コードは思い立った瞬間に書け。こたつは風邪ひく前に出せ。

ISSN 2187-9664

USP MAGAZINE

2014 Winter 38th

for the sophisticated shell scripters

特集
突撃!隣のコードレビュー会

新連載
熊野憲辰
「未来に活きる!
現場で使える!
データモデリング」

すすたわり氏
に訊く、技術者哲学

好評連載
大岩元
「今私たちは
何を学ぶべきか」

UNIXネイティブの
電子工作塾
「はんだごてとキーボード」

From Editor

今号の特集で取り上げた USP のコードレビュー会やコマンド研究会では、様々なエンジニアの異なる立場や環境、考え方がぶつかり合います。その中で、お互いに他人の方法や思想を自分のものとして受け入れ、新たなものを作り出していくのです。

マイク・ガンカーズの著書『UNIX という考え方』に、「ソフトウェアの梃子を有効に活用せよ（Use software leverage to your advantage.）」という定理が示されています。

ソフトウェアの梃子（leverage）とは、他人が書いたプログラムを自分のプログラムの中で利用すること。どんなに優秀なプログラマでも、1 人で書けるコードには限界があるが、すでにあるプログラムをうまく利用すれば、もっと多くのプログラムを書けるということです。

自分が作ったものとは異なる「他人が作ったもの」を受け入れる、このことに抵抗する人は多いでしょう。ガンカーズは「独自技術症候群」と呼んでいます。技術に限らず、自分とは異質な「新しい」ものを自分の中に受け入れることは難しいことです。

しかし、多くの人がソフトウェアの梃子を活用して作り上げた「UNIX」が、素晴らしい成果を収めていることを思い出せば、あなたにもきっと、新しいものを受け入れる " 勇気 " が出てくるでしょう。

USP MAGAZINE 編集部

Contents

※「うにっくすなやつら」、「IT 美女図鑑」はお休みします。

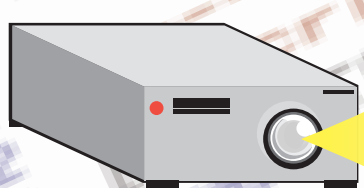
※ 薄字の記事はよりぬき版には収録されていません。是非正式版をお求めください。

特集

突撃！隣のコードレビュー会

あなたのコード、見せてください。

USP MAGAZINE 編集部



こんにちは、はじめまして。私は、全国津々浦々、面白いプログラムコードがあると聞き付ければ見せてもらいに訪問する、突撃レポーターの「ソースコード・ヨメスケ」です。
さて今日は、シェルスクリプトのコードレビュー会を開催している集団がいるということで、そちらにおじゃました時の模様をレポートします。実はレビュー会の後、コマンド研究会という会議も開かれていまして、そちらもあわせて紹介します。

1. 朝のシェルスクリプトコードレビュー会

コードレビュー形式の勉強会。参加したことありませんか？　あれって面白いですよねー。十人十色なスタイルのソースコードにお目に掛かれますし、予想もしていなかった鋭いツッコミが交わされたりしてスリリングですよね。今回、シェルスクリプトのコードレビュー会を開催しているところがあるという情報を得たので突撃してきました。さて、皆さんどんなコード書いているんでしょうか！

コードレビューはおもしろい

コードレビューといえば、制作されたプログラムコードを見返すことで、見過ごされてしまった動作の誤りや脆弱性を発見したり、効率や性能・堅牢性を上げるため、あるいはメンテナンス性・可読性を向上させるための修正点の洗い出しなど、現在のコードの品質を向上させるための作業です。

しかし向上するのはコードの品質だけではありません。コードレビューに参加したプログラマーのスキルも向上していきます。ゆえに、IT 系の勉強会などでもコードレビュー会が企画されることがあります。

勉強会としてのコードレビュー会は、自分の書いたソースコードを互いに見せ合いながら「そのやり方いいね！」とか「いや、それはこー書くでしょ」などと屈託ない意見が交わされたりします。レビューをする時も、される時も、その様子を観覧する時も、あれが実に面白いのです。

まず、ソースコードのパッと見が十人十色で全然違います。いくら記述ルールを統一していても、細部まで見ると書く人の流儀が見えてきます。なので、同じことをやるにしても実現方法・アルゴリズムが様々。「そんなふうにやるのか！頭いいな！」と自分では成し得なかった発想に何度も何度も感心します。

逆に、ツッコミを入れたくなることもしばしば。「その書き方はわかりづらいよね」とか「あれ、そこ矛盾してない？」などなど。自分ではなかなか気づかなかった事でも、他人が見ればあっと言う間に気付いたりします。時には「指摘を受けることは承知の上で、別のメリットを得るために敢えて書いている」などということもあり、それはそれでとても奥が深いです。

どんな言語であっても、コードレビュー会で様々なコードや意見を聞くことはとても刺激的。そして自分の書くコードもどんどん進化していきます。レビュー会に参加し続けるうち、自分の書くコードががらっと変わっていたことに気づき、驚く人もいます。

コードレビュー会、職場や各地の勉強会など、開催されるとあらば是非参加してみてはいかがですか？

シェルスクリプトのコードレビュー会 !?

USP 研究所では毎週金曜、技術部隊が集まって、コードレビュー会が開催されているそうです。シェルプログラミングが社名にもなっているだけあって、扱うコードはシェルスクリプトがほとんど。ビジネスシステムで使用するシェルスクリプトの例を用意し、それらについて議論を交わしているそうです。

JavaScript や Ruby, Python などの他の LL 言語のレビュー会はよく耳にしますけど、シェルスクリプトのコードレビュー会とは珍しい！ということで、社内のレビュー会ではあるものの取材させてもらい、今回の特集記事にすることにしました。

▲シェルスクリプトのコードレビュー会とは珍しい

［1］データ検索用 CGI Web フロントエンド

お一人目は USP 技術研究員の大内さん。今回発表したプログラムは、商品在庫を問い合わせる Web アプリケーションプログラムの例だそうです。

ここではその中から、ユーザーの Web ブラウザーから検索等のリクエストを受け付けるフロントエンド部分のシェルスクリプト（**リスト 1**）を紹介します。（検索ルーチン本体や結果表示などは、別のシェルスクリプトに分け、一つのプログラムが長くならないようにしているそうです）

リスト 1. 返品データ入力検索：フロントエンド CGI

```
#!/bin/ush -xve
#
# システム名      :**********
# サブシステム名:*******
# 業務名          :****
# プログラム名    :本部／返品データ入力検索CGI
# 概要            :**********
# 詳細            :
# 備考(Usage)     :**********
# シェル名        :***_HENPIN_DATA_NYUURYOKU.CGI
# 作成日          :2013/6/28
# 会社名          :USP-LAB
# 作成者          :T.Ouchi

#////////////////////////////////////////////////////////////////////////////
# 初期設定
#////////////////////////////////////////////////////////////////////////////

#-------------------------------------------------------------
# 変数の定義
#-------------------------------------------------------------

# 走行ログの記録
HOME="/home/**"
logd="${HOME}/AP/LOG"                                          # ログディレクトリ
logf="${logd}/LOG.$(basename $0).$(date +%Y%m%d)_$(date +%H%M%S)_$$"  # ログファイル名
echo  "${logf}" > /dev/null 2>&1
log 2> ${logf}

# PATHの定義
PATH=/home/UTL:/home/TOOL:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:${PATH}
LANG=ja_JP.UTF-8

HOSTNAME=$(hostname -s)
tmp="/tmp/$$-$(basename $0)_$(date +%Y%m%d%H%M%S)"          # 一時ファイル
cgid=${HOME}/AP/MCS_DENPYOU/CGI                          # CGIディレクトリ
semd=${HOME}/SEMAPHORE

#---- 処理日時 ----
sdaytime=$(date +%Y%m%d%H%M%S)

#-------------------------------------------------------------
# エラー処理(システムエラー系)
#-------------------------------------------------------------
cgi_err="*********"                # 予期せぬエラー
#
err ERROR_EXIT(){
```

走行ログ[*2]をとる場合、bash では "exec 2>${log}" などと書くが、ush ではこのように書く。

環境変数を定義する場合でも ush では export を使わない。

ワークエリア(作業用ファイル)のプレフィックス定義

ush の場合、エラー処理関数は、関数名の前に "err" を記述する

1. 朝のシェルスクリプトコードレビュー会

```
49   # エラーのロギング
50   touch $semd/$(basename $0).$HOSTNAME.ERROR.$sdaytime
51
52   cat <<ETX | nkf -W -sxLw
53 Content-type: text/html; charset=Shift_JIS
54
55 "${cgi_err}","1","EOT"
56 ETX
57   # 本ハンドラに飛び込むとエラーステータスを返す
58   exit 1
59 }
60
61 #////////////////////////////////////////////////////////////////////////////
62 # 処理部
63 #////////////////////////////////////////////////////////////////////////////
64
65 #----------------------------------------------------------------------------
66 # 画面からのPOST情報をファイルに無変換保存
67 #----------------------------------------------------------------------------
68 dd bs=${CONTENT_LENGTH}          >        ${tmp}-ifdata
69
70 #----------------------------------------------------------------------------
71 # 入力データの切り出し
72 #----------------------------------------------------------------------------
73 ${cgid}/***_HENPIN_DATA_NYUURYOKU.INITGET  ${tmp}-ifdata          > ${tmp}-flddata
74
75 #-------------------------------------------------
76 # チェック処理
77 #-------------------------------------------------
78 ${cgid}/***_HENPIN_DATA_NYUURYOKU.CHECK ${tmp}-flddata > ${tmp}-check
79
80 if [ -s ${tmp}-check ] ; then
81   #-------------------------------------------------
82   # 表示用データ作成(エラー)
83   #-------------------------------------------------
84   ${cgid}/***_HENPIN_DATA_NYUURYOKU.DISPLAY ${tmp}-check
85
86 else
87
88   #-------------------------------------------------
89   # データ検索
90   #-------------------------------------------------
91   ${cgid}/***_HENPIN_DATA_NYUURYOKU.SEARCH ${tmp}-flddata ${sdaytime} $$         > ${tmp}-search
92
93   #----------------------------------------------------------------------------
94   # データ表示
95   #----------------------------------------------------------------------------
96   ${cgid}/***_HENPIN_DATA_NYUURYOKU.DISPLAY ${tmp}-search
97
98 fi
99
100 #-------------------------------------------------
101 # 正常時のロギング
102 #-------------------------------------------------
103 touch $semd/$(basename $0).$HOSTNAME.END.$sdaytime
104
105
106 #////////////////////////////////////////////////////////////////////////////
107 # 終了
108 #////////////////////////////////////////////////////////////////////////////
```

ブラウザーから送られた POST/MIME 形式の取得

取得した POST/MIME 形式を Tukubai の「name 形式」へ変換

取得したデータの各種正当性を検査する

検査不合格ならエラー画面表示

ブラウザーのリクエストに基づく実際の検索を行うシェルスクリプト（リスト 2）

検索結果を画面に表示

```
109 rm -f ${tmp}-* > /dev/null
110 exit 0
```

ワークエリア（作業用ファイル）の全削除

■ プログラムのポイント

このプログラムで注目してもらいたいポイントは、独自シェル /bin/ush[*1] に対応させるための記述上の注意点とのことです。

走行ログ[*2] の出力には exec ではなく独自内部コマンドの log を使う点、環境変数の設定のしかた、エラー処理関数の定義には予約語 err を付ける点等が異なります。

■ 解説

ush 自体は OSS として公開されているシェルではないため、「ush 使用時の注意点」に関しては多くの読者にとって関心の及ばないところではあります。しかし、このプログラムにはシェルプログラミングを実践するうえで参考になる記述がいくつか見られます。

一部は連載「ユニケージエンジニアの作法」でも紹介してきたことですが、

- 上から下へ素直に読めるプログラミング
- 冒頭には連絡先までわかるコメント
- POST リクエストの受け取り方
- 走行ログの取り方

などです。

*1 ush（ゆーしぇる）とは

FreeBSD の /bin/sh (ash) を元にし、そこから fork する形で USP 研究所が独自開発しているシェル。

ash は、Linux の多くのディストリビューションで標準とされる Bash に比べ、機能がシンプルなぶん高速に動く。そこで、そのメリットをさらに増すためユニケージプログラミング上で不要な機能を徹底的に削ぎ落とし、更なる高速化を図っている。

*2 走行ログとは

sh -xv などと記述して、シェルスクリプト実行ステップを全てログファイル出力したものの通称。

デバッグ時や障害発生時に素早く対応できるように、ユニケージプログラミングにおいては通常必ず行う。

■ 検索ルーチン

2 本目のリストは 1 本目に紹介したフロントエンドから呼び出される検索ルーチンのシェルスクリプトの一部です。検索用の CGI スクリプトということで、肝心の検索部分はどうなっているのか気になったため、見せてもらいました。（リスト 2）

リスト 2. 返品データ入力検索：検索（抜粋）

```
129                         :
130 cat ${lv4d}/${tenpocd}/HENPIN_DATA |
131 #  1:店舗コード          2:店舗名          3:データ作成日付       4:データ作成時刻 5:SEQ
132 #  6:事業会社コード      7:事業会社名      8:商品コード           9:商品名         10:数量
133 #  11:返品先            12:返品先名       13:原単価              14:売単価        15:伝票データ取込区分
134 #  16:取込チェック結果 17:エラーフラグ 18:伝票作成ステータス 19:本部店舗区分  20:担当者
135 #  21:税区分            22:更新フラグ     23:更新日               24:ユーザID      25:入力日時
136
137 # 更新フラグが8:削除のものは排除
138 gawk '$22!="8"{print $0}'           |
139 #-------------------------------------------------------------------------------
140 # 検索処理
141 #-------------------------------------------------------------------------------
142 # 店舗コードによる絞込み
143 if [ ${tenpocd} != "_" ] ; then
144   gawk '$1=="'${tenpocd}'"{print $0}'
145 else
146   cat -
147 fi |
148
149 # 担当者に条件があれば検索
150 if [ ${tantouid} != "_" ] ; then
```

if 文を活用して、店舗コードの絞り込み指定があった場合のみ、gawk で絞り込む。（そうでない場合は cat）

if 文を丸ごとパイプで繋ぐことで、gawk が選択された場合も cat が選択された場合も次のコマンドへパイプされる。

1. 朝のシェルスクリプトコードレビュー会

```
151   gawk '$20=="'${tantouid}'"{print $0}'
152 else
153   cat -
154 fi |
155
156 # 取込チェック結果に条件があれば検索
157 if [ ${torikomi_kubun} != "_" ] ; then
158   gawk '$15=="'${torikomi_kubun}'"{print $0}'
159 else
160   cat -
161 fi |
162
163 # エラーフラグ(表示フラグ)に条件があれば検索
164 if [ ${hyouji_data} != "_" ] ; then
165   gawk '$17=="'${hyouji_data}'"{print $0}'
166 else
167   cat -
168 fi  > ${tmp}-search_result
169
170 # 該当データが無い場合、検索結果0件として終了
171 [ ! -s ${tmp}-search_result ] && ZERO_HIT_EXIT
172                       :
```

全部載せきれないため、一部を抜粋しました。興味深かったのは if 文を丸ごとパイプで繋いでいる部分です。絞り込みに使いたい項目が 4 つありますが、それら全てが毎回必要なわけではありません。必要な場合だけ AWK で絞り込み、そうでない場合は cat でそのまま次へ送る、というコードを、if 文を丸ごとパイプで挿むことで実現しています。これはウマい！

こういう書き方をするという発想がなければ、きっと AWK のパターン部分に条件をまとめて記述することでしょう。けれども、見た目がゴチャゴチャするし、絞り込む必要が無いと始めからわかっている場合は、cat を経由させる方が明らかに速いのです。

これは、真似したい小技だと思いました。こういう自分に無い発想に出会えるところもまた、コードレビュー会のいいところです。

［2］tag*コマンド(*3)を活用した伝票処理プログラム

二人目は、若手の技術研究員見習い (?) の村本さんによる発表です。

村本さんは、ユニケージの仕事に携わってまだ 2 か月程だそうで、そんな村本さんにはユニケージ開発手法のトレーニングを兼ね、新しく開発された tag* コマンドの評価をするための伝票処理プログラム制作を任されたとのこと。今日のレビューにて、それが適切なコードであるかどうか、様々な意見が出てきました。

■ 概要

このプログラムは伝票を処理するためのもので、COBOL が得意とする処理がユニケージでできる例だそうです。これをビジネス版 Tukubai で用意されているコマンド、とりわけ tag* コマンドを用いてシェルスクリプトに移植し、性能や使い勝手を評価しようというものだったそうです。

村本さんにとってはトレーニング課題になっていたものの、ユニケージ開発手法に慣れているエンジニア達にとっても、tag* コマンドは比較的新しい要素と

3 tag コマンドとは

AWK や sort、それに従来の各種 Tukubai コマンドでは、加工対象フィールド指定にフィールド番号を用いていた。しかしこの場合、フィールドの並びが変わると番号の付け直しが生じるという不便がある。

そこで SQL 同様、名称での指定ができるよう、AWK 等の UNIX 標準コマンドを含め、従来のコマンドをラッピングしたコマンド群のこと。

※ tag* コマンドは、前号特集でも紹介した AWS Tukubai によって気軽に体験することができる。

いえます。

AWK や sort、それに従来の各種 Tukubai コマンドでは、加工対象フィールド指定にフィールド番号を用います。しかしこの場合、フィールドの並びが変わると番号の付け直しが生じるという不便があるのです。

そこで SQL 同様、名称での指定ができるよう、AWK 等の UNIX 標準コマンドを含め、従来のコマンドをラッピングしたコマンド群が tag* コマンドです。（tag* コマンドは、前号特集で紹介した AWS Tukubai によって気軽に体験することができます）

このコードは、この tag* コマンドを本格的に活用してみたコードになっているというわけです。

初心者ゆえのツッコミ以外にも、tag シリーズコマンドの使用上の注意に関する議論が起こりました。

リスト 3. tag* コマンドを活用した、COBOL 処理ライクな伝票処理シェルスクリプト（一部抜粋）

```
43                          :
44 # 2-2 当日売上データファイルの伝票区分に上記が存在する場合
45 # 2-2-1当日売上データファイル.売掛先コードが売掛先マスターに存在しない場合(3を参照)
46 #    入力データをそのまま当日売上エラーデータに出力する。
47 tagmsort key=売掛先コード $tmp-ok                                    |
48 tagself 売掛先コード 現法コード 作業日/予備2                          |
49 tagcpy 売掛先コード+現法コード TAGKEY                               > $tmp-meisai
50 tagself 売掛先コード 現法コード IOCS 予備1/予備2 $datd/URIKAKE-MASTER |
51 tagcpy 売掛先コード+現法コード TAGKEY                               |
52 tagmsort key=売掛先コード                                           |
53 tagjoin0 +ng key=売掛先コード - $tmp-meisai                         > $tmp-uri-toujitu 2> $outd/TOUJIT
   U-URI-ERROR
54
55 # 2-2-2 当日売上データファイル.売掛先コードが売掛先マスター存在する場合(3を参照)
56 #    2-2-2-1 当日検収明細データの明細レコードを編集して出力する。
57 #    2-2-2-2 当日検収明細データの現法コードが("CHN","KOR","TIW")以外の場合
58 tagawk ´NR==1; NR>1&&%{現法コード}!~/^(CHN|KOR|TIW)$/{print}´ $tmp-uri-toujitu      > $tmp-noasia
59 tagmsort key=伝票区分 $datd/301VOUCHER-DIV-1                                       > $tmp-301VOUCHER-DI
   V-1
60 tagmsort key=伝票区分 $tmp-noasia                                                  |
61 tagjoin1 +ng key=伝票区分 $tmp-301VOUCHER-DIV-1 -                                  > $tmp-mok 2> $tmp-m
   ng
62
63 # 伝票区分(VOUCHER-DIV)が("118","119","128","129",
64 #                         "130","135","140","145",
65 #                         "150","155","157","198","199")の場合
66 tagawk ´NR==1;                                                                    ¥
67         NR>1{                                                                    ¥
68           if(%{消費税記載区分}==2){                                               ¥
69               %{税額}=%{税額}*-1;                                                 ¥
70               %{商品コード}="TAX";                                                ¥
71               %{税抜き金額}=0;                                                    ¥
72           print}}´ $tmp-mok                                                        > $tmp-OK
73 #tail -n +2                                                                      > $tmp-OK
74 # 伝票区分(VOUCHER-DIV)が("110","113","114","120",
75 #                         "125","138","149","148",
76 #                         "149","158","159","190","193","194")の場合
77 tagawk ´NR==1;                                                                    ¥
78         NR>1{                                                                    ¥
79           if(%{消費税記載区分}=="2"){                                             ¥
80               %{税額}=%{税額}*+1;                                                 ¥
81               %{商品コード}="TAX";                                                ¥
82               %{税抜き金額}=0;                                                    ¥
83           print}}´ $tmp-mng                                                        |
84 cat - $tmp-OK                                                                    |
85 tagself TAGKEY 売掛先コード/予備2                                               |
86 tagdelf TAGKEY                                                                  |
```

1. 朝のシェルスクリプトコードレビュー会

```
87 tagself 現法コード 売掛先コード 作業日/予備2                              |
88 tagmsort key=現法コード@売掛先コード                                      > $tmp-TOUJITU-KENSHU-MEISAI
89
90 # 当日検収明細の編集処理
91 tagchange 税抜き売単価 売単価 $tmp-TOUJITU-KENSHU-MEISAI |
92 tagchange 売単価通貨コード 通貨コード                      |
93 tagchange 明細税込み金額 税込み金額                       |
94 tagchange "注文番号(親)(明細)(20+54)" ´注文番号(親)´   > $outd/TOUJITU-KENSHU-MEISAI
95                        :
```

■ 質疑

レビューア：このプログラムは全部で何行ですか？
村本さん：コメント・空行を除いて、実質 174 行です。
レビューア：作るのにどれくらいかかりましたか？
村本さん：２週間ちょっとですね。まともに書いたのはこれが初めてだったので。
レビューア：だいぶ日数かかりましたね。でもその後はどうですか？
村本さん：この次に書いたプログラムは同じくらいでしたが 3 日間くらいで書けました。取り扱うデータが少ないというのもありましたが、でも「こういう時はこう書けばいいのかなぁ」って雰囲気が掴めてきたんじゃないかなぁと。
レビューア：ところで、これまでプログラムの経験ってありましたか。
村本さん：いえ、全く初めてです。

◇　◇　◇

いくつかツッコミは受けていましたが「全く初めて」でもここまで書けちゃうものなんですね。自分が全く初めての時に書いたプログラムは一体どんな出来栄えだったかな？　と想像すると結構へんてこなコードを書いていたような気がします。ここまで書けるのもユニケージが説く作法によるのでしょうか……。

［番外］シェルスクリプト製ショッピングカート

三人目は……なぜかレポーターの私（の中の人＝松浦）のコードレビュー。取材する側のはずなのですが、「じゃあ、今度はヨメスケさんお願いします」ということで、逆にこちらが突撃レビューされることに。

確かに私個人もシェルスクリプト好きで、前号の編集後記では「シェルスクリプト製ショッピングカートを作るぞ！」などと、豪語しとりました。これまでの取材でユニケージの作法は結構見聞きしていたものの、実際のユニケージエンジニア達からすると一体どれだけのツッコミどころを秘めているのでしょうか。ご覧ください。（・_・;）

■ 概要

リスト 4 は、ショッピングカートには付きものの「カゴに入れる」ボタン押下時に走る Ajax 処理を抜粋したものです。Cookie で買い物カゴ (セッション) ID を管理しており、カゴ入れボタンが押されたらサーバーで保持しているセッションファイルを更新し、応答を返します。複数サイト間でカゴを共有できるようにと、Cross-Origin Resource Sharing という規約にも対応させるなど、凝った作りにしてるのですが……。

■ 案の定、多数のツッコミが

まず「ユニケージの作法からすると、全体的にシェル変数を使い過ぎ」ということです。理由は、変数の代入をあちこちで行うと「この時点での値は何だっけ？」と上の行のコードを読み返させてしまうために、目線の移動を招くことになり、結果として読み難いコードになるからだそうです。

確かに、前のお二人のコードを見ると、冒頭で値を定義して、あとはほとんど参照するだけですね。それに、シェル変数の多用は良くないと説く記事「ユニケージエンジニアの作法 その四」を、過去に自ら編集しておりました……。

また、外部データである環境変数をコマンドの引数として利用する記述 (285 行) も、セキュリティーホールの元になりやすいのでやらないとのこと。こういう時は、内部結合用のコマンド join0 を使い、マスターファイルに記述してあるいずれかの行と一致するかを見て判定するんだそうです。……なるほど。

リスト 4.「カゴに入れる」ボタンが押された時に呼び出される Ajax（一部抜粋）

```
239                        :
240 # === (これより breakdown 区間) ======================================
241 failure=0
242 while [ 1 ]; do
243
244 # --- 取引対象商品在庫数は足りているかどうか確認 ----------------------
245 GET_STOCKQTYS.SH $reqpid > $Tmp-currentstock
246 [ $? -eq 0 ] || errorcode_exit 'Cannot_check_stock_quantity'
247 curqty=$(cat $Tmp-currentstock | self 2)
248 if [ ¥( $curqty !='*' ¥) -a ¥( $curqty -lt $reqqty ¥) ]; then
249   failure=1
250   break
251 fi
252
253 # --- セッションファイルの中身を更新する ------------------------------
254 zgrep -v "^$reqpid[[:blank:]]" "$File_session" >  $Tmp-newsessionfile
255 echo "$reqpid $reqqty"                          |
256 awk '$2>0'                                     >> $Tmp-newsessionfile
257 cat $Tmp-newsessionfile |
258 gzip                    > "$File_session"
259 [ $? -eq 0 ] || errorcode_exit 'Failed_to_update_the_session_file'
260
261 break; done
262 # === (breakdown 区間ここまで)======================================
263
264 # --- Cookieの寿命分だけ延長した日時を得る(dummy sessionでない場合) --
265 cookiestr=''
266 if [ "_$visitorid" != '_' ]; then
267   newexpire=$(TZ=UTC/0 date +%Y%m%d%H%M%S                |
268               calclock 1 -                               |
269               self 2                                     |
270               awk '{print $1+60*'$COOKIE_LIFELIMIT_MIN'}' |
271               calclock -r 1 -                            |
272               self 2                                    )
273   case "${HTTPS:-off}" in [Oo][Nn]) secure='; secure';; *) secure='';; esac
274   cookiestr=$(printf '¥nSet-Cookie: visitorid=%s; expires=%s; path=/%s' "$visitorid" "$(cookiedate $new
    expire)" "$secure")
275 fi
276
277 # --- Cross-Origin Resource Sharing 対応 -------------------------------
278 # 環境変数HTTP_ORIGINと等しい文字列の行が ALLOWED_ORIGIN_LIST.TXT の中にあったら
279 # CORSに対応した2つのHTTPヘッダーを生成する
280 cors=''
281 cat $Homedir/CONFIG/ALLOWED_ORIGIN_LIST.TXT |
282 env - sed 's/^#.*$//'                       | # コメント除去1
283 env - sed 's/[[:blank:]]¥{1,¥}#.*$//'       | # コメント除去2
284 grep -v '^[[:blank:]]*$'                    | # 空行除去
285 awk '$1=="'"$(echo "_${HTTP_ORIGIN:-.}" | sed '1s/^_//' | tr -d '"')"'"{ret=1} END{exit 1-ret}'
286 if [ $? -eq 0 ]; then
287   cors=$(printf '¥nAccess-Control-Allow-Origin: %s¥nAccess-Control-Allow-Credentials: true' "$HTTP_ORIG
    IN")
288 fi
289
290 # --- HTTPヘッダー送信 -------------------------------
291 cat <<-HTML_HEADER
292       Content-Type: text/plain$cookiestr$cors
293       Cache-Control: private, no-store, no-cache, must-revalidate
294       Pragma: no-cache
295
296 HTML_HEADER
297                        :
```

変数に値を書き込んでいる箇所が全体的に多い。（今の値は何だっけ？と、読む人の目線移動を招き、可読性を下げる要因になる）

自作のシェルスクリプトを呼びだす時は、必ずフルパスで指定すること。環境変数 PATH を不正に書き変えられていたら、何が呼び出されるかわからない。

HTTP のセッション管理を行うための Cookie。ここでは expire を 24 時間後にセットするためのタイムスタンプを生成している。

環境変数をコマンド引数にすることは、一歩間違うとセキュリティーホールに繋がるのでやるべきではない。この行のように AWK で特定のレコードを抽出したいのなら、代わりに Tukubai の join0 コマンドを使うべき

未来に活きる! 現場で使える!

データモデリング

第1回　データモデリングとモデリング

熊野憲辰

企業情報システムに欠くことのできないデータベース。各種 NoSQL が注目されるが、今なお多くの企業システム、特に重要な基幹業務については RDB および RDBMS が支えているのが実情だ。しかし、長年改修を重ねてきたデータベースを基盤とするシステムが経営環境の目まぐるしい変化に追従できなくなっている。その再構築においてまず鍵を握るのは、ビジネスや業務を可視化する「モデリング」である。

熊野と申します。今回から何回かにわたり、「データモデリング」をテーマに記事を連載します。

初回ですので、簡単に自己紹介をさせていただきます。1989 年に製薬会社の情報システム部門に配属され、そこで自社情報システムの設計〜内製開発を担当してきました。2013 年 9 月末に退職し、エンジニアとして独立、現在に至ります。

また、ユーザー企業の IT 部門を経営課題の解決に向けた活動に直結させることを目的とした「特定非営利活動法人システムイニシアティブ協会」で運営に携わっています。長年本業に IT を利用してきたバックグラウンド、またユーザー主体のシステム開発を推進してきた経歴がありますので、この連載は、特にユーザー企業の IT 部門の方々に是非ご覧頂きたいと考えています。データモデルを武器として使いこなせれば、自社のビジネスをより深く理解でき、強くすることにつながります。

本連載の第 1 回では、私がユーザー企業の情報システムの設計〜開発を行ってきた中で感じた様々な問題の提示から、始めたいと思います。

エンタープライズシステムで噴出する様々な問題

昨今、エンタープライズ系のシステムよりも、コンシューマ系のアプリケーションやクラウドサービスのほうが必要とされる技術の水準も高く、開発者にとっては花形の仕事だ。それに比べると、エンタープライズ系のシステムは、地味でありすでに語りつくされた感もある。

しかし、それは表層的な見方に過ぎない。エンタープライズシステムは今日もなお、企業ビジネスに価値を提供するシステムであり続けている。歴史ある会社のシステムは機能拡張を重ねて大規模化、複雑化しているが、そのシステムを開発し、安定的に運用することは、極めて高度な知的作業である。企業活動は生き物であり、様々な変化と制約を内包している。これを支えるのが企業における情報システムの本来の役割である。だがそれを阻害する多くの問題を抱えていることも事実だ。

以下、エンタープライズシステムにおける主たる問題を 2 点示す。

1. 情報システムの老朽化

いまなお 20 〜 30 年前にシステムを作ったシステムを使っている企業は少なくない。大企業においてとりわけ顕著である。逆に、中小企業のほうがコンシューマ系のサービスアプリなどを軽快に使いこなし、企業を取り巻く環境変化や技術の進展に対して俊敏だ。

20 〜 30 年前、大企業は大型コンピュータを保持し、専用回線を敷き、それまで人手が主体だった定常的な業務の電算化や省力化を力強く推進した。しかし、業務の効率化が一巡してしまうと古いシステムが現在では足かせになり、それを有する大企業ほど目まぐるしい経営環境に追随できず、鈍重に映る。

そこで、情報システムの再構築だ、となるのだが一筋縄ではいかない。理由はいくつもある。システム開発時に書かれたはずの仕様書が見当たらない、業務を理解している IT 部門人員が不足している、縦割り組織の弊害などなど。なかでも仕様書はまったくアテにならないといえるだろう。マクロに俯瞰する仕様記述はほとんど存在せず、ミクロなプログラムレベルの仕様記述はたいてい間違いだらけだ。

2. 意図を汲めないテーブル設計

前述のように、システムの仕様書はどこの企業もかなり脆弱だ。しかし、

データベースにおけるテーブルの定義書というものは存在する。これが無いとデータベースを作る上で話にならない。

テーブルの定義書は、一般的にデータベースのテーブル一つ一つの項目レイアウトとして記述されている。

しかし、エンタープライズシステムの再構築の場面において、このテーブルの項目レイアウトを、ためつすがめつ眺めてみても、データが持つビジネス上の「意味」がなかなか見えてこないのだ。別の言い方をすると、テーブル設計における意図や全体の構造が読み取れない、項目間の帰属性もあいまいだ。そして、設計者の真意を探るために項目名、項目の値、実際のデータを基に、“推理”による現状分析が始まることになる。

ところが、テーブル項目の数は膨大であり、なおかつ各項目間の関係・制約にも気を配る必要がある。筆者の経験上、現在使われてない項目が全項目の中に 10 ～ 20%ほど混在しているものだが、これがまた厄介だ。

テーブル定義が単なるフラットな項目一覧になっている事態も頻発している。とりわけ古いシステムになるほど一つのテーブルが有する項目数が多く、推理の邪魔をする。こうした多くの項目を持つテーブルには、実際にはいくつものサブタイプ構造が見つかるものだ。サブタイプ構造とは、ある区分となる項目で大きくレコードが分割される構造を指す。

また、オプション項目も多くのデータベースで見受けられる。これは必須入力の項目でなく、必要に応じて記録される属性のことである。

これらの項目や属性を示すデータ群を立体的に表現し、ビジネスまたは業務を可視化するモデル化の作業に欠かせない手段の 1 つがデータモデリング、ということになる。データモデルを的確に記述することで、ビジネスを可視化することができる。

事実、多くの企業には、データモデルとしての物理的な ER（Entity Relation）図は（それなりに）存在する。しかし、上述のようにフラットにたくさんの項目を横に並べただけのテーブル定義書や物理モデルを眺めただけでは、システムに込められた真意を読み取ることは難しい。

モデルとは？

そこで、あらためて「モデル」が重要になる。モデルとデータモデルは厳密には異なる。次回以降それを詳述するが、一言でいうとモデルはビジネスや業務を記述するために描かれる記述文法の総称である。モデル化の手段はいくつもある。UML でもポンチ絵でもモデルはモデルである。データモデルもモデルの 1 つだ。ただ、RDB の設計など少し実装寄りになる。思い返すと、私は以前勤めていた会社でシステム開発時に要件定義書、基本設計書などを書いたことがない。これらはすべてモデルに帰着するのである。モデルはいわば通常のシステム開発で使われる「仕様書」そのものである。だから、要件定義書、基本設計書という言葉がそもそも出てこない。

人間は言語なくして思考することはできない生き物だ。また、モデルは言語の一種である。ゆえに、人がモデルを記述することは、思考することと等価であるといえる。モデルが思考のための箱庭装置であるといえば、この意図が伝わるであろうか？

モデルを描くための記述文法はそれほど難しいものではない。実は、それよりもデータモデルと表裏一体となっているビジネスの構造や、業務と役職の関係を精確に紐づけるなどするほうが、はるかに難しい。

だからこそ、モデルは、社外の人間ではなく、ビジネスを知っているべき企業の IT 部門の人員が描かなくてはならないと思っている。

次回以降、エンタープライズシステムのモデリングについて、詳細に解説をしていきます。よろしくお願いします。

TEX の強みはプログラムにあり

TUG2013レポート

USP MAGAZINE 編集部

皆さんは TEX[*1] と呼ばれるソフトウェアをご存知でしょうか。テキストエディターでマクロを記述すれば、それだけで組版[*2]作業まで済んだ文書を生成してくれるソフトウェアのことです。HTML がこれに近い存在と言えるでしょう。違いは HTML が Web 文書作成を得意としているのに対し、TEX は数式を扱う論文作成に特に向いているという点にあります。

日本初開催の TEX 国際ミーティング

去る 10 月 23 日～26 日の 4 日間、東京大学駒場キャンパスにて、第 34 回 TEX Users Group 年次大会 (TEX ユーザーによる国際会議) が開催されました。年に一回、これまで 34 回も開催されてきましたが、実は日本ではこれが初めての開催だったのです。

TEX といえば論文作成で使ったという方が多いのではないでしょうか。ちなみに USP 教育講座のテキスト作成もこのソフトウェアのお世話になっているそうです。しかし TEX の強み、あえて TEX を選択する意義はどのあたりにあるのでしょうか。これを知るべく TEX の最新技術・活用事例の数々が披露されるこのミーティングに参加してきました。

1 発表には驚かされるばかり

開催初日から、大変面白い発表が目白押し。観ていて特に強く感じたことは、TEX とプログラムとの親和性でした。例えば、TEX のマクロを記述するにあたって Lisp の制御構文を使えるようにする LISP on TEX という拡張機能の紹介や、TEX マクロ中に Python コードを埋め込むとその実行結果がドキュメントになるという PythonTEX[*3] の紹介などです。

活用事例に関する発表でも、やはりプログラムとの連携をうまく活用しているものがありました。その一つ、学習塾の事例では、塾生達に解かせる英国数理社の問題を作るために TEX を利用しているのはもちろん、計算ドリルなどの問題と解答をプログラムと連携して自動生成していると紹介されました。

そんな数々の発表の中で、特に凄いと感じた発表を 2 つ紹介します。

1.1 How I use LATEX to make a product catalogue that doesn't look like a dissertation

「見た目が論文臭くない製品カタログの作り方」と題したこの発表 (発表者 Jason Lewis 氏) には驚かされました。発表が始まると観覧者にサプリメントのカタログ冊子が回覧されてきたのですが (写真 1)、何とこれは LATEX で作ったというのです。「TEX でここまでできるのか！」と、ただただ驚きました。

写真 1 LATEX で制作されたカタログ冊子

一体どうやって作ったのか気になるところですが、コツは次の 2 つだそうです。

- フォントはサンセリフ系 (和文でいうとゴシック) を選ぶとそれっぽくなる。
- 色つきの長方形をデザインに積極的に活用すると、やはりそれっぽくなる。

*1 テフまたはテックと読みます。
*2 本文や見出し、図版のレイアウトを行うこと。
*3 最近の TEX インストーラーパッケージには標準で収録されているようです。

確かに配布されたカタログは、この 2 つのコツを守っているだけなのに、かなり見栄えが良いものになっています。

しかしカタログを作りたいだけならべつに TEX を選ばなくてもいいような気がしますが、Lewis 氏は TEX を選んだ理由をこう述べました。

> 商品カタログを生成するのに、DB から自動で商品データを引っ張ってきて簡単にカタログを作るという技が使えるのは、プログラムやスクリプトとの相性が良い TEX ならではだ。

ここでも TEX とプログラムの親和性が活かされていました。

1.2 Online Publishing via pdf2htmlEX

「pdf2htmlEX によるオンライン文書作成」と題した発表 (発表者 Lu Wang 氏) にも驚きました。pdf2htmlEX[*4]とは、名前のとおり PDF 文書を HTML に変換するコンバーターです。しかしその再現性が半端じゃないのです!! そのおかげで、TEX から生成した美しい数式入りの文書も、その美しさを保ったまま HTML 化できるというのがウリです。

Wang 氏は、「こういう雑誌のドキュメントがあります。これを html2EX で変換するとどうなるでしょう」と、スライドでサンプル文書を見せていました (写真 2)。中味は全くもって普通の商業誌で、「これはいくらなんでも無理でしょ。どこまで再現できるのか」と半信半疑でした。ところが Wang 氏が続けてこう言ったのです。「実は今ご覧いただいているのは pdf2htmlEX で変換した Web ページなんです」と。そのあまりの再現性に度胆を抜かれました。

写真 2　HTML 化しようとしている PDF、実は……

HTML5 を駆使するとここまでできてしまうんだそうです。中身のソースコードが一体どうなっているのか大変興味ありますが、なんとこのソフト、オープンソースで公開されていて誰でも試せるとのこと。スゴいです。是非皆さんにも試してもらいたいです。

2　今も愛され続けている TEX

TUG 2013 は大変面白いイベントでした。発表内容は、今回紹介したようなプログラムとの連携事例や、文書作成業務における活用事例、開発上の苦労話[*5]、そして最新の LATEX3 の動向など、盛りだくさんでした。いずれの発表者も皆楽しそうに見えました。

HTML5 の登場もあり、最近はドキュメントとプログラムの統合が促進されていますが、もともと組版という軸で高度に発達してきた TEX は、その分野では先行しているのではないでしょうか。互いの優れた面の影響を受けながら進化を続け、文書作成環境がより便利で効率的で楽しくなっていく未来が待ち遠しいです。

イベント会場の様子

TUG 2013 ポスター

実行委員のお一人、奥村晴彦先生

会場内の様子（藤田眞作先生による “XƎMTEX”）

[*4] http://coolwanglu.github.io/pdf2htmlEX/
[*5] 縦書き言語や右から左へ書かれる言語等、多国語対応の苦労話など

シェルスクリプト大喜利 第十一回

司会：『もっと吹く』編集長・みかん

皆様こんにちは。今月もシェルスクリプト大喜利（略して sh 大喜利）のコーナーがやってまいりました。

年の瀬ですなぁ。大晦日には年越し蕎麦ってのが定番ですが、由来の一つに「他の麺類より切れやすい蕎麦で一年の厄災を断ち切る」というのがあるんですって。そりゃめでたいのかもしれませんけど、このコーナーの司会やってるアタクシは、「切る」だの「断ち切る」だの言われると、どーしてもパイプ (|) を連想しちゃいまして、しかも「切れたらそれバグでしょ」なんて考えちまいます。重症ですね。

さて、そんな危険な (?) シェルスクリプト大喜利。まずは本コーナーのご説明からいきましょう。

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、芸術的、記述がシンプル、高速、など

ハ、アイデア・こだわりが光る

ニ、ネタになるバカバカしさ、くだらなさがある

など。でも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、基本的に 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。（JM にあるので、C シェル系での解答も OK ！　あと ksh、zsh も OK にしました）

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、Perl とかなきにしもあらず ??

五、Open usp Tukubai(http://uec.usp-lab.com/) も使用 OK ！　但し、使う意義がそれなりに感じられないと採用はキビシいですよ〜。

ルールもおさらいしたところで、さあ始めましょう！

本番開始

<一問目>

1 〜 n（n は引数で指定）の自然数の中に存在する『ズッカーマン数』を全て求めるシェルスクリプトを書いてください。

ズッカーマン数とは、元の値 n の各桁の数字を掛けあわせた数が、元の数 n の約数になっている数 n のこと（例 315 は 3 × 1 × 5 の 15 で割り切れる）。さて、早速投稿見ていきましょう。

◎むっつーさんの解答

```
1 #!/bin/sh
2 seq 1 $1 |grep -v 0 |xargs -i -P4 bash -c ' [ $(
  echo {} |sed -e "s/¥(.¥)/¥1*/g" -e "s/¥*$/)/" -e
  "s/^/{} % ¥(/" |bc ) = 0 ] && echo {} '
```

おぉ、これはアタクシの好きな蕎麦……、じゃなくてパイプでつながれた一行野郎。添付のコメントによれば、あれこれ考えた挙句に結局このシンプルな形になったようです。因みに xargs の -i オプションが使えない環境では代わりに "-I {}" と書けば動きますよ。

初投稿ありがとう！　なるほどね、各桁の掛け算やるのに sed で 1 文字ずつ分解しながら "*" を付け足してるのか。それを bc コマンドに食わせて掛け算させて剰余を確認するという。ウマイね！　**初段授与**だ。

◎お酒屋さんの解答

```
1 #!/bin/bash
2
```

```
seq 1 $1                                    |
awk '{for(i=1;i<=length($1);i++){
          printf substr($1,i,1)" "}
     }{print $1}'                           |
awk '{seki=$1;
      for(ii=2;ii<NF;ii++){seki=seki*$ii}
     }seki!=0{print $NF,seki}'              |
awk '$1%$2==0{print $1}'
```

さて次は、2回目のご参加、お酒屋さん。こちらは、AWK、AWK、AWK、と綺麗に繋がっていますね。えぇ、添付のコメントによりますと、1つ目のAWKで元数を各桁に分解、2つ目のAWKで分解後の各桁の積を求め、3つめのAWKでズッカーマン数の条件に当てはまるか判定しているそうです。

さすが、教育講座修了生。分かりやすい単位に処理を分解し、綺麗に解いてきたねぇ。しかも速い！　よし、**二段褒与**。これで五段だ。

◎同姓同名さんの解答（編注：誌面の都合で適宜改行挿入しました）

```
#! /bin/bash
N=$1
rm tmp_.*
yes ¥
    | head -n${N} ¥
    | grep -n y ¥
    | cut -d: -f1 ¥
    | grep -v '0' ¥
    | tee >(grep '[1379]2$' ¥
            | grep -v -e '[48]12$' ¥
                      -e '[48]92$' ¥
            >| tmp_2.txt)¥
    | tee >(grep '[2468]4$' ¥
            | grep -v -e '[1379]24$' ¥
                      -e '[2468]44$' ¥
                      -e '[1379]64$' ¥
                      -e '[2468]84$' ¥
            >| tmp_4.txt) ¥
    | tee >(grep '[1379]6$' ¥
            | grep -v -e '[48]36$' ¥
                      -e '[48]76$' ¥
            >| tmp_6.txt) ¥
    | tee >(grep '[2468]8$' ¥
            | grep -v -e '[2468]28$' ¥
                      -e '[1379]48$' ¥
                      -e '[2468]68$' ¥
                      -e '[1379]88$' ¥
            >| tmp_8.txt) ¥
    | tee >(grep -v '[2468]' ¥
      | tee >(grep -v '5' >| tmp_odd_not5.txt) ¥
      | tee >(grep '5$' ¥
        | tee >(grep    '75$' ¥
                >| tmp_odd_75.txt   ) ¥
        | tee >(grep -v '75$' ¥
                | grep -v '5[13579]*5$' ¥
                >| tmp_odd_not75.txt) ¥
        > /dev/null ¥
      ) ¥
      > /dev/null ¥
    ) ¥
    > /dev/null
ping -c 2 -i .2 localhost > /dev/null
```

```
sort -nm <({ echo 2; echo 4; echo 6; echo 8; })
  tmp_2.txt tmp_4.txt tmp_6.txt tmp_8.txt tmp_odd_
  not5.txt tmp_odd_75.txt tmp_odd_not75.txt >| tmp
  _.txt

paste -d' ' ¥
    tmp_.txt ¥
    <(paste -d' ' ¥
      tmp_.txt ¥
      <(cat tmp_.txt ¥
        | sed 's/1/ 1/g' ¥
        | sed 's/2/ 2/g' ¥
        | sed 's/3/ 3/g' ¥
        | sed 's/4/ 4/g' ¥
        | sed 's/5/ 5/g' ¥
        | sed 's/6/ 6/g' ¥
        | sed 's/7/ 7/g' ¥
        | sed 's/8/ 8/g' ¥
        | sed 's/9/ 9/g' ¥
        | mawk '{p=$1; for(i=2;i<=NF;i++)
                 {p*=$i}; print p}' ¥
      ) ¥
    | mawk '{print $1%$2}' ¥
    ) ¥
    | grep '^0$' ¥
    | mawk '{print $1}'
```

お名前は「同姓同名」で正しいそうです。同姓同名さんは素朴版、高速版と2つ投稿してくれたのですが、高速版を採用。Linux JMに載っていないmawkがありますが、堅いことは抜き！　引数が100000くらい以上なら他解答に比べて圧倒的な早さで解き終えます。

ん、42行目のpingって何だい？　と思ったら投稿コメントに、テンポラリーファイルが確実に出来るようにするための時間稼ぎなのかい。うーん、zshとかならここはwaitを使ってスマートに行くんだけど、bashじゃ効かないんだよね、これが。引数が小さいと早さを発揮できないのは一つにはこのせいかもしれないね。今回の最高速解答かつ、2つのバージョンを送ってくれた努力を称えて**二段褒与**だ！

さて……、2年半やってきたこのシェルスクリプト大喜利のコーナーなんですけど。ついにこの時が訪れましたよ。何がって？　それじゃ次の投稿いきましょう。

◎Kさんの解答

```
#!/bin/sh
seq 1 $1 |
while read i; do
  p=$(( $(echo $i | sed -e 's/¥([0-9]¥)/¥1*/g' -e
  's/$/1/') ))
  [ $p -ne 0 ] && [ $(($i / $p * $p )) -eq $i ]
  && echo $i
done
```

常連のKさん、やはり今回も投稿してきましたね。内容は、1から順にズッカーマン数の条件を満たしているか検証するというオーソドックスなもの。ただ、変数pがiの約数かどうかを判定するのにiをpで割っ

てｐで掛けてｉに戻るかを見てるあたりは、ちょっとした工夫ですね。

さあ、採用してしまったぞ。というわけで、**一段授与**。これで十段。ついに出たー！十段到達おめでとう‼

そうそう、**筆舌に尽くし難いちんじゅうちゃんグッズ**をプレゼントしなきゃねぇ。考えときます（＾＾；それから、十段獲得者は「師範」の称号を進呈して、また初段からはじまるものということに、**今決めました。**これからも投稿よろしくお願いしまーす。

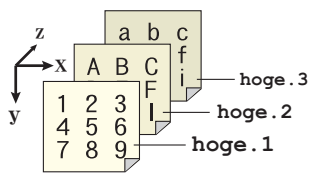
◇　◇　◇

「師範」の特典だからというわけでもないんですが、Ｋさんが更に前回のお題に対する改良版を送ってくれたので紹介します。そのお題とはこれです。

第十回<二問目>

図のような３行３列（列間は半角スペースで区切られている）の３つのファイルがあります。列方向をＸ軸、行方向をＹ軸、ファイル間方向をＺ軸とし、x-z 平面で 90°、180°、270°回転させられるシェルスクリプトをそれぞれ書いてください。

90°回転後の hoge.1 は 1 行目から順に“a A 1”、“d D 4”、“g G 7”となれば正解とします。



元のお題では、回転方向は x-z 平面のみ、かつ対象データサイズは 3*3*3 限定でよかったのですが、前回既に回転方法を一般化した解答を送ってきてました。そして今回、サイズまで一般化できたということで記念に掲載！（編注：掲載の都合で適宜改行してます）

```
#!/bin/sh
#3drotate.sh <input_file_name> <output_file_name> ¥
#            <rotation> <pad>

#get Z; the maximum number of $1's suffix
Z=$(ls $1.[0-9]* | sed -e "s/$1¥.//" | sort -nr |
 head -n 1)

# get X, Y; the maximum number of x, y size
X=0
Y=0
for i in $(seq 1 $Z); do
  [ -f $1.$i ] && ¥
  XY1=$(awk 'BEGIN{X=0}
         {if (X<NF) {X=NF}}
         END{print X, NR}' < $1.$i )
  X1=${XY1% *}
  Y1=${XY1#* }
  [ $X -lt $X1 ] && X=$X1
  [ $Y -lt $Y1 ] && Y=$Y1
done

pad=${4:-@}

tmp_files_name=""
null_files_name=""
for i in $(seq 1 $Z); do
 [ ! -f $1.$i ]  && { touch $1.$i; null_files_name
="$1.$i $null_files_name"; }
 awk -v X=$X -v Y=$Y -v P="$pad" '
  { printf("%s ", $0);
   for(x=NF+1; x<=X; x++) {printf("%s ", P);}
   printf("¥n");}
  END{
   if (NR < Y) {
    for(y=NR+1; y<=Y; y++) {
     for(x=1; x<=X; x++) {printf("%s ", P);}
      printf("¥n");}}}' $1.$i > $$.$i
 tmp_files_name="$$.$i $tmp_files_name"
done

case $3 in
 "xz180"|"zx180"|"yz180"|"zy180"|"xy180"|"yx180")
NX=$X; NY=$Y;;
 "xz90" |"zx270"|"xz270"|"zx90") NX=$Z; NY=$Y;;
 "yz90" |"zy270"|"yz270"|"zy90") NX=$X; NY=$Z;;
 "xy90" |"yx270"|"yx270"|"yx90") NX=$Y; NY=$X;;
esac

function rotate() {
#rotate <input_file_name> <output_file_name> ¥
#       <rotation> <X> <Y> <Z> <NX> <NY>
 for z in $(seq 1 $6); do
  awk -v z=$z '{for (x=1; x<=NF; x++) print x, NR,
z, $x}' $1.$z
 done |
 case $3 in
  "xz90" |"zx270") sort -k1n,1  -k2n,2  -k3nr,3;;
  "xz180"|"zx180") sort -k3nr,3 -k2n,2  -k1nr,1;;
  "xz270"|"zx90" ) sort -k1nr,1 -k2n,2  -k3n,3 ;;
  "yz90" |"zy270") sort -k2n,2  -k3nr,3 -k1n,1 ;;
  "yz180"|"zy180") sort -k3nr,3 -k2nr,2 -k1n,1 ;;
  "yz270"|"zy90" ) sort -k2nr,2 -k3n,3  -k1n,1 ;;
  "xy90" |"yx270") sort -k3n,3  -k1n,1  -k2nr,2;;
  "xy180"|"yx180") sort -k3n,3  -k2nr,2 -k1nr,1;;
  "xy270"|"yx90" ) sort -k3n,3  -k1nr,1 -k2n,2 ;;
 esac |
 awk -v X=$7 -v Y=$8 -v out_file_name="$2" '
  BEGIN{z=1; out=out_file_name"."z}
  { if(NR%X!=0) {printf("%s ", $4) > out}
    else {printf("%s¥n", $4) > out}
    if(NR%(X*Y)==0) {z++;out=out_file_name"."z}}'
}

rotate $$ $2 $3 $X $Y $Z $NX $NY

rm $tmp_files_name
rm $null_files_name
```

これで三次元なテキストデータも問題無く扱えるようになりますな。記念に**初段授与**しますよ。師範としての二週目も投稿お待ちしてます！さて、二問目！

<二問目>

与えられたテキストファイルをセンタリング（中央寄せ）するシェルスクリプトを書いてください。

左寄せなら“sed 's/^ *//'”でいけるし、右寄せも rev コマンドの活用でできます。ではセンタリングはどうやる？ということで出題したんですが。さてどう解きますか？

◎むっつーさんの解答

```
#!/bin/sh

if [ -p /dev/stdin ]; then
  InText="$(cat /dev/stdin)"
else
  exit 1
fi

MaxLength=$( echo "$InText" |while read LINE; do (echo "$LINE" |grep -o . |sort |tr -d "¥n"; echo ) |fold -w 2 |wc -l; done |sort -n |tail -1 )

echo "$InText" |while read LINE;
do

   ## 空行はセンタリングしない
  if [ -z "$LINE" ]; then
    echo ""
    continue
  fi

  LineLength=$(echo $LINE | ( grep -o . |sort |tr -d "¥n"; echo ) |fold -w 2 |wc -l )
  echo -n "$LineLength:"
  seq 1 $( expr $MaxLength - $LineLength ) |xargs -i echo -n " "
  echo "$LINE"

done
```

コメントによると、全角半角混じっている場合……と悩み始めたらゴチャゴチャになっちゃったそうで。

そうか、言われてみれば確かにねぇ。出題時にそこまで想定してなかった！　半角文字ができれば OK。申し訳ない‼　労をねぎらって**二段授与**。これで三段だ。

◎お酒屋さんの解答

```
#!/bin/bash

cat $1                  |
sed 's/^ *//'           |
awk '{print length($0)}' |
sort -k1,1nr            |
head -1                 |
cat - $1                |
sed 's/^ *//'           |
awk 'NR==1{haba=$1};
     NR!=1{yohaku=int((haba-length($0))/2);
           for(i=1;i<=yohaku;i++){printf " "};
           print $0
          }'
```

こちらは、パイプを駆使してうまいことスッキリ書けてますねぇ。length($0) とやっているので全半角の区別無しなわけですが、やはり半角文字だけと割り切ってしまえば簡単に書けるのでしょうか。

うーん、すっきり書けたもう一つのポイントは元データを実ファイルとして二か所 (3 行目と 8 行目) で開いてるからかな。うまい。**一段授与**。只今六段。

◎gori.sh さんの解答

```
#! /bin/sh
sed 's/^/.TH "" "" "" "" /' > /tmp/man$$
man /tmp/man$$ | sed -n 's/^()/  /p' | sed 's/()  $/  /'
rm /tmp/man$$
```

うわー、これはやられた !!! man コマンドを使うのか。うーん、グルー言語の美学。スバラシイ！**三段授与**。

ん？まさか、gori.sh さんも十段到達か。オメデトー！

それじゃ、次回から師範で二周目ってことで。よろしくおねがいします。では、最後のお題にいきます。

<第三問>

過去にディレクトリー掘りまくってオカシな動作を誘う投稿がありました。そんなふうにヘンなことして珍現象を起こしてください。ただし、致命的な損害は与えないように！ くれぐれもね！

さてこれ、「UNIX は知育玩具『こわしたい放題』だ！」と題したお題なんですが、UNIX やコマンド制作者の想定を超えた使い方をした時に起こる変な動作を見つけて楽しもうという趣旨です。これもいくつか投稿きましたが、まぁ皆さんよく見つけますねぇ。

※ 編集部よりご注意

重要な役目を担ったサーバー、その他再起動させられては困るホスト上では、このお題で投稿されたコードは動かさないでください。

ディスクを破壊するなど、再起不能にするような投稿はありませんが、プロセスが壊れるなどしてOSの再起動を余儀なくされる可能性が無いとは言えません。

◎イタローさんの解答

```
$ mkdir hoge
$ cd hoge
$ touch 0
$ for n in $(seq 0 100000); do ln $n $((n+1)); done
```

こーいうお題の時はいつもこの方から来ますね。毎度ありー！　さて、このコマンドを実行に際して確認した動作環境は「Fedora19 上の bash と FreeBSD 9.2 上の sh」とのことです。

なるほど、ハードリンクをどこまで作れるか試したのね。やってみたら 32767 個しか作れなかったよ。OS の制約なのかねぇ。面白い、**一段授与**。

◎イタローさんの解答

```
$ a=$(yes TakasuClinic | head -10000000|tr -d '¥n')
$ export a
$ yes
```

連続投稿。動作環境は同じだそうですよ。

えぇと今度は何だい？環境変数にバカデカい文字列をぶち込んで、他のコマンドに食わせるってわけか。今度は“Argument list too long”になったぞ。へぇ、こんなことになるのか。ただ、true とか false とか、影響受けないコマンドもあるね。引数をとるかとらないかかな？　よし、これも**一段授与**。これで三段だ。

◎猿二号さんの解答

```
$ truncate -s 100000000000000 BIGDATA
$ vi BIGDATA
```

猿二号さんもこの手のお題の常連さんですね。ありがとうございます。動作環境は「truncate コマンドがあるコンピューター上ならたぶんどこでも動きます」とのことでした。

今ハヤリのビッグデータか（違）。えぇと、sオプションの数字は何桁あるんだ。一、十、百、千、万、…、って**100テラかい！**こんなに大きなデータ作ったら1行目を実行した時点でディスクが溢れるでしょと思ったら、違うのね。“ls -lh”って実行するとファイルサイズは 91T とか表示されるものの、実際のディスク領域は食っていないらしい。でも、vi はマジメに読み込もうとするのね。それでメモリを喰いまくる、と。

スワップしたら負けなマシンなのにスワップしまくちゃったじゃないか。プロセス殺すのも一苦労だったよ。あぁ酷い目に遭った。これは**ちょっとキケンな解答だなぁ。動作は自己責任でね！**とはいえ、truncate コマンドの動作は興味深かったし、**一段授与**だ。現在五段。

◎伊佐坂劇物さんの解答

```
$ eval ls $(yes 'cat' | head -3333 | tr -d '¥n') |
  cat
```

ユニークなお名前ですね。投稿コメントによれば、動作環境は bash 3.2、テーマは「bash でパイプは何本繋げるか」とのことです。そして肝心の結果は、3333 本だったそうです。（上記の 3333 を一つ減らして 3332 にすると正常に動くそうです）

へぇー、こりゃまた興味深い。でもこれって「単純に、コマンド一行として書ける文字列長の制約に引っ掛かってるんじゃないの？」とも思ったんだけど、“cat”を“cat -”に替えてもやっぱり 3332 と 3333 の間に境界があるんだよね。どうやらホントにパイプの本数みたい。気になったんで同じホスト上で他のシェルでも試してみたんだけど、例えば dash だともっと多くても平気だし、逆に zsh だと 2000 でもダメだったね。これは、面白い。**初段授与**。

◇　◇　◇

といったろころで本日の大喜利はこれにてお開き！十段達成者が出てめでたいねー。読者の皆さん、投稿してくれた皆さん、今回もありがとうございました。

投稿大募集

次回のお題

一、1 ～ n（n は引数で指定）の自然数の中に存在する「安全素数」を列挙するシェルスクリプトを書いてください。

二、16 行 16 桁のテキストファイルがあります。この外周部に 0~9, A~Z, a~x の 60 個の英数字が記されており、内部（14 × 14）は全て空白文字です。このようなテキストを与えると、外周を時計回りに 1 文字移動させるシェルスクリプトを書いてください。

三、今回の解答に ping コマンドで時間を稼ぐものがありましたが、他の何らかのやり方やコマンドで、時間稼ぎをするための方法はありませんか。ただし、次のようなリクエストがあります。

イ、ある程度の精度で秒数を予測できる（従って単純なループによる時間稼ぎはダメよ）

ロ、珍しいもの大歓迎（sleep とかはちょっとねぇ）

投稿のしかた

お題への解答は、お名前（ペンネーム）、解答したいお題番号、解答スクリプト、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解答可！
尚、次回締め切りは**2月24日（月）午前0時**とします。しかもその間は何度でも解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

お題の投稿も大募集。こっちは締め切りなしでずーっと募集中。そして、考えてくれた方にも段位を授与します。自分で出題して解答するのも、OK！

投稿先

どちらも投稿先は、mag@usp-lab.comです。面白ければ（ワリと）何でもあり！　じゃんじゃん投稿待ってます！

仕事の原点

―シェル魔人―

僕は現在ＩＴ会社の社長をやっているが、これは偶然だ。

時代も昔ということもあったが、コンピュータとの出会いは大学に入ってからで、遅めだった。先輩がパソコンに向かってキーボード入力しているのを見て、「テレビに向かって英語を勉強している」と言って、失笑を買ったくらいだ。

金欲しさに「コンピュータができる」と嘘をついて、ソフト会社のアルバイトを始めた。嘘を真にするために、速攻でアセンブラ言語を覚えた。先輩のパソコンのキーボードの絵を紙に書き写して、タッチタイピングの練習をしながらのスタートであった。今だと笑い話だが、この時の経験がコンピュータの仕事の原点だった。

高校生の時は、叔父さんの仕事をアルバイトで手伝った。叔父さんは縫製の仕事を夫婦でやっていた。僕の仕事は大阪で仕入れた鞄の部材となる布や、仕事先で受け取った鞄製品を叔父さんのトラックに積み込むことだった。

トラックの助手席に乗って、叔父さんから色んな仕事の話を聞かされた。その中で印象に残ったのは、「時間さえ守っていれば仕事になる」「世の中で数を正確に数えることが一番難しい」「誰も見ていなくてもどんなことでも一生懸命やっていると、不思議なことに誰か見ている」というものだった。

当時高校生だった僕は、毎朝学校へ定時に行くのは当たり前だったし、ちょっと数学が出来て得意だったので、数のことは、正直叔父さんが何を言っているのか分からなかった。

しかしある日、僕は大失敗をしてしまった。朝３時でまだ暗く、大阪で布を１５００枚数えてトラックに詰め込む仕事だったのだが、時間がかかった上に、数を間違ってしまった。トラックで何時間もかけて走った契約先で、大好きな叔父さんが謝っている後ろ姿を見て、ああ、自分の責任だ、とものすごく情けない思いをした。

アタマでっかちだった僕は「取り返しのつかない」とはこういうことだ、と思った。アタマの中の話と、実際に手足を動かして時間に間に合わせるというのは、完全に別物だ、そう感じた。

それから、僕は何十という仕事を経験してきたが、その底辺に流れるのは、やはり叔父さんの言葉だ。僕は今でもお酒が入ると、若い人に当時の話をしてしまう。

どんな仕事でもプロがいて、その熟練たるや、すごいものだ。清掃の仕事でも皿洗いでも、ビラ配りでもだ。叔父さんの言葉に従い、どんな仕事でも、誰が見ていなくても僕は一生懸命やることをモットーにしてきた。

それは、今の仕事にもつながっていると思う。それが何であってもだ。

USP MAGAZINE info

USP MAGAZINE の最新・詳細な情報はこちら。
読者アンケートにも是非ご協力を。

USPマガジンFB　検索

編集後記

- 古代から道は人馬の移動により形作られた。誰も通らなければ道は廃れる。いま足下、眼下にある道も無数の何者かが通り続けた証だろう。路上で起きた出来事を逐次書き留めていれば長大な年表ができそうだ。泥酔して寝込んだり迷惑をかけたりした人間は書き切れないほどいたに違いない。年末は特に（柏崎＠エクリュ）
- ところ変われば品変わる。今回 IT 美女図鑑がない理由なのですが、vol.11 制作中に地球の裏側、チリのサンチアゴへ遊びに行ってきたのです。どうだったか？ net-cafe に行ったのですが、2byte の表示できないブラウザとか久しぶりでした。雨は 1 回だけ降りました。8 月ぶりだったそう。（鎌田）
- 早いものでもう年末ですね。毎年年賀状はお絵かきしたものを印刷して送っているんですが、ここ何年かはお正月を大分過ぎてからの作業になってしまってます。来年は午年！今回こそはせめて三が日中にお届けできるように仕上げたいところです (@_@;)…お、おそくても 7 日までには…！！！（星）
- 先月再び沖縄へ出張。経費節約ということで、なるべくバスで移動したが、目的地へ行く便がなかなか来ないことも。知らない土地、暗いバス停で過ぎ去る車の光を眺めつつ、バスを延々と 30 分以上待つ。そして乗ってからも渋滞で進まず、1 時間以上かかってホテルへ。忍耐力はこうして鍛えられる？ それが役に立つ仕事だ、編集者は（青井）
- できた。雑誌じゃなく、前回の編集後記で豪語してたショッピングカートが。当然シェルスクリプト製。PayPal 決済にも DL 販売にも対応。さぁ早くこれで本誌を売りたい！が、特集でご覧の通り多数のツッコミが……。3 年間の取材を通して作法はだいぶ勉強したはずだったのに。早く直して使いたい。（松浦）

USP MAGAZINE 2014 winter

Vol.11

編集長　松浦智之
副編集長　青井大地
編集　柏崎吉一
星春菜
鎌田広子

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 500 円（本体価格 476 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）

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　発行日 2013 年 12 月 20 日
定価 500 円（本体価格 476 円）