、余分な物を全て削ぎ落とした合理性、必然性による造形は、後に「美しい・かわいい」と評価が変わり、「田舎にも都会にも若者にも年寄りにも似合う」と世界中から絶賛されます。信頼性はラリーにおいても発揮され、若者を世界一周の冒険へと誘いました。信念が単なる思い込みではなく技術的に合理的で、人々のニーズに応えるためだけに生み出されたものなら、やがては市場が受け入れることが分かります。

さて今、私達の取り組みは、合理的で必然的でしょうか。技術的に枯れていてもシンプルで必要な機能を発揮できるデザインでしょうか。思い込みに陥ることのないよう、徹底チェックの姿勢を貫き、日々積極的に生きてこそ、不況の世界を好転させていけると思っています。

【著者プロフィール・なすけいじ】
外部役員、監査、プロジェクトリーダーとして様々な計画に横断的に参加するソリューションプランナー。§このコラムがあなたのアンテナの一つとして、考えをまとめたり見つめなおすきっかけの一つとしてお役に立てれば幸いです。

USP MAGAZINE vol.5

2012 Summer

編集長　松浦智之
編集　鎌田広子
青井大地
ライター　柏崎吉一
制作協力　USP 友の会
表紙デザイン　石塚幸治
（ジーズバンク）

発行人　當仲寛哲
発行元　（有）USP 研究所
定価 400 円（本体価格 381 円）

ご意見・ご感想・投稿はこちら
mag@usp-lab.com
（技術的なお問合せには対応しかねますのでご了承ください）

USP Magazine info

USPMagazine の細かい情報が定期的にアップアップデートされています。
「イイね」ボタンぽち！ ----->>

USPMagazine　検索

編集後記

・東京スカイツリーが開業しましたね。「粋」と「雅」を毎夜交互にライトアップしてますが、雅の日の江戸紫色の照明は、見てるとタワーの北側にしか当ててないようなのですよ。なので錦糸町側からは全然江戸紫に見えないのです。何故そうしているんでしょうねぇ？（編集長　松浦）
・３月末に開催された BSD カンファレンスでのお話。 vol.4 から始めた「Tech 数独」がカンファレンスに来ていた外人さんに人気でした。30 分ぐらいあーでもないこーでもない、って夢中に解いていたのを覚えています。言葉関係ないのだ！と、コーナー作った後に意外な効果を知りました。しかし、外人さんでも「日本語ができる」方が２名ほど、USP Magazine 買って行ってくれたのです。嬉しかったです。（鎌田）
・今号から編集に本格的に参加しました。今まで外側から眺めるだけだった UNIX/Linux/ シェルスクリプトの世界にどっぷり浸かっています。いろいろな文化の違いに驚きつつ、思想に共感し、そして志の高さに敬意を表し、どんどん深みへはまり始めています。（青井）
・伊豆・網代にある洋上の釣り堀に立ち寄った。網の中を泳ぐ鯛と小ぶりのアジ。所詮釣り堀と高を括っていたが、アジの動きが予想以上に素早い。こちらを嘲笑うように餌のオキアミだけ持っていかれる。鯛は釣れるのになぜ？忍耐力の尽きる頃にやっとかかり始めた。手を焼く方が燃えるが、コツをつかむまではしんどい。なんでもそうだ（ライター 柏崎）

次号予告

2012 年 9 月発行予定です。

- **Tukubai レシピ　Web 編**
- **TechLION 再録 砂原秀樹教授**
- **デール・ダハティ（O'Reilly Media 創始者）に訊く技術者哲学**
- **後藤大地　漢の UNIX**
- **ほか　好評連載陣**

内容は変更になる場合があります。

発行人　當仲寛哲　発行・発売 有限会社ユニバーサル・シェル・プログラミング研究所
〒105-0003 東京都港区西新橋 3-4-2 SS ビル 3 階　電話 03-3432-1174

定価 400 円（本体価格 381 円）

イベント予告

THURSDAY 26 JULY

TechLION vol.8
すなはら ひでき TALK LIVE

MC：法林浩之、馮富久
Talk Guests：藤川真一（えふしん）、山本裕介、高橋真弓

2012.7.26 Thursday　Start 19:30 ～ Close 22:00　Place：六本木 SuperDeluxe

Ticket：前売り・予約　2700 円、当日 3200 円（1ドリンク込）

主催：ユニバーサル・シェル・プログラミング研究所　協賛：技術評論社、オライリージャパン、サムライファクトリー
後援：日本 UNIX ユーザ会、EMZERO、キャリアデザインセンター「エンジニア Type」

TechLION vol.8　検索

UNIX/Linux/ シェルスクリプトの可能性を極限まで追求します！

USP友の会 会員募集中

シェルでつながる技術の輪！
なんでもありのグルーコミュニティ

約 350 名の会員が定例会や勉強会で活発に交流をしています。

詳しくは
http://www.usptomonokai.jp　にアクセス！

twitter　@usptomo　検索

USP 友の会　検索

USP MAGAZINE バックナンバー 好評発売中！

vol.0

vol.1

vol.2

vol.3

vol.4

USP MAGAZINE は、世界初のシェルスクリプト技術情報誌。でもご存知のとおり、シェルスクリプトはグルー言語。OS 深層から様々な言語・アプリの話まで、さらには技術の先にいるエンジニア達にもスポットを当てます。**目指すは、シェルスクリプトとエンジニア達の地位向上！**

自炊不要　定期購読又はバックナンバーをお申込みいただくと、PDF 版が無料で手に入ります。

ご購入、お申込みはこちらから⇒　USP 出版　検索