作り手は道具を通して使い手の幸せを願い、使い手は道具を通して作り手の幸せに共感する。

ISSN 2187-9664

USP MAGAZINE

2013 winter

よりぬき版

for the sophisticated shell scripters

Tukubai流 コマンド自作文化

TechLION再録
村井純
UNIXとインターネットだぁ!

山本裕介の
Twitter APIで広がる
シェルスクリプトの世界

茶筌匠 3代目 谷村弥三郎氏
に訊く、技術者哲学

IT美女図鑑　つつみともよ

From Editor

ノーム・チョムスキー。この方をご存じの方は少ないのではないでしょうか。科学技術賞などで賛辞されるものの、テレビなどではなかなか見られません。それもそのはず、ブッシュ政権時代のアメリカ政府や、社会に浸透するメディアを痛烈に批判する文献を多く世に出しているからです。本質を突く彼の洞察力の源はどこにあるか、それは〝言語の研究〟なのです。言語学を研究し、社会の倫理・哲学の研究を経て、世界に広がる矛盾したロジックに嘆く。そして彼の言語学における研究は、情報処理の分野でも大きな功績を残しました。大岩先生の連載では、チョムスキーのお話から入ります。

TechLION 再録記事では、インターネットの父である村井純氏のトークライブの模様を収録しています。村井先生はインターネット普及前の偉人達の数々の名言をトークライブ中に紹介されましたが、最後の質問者に対する答えが、まさに名言そのもの。エンジニアが涙する言葉とは。一方、ものづくりをする上で大事なことも教えてくれます。

～　人と社会に受け入れられてこそ　～

IT 美女図鑑に出てくださったつつみさんも、社会学を学ばれているそうではないですか。情報という道具と、多様な人が瞬時につながる社会。受け手の気持ちと、どう関わるのか。今を生きる人には、一番重要な課題なのかもしれません。

特集では、コマンド自作文化というちょっとミステリアスなタイトル。コマンドというと一般的に〝そこにあるもの〟または〝すでにあるもの〟と想定される方が多いかと思います。しかし、本誌では、Open usp Tukubai で〝無いものは作れ〟という自立精神で作られたコマンドたちの生い立ち、すなわちコマンドを作るまでの過程を Tukubai 作者に取材しました。

今号のテーマの一つが「道具」です。身の回りにある「道具」、これを見る価値観に変化が生まれてくるかもしれません。

USP MAGAZINE 編集部

Contents

※ 薄字で記している記事は、よりぬき版には収録されておりません。また、本誌とは収録順序が若干異なります。

特集1

無い物は作れ

Tukubai 流 コマンド自作文化

USPマガジン編集部

各地で Open usp Tukubai を紹介すると、
多くの人にとても興味を引かれることがある。
それは、Unix コマンドを自ら作っていくと言う習慣だ。
コマンドを作るということは、
他言語で考えれば
if や for 等の予約語を追加するようなものかもしれない。
そう考えると確かに興味深い。
だが、Unix の世界ではそれが当初のやり方だった。
コマンドを自作する文化。
この素晴らしさを、改めて伝えたい。

第一章 共通部品でない、道具を作るということ

Open usp Tukubai は、何のために生まれたか

2012 年の今年、USP 研究所は Open usp Tukubai と名付けた Unix シェル向け追加コマンドセット（通称 Tukubai コマンド）をリリースした。

名称の一部となっている “Tukubai” は、このコマンドセットに付けた単なる名前ではない。オープンソースやオープンプラットフォームの「ソース」や「プラットフォーム」などと同様に、ある一つの概念を指し示す用語なのだ。この「ある一つの概念」を端的に言い表す用語というものが存在しなかったため、USP 研究所がコマンドセットのリリースに合わせて新たに定義したものなのである。

本誌もこれまでこの語を度々用いてきたが、ここで改めて Tukubai の意味を記すことにする。

Tukubai とは

Unix 哲学や “Software Tools”（B.W.Kernighan、P.J.Plauger 著）等の作法を忠実に受け継ぎ、進化させたシステム構築法。同音である蹲（蹲踞とも記す）という道具が用いられる茶道の精神に通じるものがあることからこの名を付けた。

つまり Tukubai は一種の思想であり、その源流は、Unix を生み出した先人たちの哲学にある。そしてこの哲学を受け継ぎ、進化させ、USP 研究所なりに具現化したものが Open usp Tukubai なのだ。将来 Tukubai という語が一般化し、Tukubai の思想に基づいてある人 X がコマンドセットをリリースするなら、それは Open X Tukubai と呼ぶべき存在になるかもしれない。

従って Tukubai コマンドは、実はその表面的な姿（書式や仕様、種類）が重要なのではない。重要なのは、コマンドセット全体としてどんな役割を果たしているかである。そして Tukubai コマンドが果たすべき役割とは、ソフトウェア開発における「道具」になることである。

「道具とは何か」を理解する

Tukubai が指し示す「道具」は、共通部品とは明確に区別される。では、共通部品ではない道具とは何なのか。まず道具が持ち合わせるべき三つの性質を記す。

一、単機能であって汎用性がある。
二、インターフェースが決まっているが、逆にそれさえ守っていれば他には一切の制約がない。
三、同じものを発明する気を起こさせない。

更に付け加えるなら、「作ってみればあって当たり前の存在」、「何処の誰もが使えるもの」などとも言いたいところだ。これらの性質を満たしているものは、世の中の様々な分野において、作業を効率的に熟（こな）すうえで必要不可欠なものに成り得る。勿論、ソフトウェアの世界においても例外ではない。

身近にある道具と呼ばれるものを色々思い浮かべてもらいたい。それらがこの三つの性質を満たしているかどうか考えてみれば、ここで言わんとしている道具がどういうものであるかが見えてくるだろう。

身の回りにある例

例えば西洋料理を食す際に使うナイフやフォーク、スプーンの類。これは道具であろう。西洋料理を食す時は誰もが使うし、同種のものを発明する気も起らない。インターフェースについて考えてみても、どれも片手で持って使うようにできているし、皆棒状の形で大きさや重さ等も概ね揃えられている。もし違っていたら、揃えるべく、きっと誰かが同じ類のものを発明しているに違いない。

一方で共通部品的な物とは、複数の道具をくっつけてみたり、よくある道具の形をちょっとだけ変形してみたり、といった俗に言うアイデア商品の大部分のものではないだろうか。それらは特定の作業には驚く程の利便性を発揮するが、汎用性が乏しいために歴史の一部となるほどには普及しない。そして似たようなものが再発明されたりもする。

“Software Tools” の邦訳版「ソフトウェア作法」（共立出版）。30 年以上発行され続けている名書である。

ソフトウェアにおける例

そして、ソフトウェアにも道具的なものと共通部品的なものがある。ある程度汎用性があり、再利用可能なコードを集めたものとされるライブラリーは、その両方を持ち得る。

例えば、Microsoft Visual C++ には通称 MFC と呼ばれるクラスライブラリーがある。これは C++ による Windows プログラミングを容易にするためのものであり、数百個ものクラスが用意されている。しかしそれらを全てを把握して使いこなしている人は恐らく少数だろう。その一方で、C 言語の文字列変数の扱いづらさを補完する CString のように、MFC を利用する人の大多数が利用するクラスもある。このような違いというのも、先に記した三つの性質の有無によるところが大きい。

ユーティリティー、アプリケーション

類似の用語に、ユーティリティー、アプリケーションといったものがあるが、これらもここで言う道具とは違う。ユーティリティーは例えば、バックアップソフトとかテキストエディターといった物を指すが、単機能とは限らないし、インターフェースも様々だし、同様のものが様々作られる。アプリケーションはさらに複合的な機能を有する物を指し、それ単体の使用で大きな目的を達成できる。これらも勿論有用なものだが、単体では目的を達成できない代わりに組み合わせ次第で如何なる目的にも対応できる道具とは方向性の違うものだ。

道具とは何か、ご理解いただけただろうか。これが道具であるかどうかという上記の例は、明確な基準に基づいたものではないため、あまり心地良いものではないかもしれない。しかし確実に言えることは、三つの性質を満たしていると思える人が多くいるかどうかだ。多くの人が支持するからこそ物は道具になれる。逆に、それは見る立場によって判断が分かれ得るものであるから、特定の地域や時代、或いは特定の分野でのみ道具として認知されるものもある。

Unix における道具とは

既に述べたとおり、Tukubai コマンドは道具としての役割を果たすべく作られたものである。しかしそれ以前に、Unix シェルに用意されている標準コマンドには、先に記した三つの性質を満たす道具的なものが数多くある。

例えばテキストフィルターに分類される AWK, grep, sed, tr などは道具的であると言えるだろう。一方で、パスワードを設定する passwd コマンドやテキストエディターである vi[*1] といったコマンドは誰もが使うところであるが、使い方が既に決められていて汎用性は無く、インターフェースが決まっているかという性質に照らし合わせてみても当てはまり難い。これらはむしろユーティリティーと呼ぶべきものであろう。

このように Unix のコマンドにおいても、道具的なものとユーティリティー・アプリケーション的なものがある。そして、道具的と呼ぶべき Unix コマンドは、どれも共通した下記の作法を概ね守っている。

◆ データのやりとりに標準入出力を使う

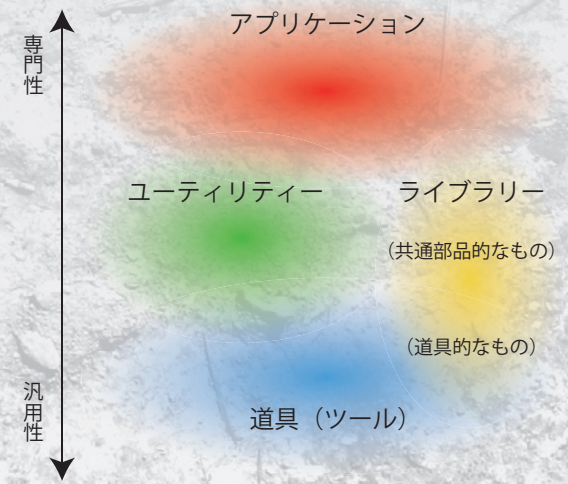
◆ 入出力データをテキストデータとする

道具は、組み合わせて使うことで真価が発揮される。従って、組み合わせて使えるようにすべく、このような作法が生まれたのではないだろうか。Unix において偉大な発明の一つであるパイプ（パイプライン）も、そのような背景から生まれたに違いない。

そして Unix シェルやシェルスクリプトも上記の作法を守るうえで実に適した仕様になっている。道具たるコマンドを予約語のような感覚で呼び出せ、組み合わせられる。時には “if [” などのように、予約語とコマンドが密接に結び付くこともある。

こういった仕様は、Unix 発明者達の思想の表れであって、「道具をもってして問題を解決するのが Unix シェル」ということを言わんとしているのではないだろうか。

*1 vi は好みが別れそうだが、何らかのテキストエディターは使うだろう。



道具に類する用語の整理。ユーティリティー、アプリケーションと呼ばれるものになるに従って、汎用的な用途から専門的な用途へと変化する。ライブラリーは部品であるが、汎用的なものもあれば専門的なものもある。

第二章 sm2 コマンドという道具の発明

道具になることを目指して作られた Open usp Tukubai。それを構成している 49 のコマンド群。実際これらはどのようにして生まれたのだろうか。そのうちの一つで、表の合計値を求める sm2 コマンドの生い立ちを取材した。

sm2 コマンドとは

Open usp Tukubai の sm で始まるコマンドは sum-up という表計算の世界で頻出の操作を行うシリーズだ。sm2 の場合、例えば下記のように生徒の各教科のテスト成績表があると、これに基づいて生徒毎の合計を求めるなどという用途に用いられる。

飯山満くん	英語	75
飯山満くん	数学	98
飯山満くん	国語	74
近江舞子さん	英語	95
近江舞子さん	数学	84
近江舞子さん	国語	80
中山香さん	英語	93
中山香さん	数学	72
中山香さん	国語	97

▶

飯山満くん	247
近江舞子さん	259
中山香さん	262

sm2：数値を縦（行）に合計

この例では第 1 フィールド（名前）が同じ行の第 3 フィールド（点数）合計し、各個人の合計点を出している。

それではこの sm2 コマンドの進化の過程を見ていくことにしよう。

1．未来の開発者を想像してみる

頻出の操作であるなら迷うことなく作ればいいじゃないかと思うだろう。いやいや、そこで冷静にならなくてはならない。「道具として本当に必要なものなのだろうか」と、考えるのだ。「既存コマンドをちょっと工夫して使えば無くてもいいかも」、そう思えるかもしれないから。

あれば便利だろうと言ってコマンドを乱立させても、数が多いと覚えきれない。そういう宿命がある。すると、存在意義の薄いもの、使用頻度の低いものからどんどん忘れ去られていく。[*2]

USP 研究所にも、忘れ去られて今やハードディスクの肥やしとなっているコマンドが何百とあるという。[*3] そうなってしまう未来が思い浮かぶなら、作るだけ無駄だ。無駄なだけでなく、そのコマンドの含まれるコードを読む未来の開発者が苦労することになる。「あれ、このコマンド……。どういう仕様なんだ？知ってるコマンドで事足りるならそれで済ませといてくれればいいのに」と。

そう、コマンドを作る時には未来の開発者の姿を思い浮かべることが重要なのだ。

> § こんなのがあったらきっと便利だろうなー。
> § でも、いっぱい作っちゃったら不便だろうなー。

と。そんな葛藤の末、それでも「未来の開発者達も、これがあればきっと便利になるはず！」と思えたらなら作ればいい。

> § 末永く使われますように。

そう心で祈りながら……。

「そんなコマンド本当に必要なの？」USP 社内では毎週、エンジニアたちが寄ってたかって未来に求められるコマンドについて語り合う「コマンド研究会」と呼ばれる会議が開かれている。

2．作業指示の言葉をよく観察してみる

作るからには末永く使われるものになれなければ迷惑だ。だから使いやすい仕様を心掛ける。コマンドの場合、書式が作業指示の写像になっていることが重要だという。では sm2 はそうなっているのか？

sm2 を作り始める前、表計算を行っていた作業現場では次のようなセリフが飛び交っていたという。

> その表の 1 列目から 2 列目をキーにして 3 列目から 6 列目の合計を出して！

だから sm2 は、この文章をほぼ過不足無く表現できるよ

*2 どの言語も予約語が数十〜百数十個程度に落ち着いている理由は、そういうところにあるのだろう。
*3 次章でいくつか紹介しよう。

うな書式にした。結果、決められた sm2 の書式がこうだ。

```
Usage : sm2 [+count] <k1> <k2> <s1> <s2> <file>
```

先のセリフをこの書式に当てはめればこうなる。

```
sm2 1 2 3 6 その表
```

オプションの "+count" を除けば、「合計を出す」という動詞がコマンド名に、他の名詞が書式にピタリと当てはまっている。このように現場で頻繁に飛び交っている指示の写像になっていれば、覚え易いし、また頻繁ゆえ忘れ去られる可能性も少なく、道具として認知される存在に成り得る。

単純でも頻繁に飛び交う動詞なら作る

今、「動詞がコマンド名になっている」と述べたことに注目してもらいたい。

sm2 の話から少し外れるが、例え既存のコマンドのごく簡単な応用で実現できることであっても、敢えてコマンド化した方がよい場合がある。それは、作業指示として頻繁に耳にする動詞があった場合だ。

例えば Tukubai コマンドに gyo というものがある。これは単にテキストの行数を求めるだけであり、"wc -l | awk '{print $1}'" とか "awk 'END{print NR}'" などとやれば簡単に求められる。それでもコマンド、つまり道具として成立したのは、「行数求めて」という指示（動詞）が現場で頻繁に飛び交っていたこと、そして gyo コマンドの仕様がピタリとその指示の写像になれていたからである。

このケースからも、言葉の観察がいかに大切かということがわかる。

オプションの扱い

sm2 には一つだけオプションがある。これは各々の合計を出すのに使った元の行が何行あったのかを併記するものだ。最初の成績表の場合、各生徒の受けた科目は三つだったので 3 という数字を返す。後続のコマンドで平均点を求めたいという場合に有効だ。

しかし Tukubai 的には、どうしようもない場合でもなければオプションなど殆ど付けないという。オプションもコマンド同様、乱立させても覚えきれずに忘れ去られるからだ。

例えば有名な ls コマンドの -r オプションの働きを知っているだろうか？ファイルの表示順が通常なら文字コード昇順であるところを降順にするものだ。しかし、文字コード順ではなくて大文字小文字を区別しない（辞書的な）降順にしたければ……、結局 sort コマンドを併用するしかない。ならばそんなマイナーなオプションなど用意せず始めから sort コマンド併用でいいじゃないか、という意見が出てくるのと同じことだ。

オプションもまた、付けるべきか付けざるべきか。或いは増やすぐらいなら別コマンドにすべきか。よく検討したい。

最後に、sm2 の書式に対して抱くであろう疑問について答えておこう。例えば「3 番目から 5 番目、それから 7 番目の合計を出して」というように、k1 ～ k2 や s1 ～ s2 が不連続な指示が与えられたらどうするのかというものだ。確かに sm2 の書式では表現できない。では改良を検討しているのかというと sm2 にその予定は無いらしい。この判断も現場で飛び交うセリフに基づいている。そのような指示は稀だったのだ。そんなレアケースのために書式を複雑にしたら途端に使い辛くなる。もしそんな指示が来るなら self コマンド等と組み合わせ、列が連続するように予め入れ替えるべきだ。こういったコマンドに持たせる機能のさじ加減を見極める上でも作業指示の観察は重要なのだ。

新機能が求められた時、新規コマンドで対応するか、追加オプションで対応するか。これも重要な議題だ。

3．始めは軽量言語（LL）で実装してみる

Open usp Tukubai の有償版（ビジネス版）のコマンドの殆どは C 言語で実装されているが、最初はどれも AWK やシェルスクリプトで作られたという。まだ定番の道具になるかわからず、廃れたり、試行錯誤によって仕様が変わる可能性も考えれば、未来の開発者にとってもコマンドの動作が追いやすい言語を使うべきだ。

こうして、sm2 も当初は AWK で実装された。残念ながら AWK で書かれていた頃のバージョンは発掘できなかったが、+count オプションをサポートしないごく簡易的なものなら**リスト1**のように直ぐに書き起こせてしまう。

sum-up という操作は表計算では頻出なので、実際 sm2 は多用され、重宝される道具になっているのだが、こうして 50 行程で書けてしまう。

全てがそうとは限らないが、優れた道具とは概してこのように単純な構造をしているのだろう。もし何百行にも及ぶようなものができてしまったら、それは単機能なものになっておらず、道具として洗練されていないのかもしれない。

リスト1．当時の sm2 を AWK で復元してみたもの

```
#! /usr/bin/awk -f

BEGIN {
  if (ARGC < 4)
    exit 1
  k1=ARGV[1]
  k2=ARGV[2]
  s1=ARGV[3]
  s2=ARGV[4]
  if (k1 k2 s1 s2 ~ /[^0-9]/)
    exit 1
  if ((k1 > k2) || (s1 > s2))
    exit 1
  if ((k1 < 1) || (s1 < 1))
    exit 1
  file = (ARGV[5] != "") ? ARGV[5] : "-"
  value[0]=0
  delete value[0]

  if (getline < file) {
    key_old = $k1
    for (i=k1+1; i<=k2; i++)
      key_old = key_old " " $i
    for (i=s1; i<=s2; i++)
      value[i] += $i
    while (getline < file) {
      key = $k1
      for (i=k1+1; i<=k2; i++)
        key = key " " $i
      if (key != key_old) {
        print_total()
        for (i in value)
          delete value[i]
      }
      for (i=s1; i<=s2; i++)
        value[i] += $i
      key_old = key
    }
  }

  print_total()
}

function print_total() {
  printf("%s",key_old)
  for (i=s1; i<=s2; i++)
    printf(" %d", value[i])
  print ""
}
```

4．C 言語への移植は、本当に必要とされてから

その後、sm2 コマンドは C 言語へと移植されていった。[*4] 次第に道具としの必要性が認知され、不可欠な存在となって、そこで初めてこのコマンドに対してスピードを求められるようになったからだ。

C 言語に移植後は、C 言語の鬼によって更なる高速化のための改良が続けられているという。高速化の作業に専念する為にも、LL 版で仕様をきっちり固めておくべきであろう。

ソースファイルは大抵一つ

sm2 の C 言語版ソースファイルは、ライセンスの都合により残念ながら掲載できないが、ソースファイルは sm2.c の 1 つだけだ。[*5] これは Tukubai コマンドの殆どに言えることである。もっとも、AWK 版ソースの規模を見れば自明だろうし、同規模の Unix 標準コマンドの多くも一つだ。

一つなので Makefile は不要。"cc -o sm2 sm2.c" などと書き、コンパイルが通るまで数秒待てば完成だ。

こんなあきれるほど簡単なコマンドやシェルスクリプトによって、一部の大手企業の業務データが管理されている [*6] と聞かされたら唖然するに違いない。しかし、優れた「道具」とは、そんなことをも可能にしてしまうのだ。

*4 これが、ビジネス版の usp Tukubai として現在に至る。
*5 もちろん標準ライブラリー等をインクルードしてはいるが。
*6 通信や WebUI 周りが必要な箇所では、Apache 等の HTTP サーバーアプリケーションのお世話になるのだが。

第三章 道具を夢見たコマンドたち

USP 研究所は、前時代を含めてこれまで 2000 個にも及ぶコマンドを作ってきたという。多い日は 1 日で 10 個作ったこともあったのだとか。しかしそれら大半のコマンドが淘汰されていったことは、今の Open usp Tukubai のコマンドの数を見れば容易に想像がつく。

「道具とは何たるか」などと偉そうなことを語っておきながらこれは一体どういうことか！とツッコミを入れたくなるところではあるが、そういう歴史があるからこそ道具の本質を見極め、そして Tukubai の思想を確立させられたのだろう。

そこでこの章では、そんな道具を夢見て生み出されながら淘汰されていったコマンドや、これから道具として活躍する可能性を秘めたコマンドなど、いくつか紹介していこう。

その１．yuniq コマンド

uniq コマンドの横方向版「横 uniq」ということで登場したコマンドだ。その名のとおり、横方向に uniq がなされる。

使い方と動作例

```
〉echo "aa bb bb bb bb bb1 cc" | yuniq
aa bb bb1 cc
〉echo "aa bb bb bb cc bb" | yuniq
aa bb cc bb
〉
```

スペース区切りで、同じ綴りの文字列が連続してるとそれを一つに集約する。ただし、連続せずに出現したものに関しては集約されない。このあたりの仕様は uniq コマンドと同様に作られている。

ソースコード

リスト２に全体を掲載する。勿論 cc でコンパイル可能だ。

道具になれなかった理由とは

yuniq 無しに横方向 uniq をせよ、と言われたらどうやる

リスト２．yuniq.c（横ユニーク）…横方法に uniq 処理を行うコマンドのソースファイル

```
 1 /*                                                  */
 2 /* yuniq.c 〉〉〉 Ｃ言語版ＹＵＮＩＱ                 */
 3 /*                                                  */
 4 /* Usage : yuniq < infile                         */
 5 /*                                                  */
 6 /* Written by N.Tounaka (usb-lab)                 */
 7
 8 #include <stdio.h>
 9 #include <string.h>
10
11 #define BUFMAX 1024
12 char RECORD[BUFMAX],OUTBUF[BUFMAX];
13
14 void trim();
15 char *edit();
16
17 main(int argc,char **argv)
18 {
19 FILE *fp;
20
21     /* 入力ファイルの取得 */
22     if(1 != argc)    {
23       if(NULL == (fp = fopen(argv[1],"r"))) {
24         fprintf(stderr,"Error : %s ファイルがオープンできません.¥n",argv[1]);
25         exit(1);
26       }
27     }
28     else fp = stdin;
29
30     /* 各レコード処理 */
31     while(NULL != fgets(RECORD,BUFMAX,fp))  {
32       trim(RECORD);       /* 要BOF対策! */
33           strcpy(OUTBUF,edit(RECORD));
34       printf("%s¥n",OUTBUF);
35     }
36
37     /* 読み取りエラーチェック */
38     if(!feof(fp) || ferror(fp)) {
39       fprintf(stderr,"Error : 入力ファイル読み取りエラー.¥n");
40       exit(1);
41     }
42
43     /* 終了 */
44     exit(0);
45 }
46
47
48 /* レコードのゴミを取る */
49 void trim(char *buf)
50 {
51
52     /* 行末の改行文字を削除 */
53     if('¥n' == *(buf+strlen(buf)-1)) *(buf+strlen(buf)-1) = '¥0';
54
55     /* 行末の空白を削除 */
```

次ページへ続く→

```
56      /* 空白だけからなるデータへの対応が必要! */
57      while(' ' == *(buf+strlen(buf)-1)) *(buf+strl
    en(buf)-1) = '¥0';
58
59      return;
60  }
61
62
63  /* yuniq 処理を行う */
64  char *edit(char *buf)
65  {
66  char wbuf[BUFMAX],rbuf[BUFMAX]; /* 要BOF対策! */
67  char old[100],new[100];
68  char *p,*q,*r;
69
70      /* 作業領域の初期化 */
71      p = wbuf; strcpy(p,buf);
72      r = rbuf; *r = '¥0';
73      *old = *new = '¥0';
74
75      /* 各フィールドを切り出す */
76      while(NULL != (q = strtok(p, " "))) {
77
78        /* 前後のフィールドの受渡し */
79            strcpy(old,new);
80        strcpy(new,q);
81
82        /* 前のフィールドと異なっていたら連結 */
83        if(NULL != strcmp(old,new))    {
84          strcat(r,new); strncat(r, " ",1);
85        }
86
87        p = NULL;
88      }
89
90      /* 最後の空白を削除する */
91      *(r+strlen(r)-1) = '¥0';
92
93      return r;
94  }
```

だろうか。Tukubai コマンドを使えば次のように書ける。

```
> echo uniqしたい文字列 | tarr | uniq | yarr
```

つまり、単語を一旦縦に並べてソートして、横に並べ直せばいいというわけだ。

横方向 uniq の使用頻度がそう高くなかったために、ユニケージエンジニア達もこのコマンドを覚える機会なかった。そのうえ、たまに横方向 uniq が必要になった時も tarr → uniq → yarr の組み合わせで事足りてしまったためにいつしか忘れ去られてしまった。

その2．exist コマンド

名前のとおり、ファイルの有無を返すコマンドだ。「それだけ？」と思うかもしれないがそれだけだ。ただし、標準出力から有無を調べたいファイルを一覧として受け取れるようなっている。

使い方と動作例

```
> exist /etc/resolv.conf
1
> exist /etc/no_such_file
0
> echo "/etc/resolv.conf"  >  filelist.txt
> echo "/etc/no_such_file" >> filelist.txt
> exist -display < filelist.txt
/etc/resolv.conf
>
```

ファイル名を引数で直接指定すればそのファイルの有無を 1（＝ある）または 0（＝なし）で返してくれる。一方 "-display" というオプションを付け、有無を調べたいファイルリストを標準入力から渡せば、存在しないファイルがフィルタリングされ、存在するファイルのみなって標準出力される。（"-display2" オプションにすると動作が反転する）

ソースコード

残念ではあるが長いため割愛させていただく。しかしながら、中身は C 言語で書かれている。

道具になれなかった理由とは

ファイルの有無を確認したいなら、わざわざ新しいコマンド覚えて使わずともシェルスクリプトのファイル演算子（"-f" や "-e"）を使えばいいと誰でも想像が付く。それゆえに廃れていったのだろう。

「そんな未来が容易に想像できるにも関わらず、何故わざわざこんなコマンドを作るという愚行に及んだんだ！」そう思うかもしれない。しかし実は、擁護すべきある一つの意図があった。

複数あるファイルの有無確認を、シェルスクリプトや標準の Unix コマンドの範囲で行おうとするとループ構文が必要になる。[*7] しかし、これはパイプで処理をこなす合理性を追求するユニケージ開発手法からすると避けたい手段だったのだ。ゆえに、このコマンドに関しては今でもこのために要・不要論が巻き起こるのだという。

*7 実は "xargs ls -1 2>/dev/null" などとやれば出来なくはないのだが、内部的にエラーが起こり、戻り値も 0 にはならない。従って最善の方法とは言い難いのだ。

その３．seniq コマンド

由来は “select field and uniq” だという。つまり、今の Tukubai の self コマンド＋ uniq コマンドだというのだが。

使い方と動作例

```
> cat 週末の予定.txt
先週末 朝方 自宅 部屋を掃除
先週末 午後 自宅 DVD鑑賞
先週末 夜間 自宅 焼肉パーティー
来週末 終日 都内 美術館巡り
> cat 週末の予定.txt | seniq 1 3
先週末 朝方 自宅 部屋を掃除
来週末 終日 都内 美術館巡り
>
```

基本的な機能は uniq コマンド同様「行の集約」であるが、上記の動作例の場合、最初の３行は引数で指定された第１、第３フィールドがいずれも一致しているので（最初の行に）集約される。一方、４行目は第３フィールドが一致しておらず、この行だけ集約されずに出力されたというわけだ。

ソースコード

リスト３に全体を掲載する。珍しいことにＣシェルで書かれている。[*8]

このコマンドの意図

こんな特殊な働きをするコマンド、一体どんな用途を想定して作ったというのだろうか。それは、例えばこういう用途である。

> **銘柄 日付 時刻 株価**
> という４フィールドから構成された株式チャート表がある。銘柄・日時毎に取引開始から終了まで１分刻みで株価が記録されている。この表から各銘柄の毎日の始値を抽出してもらいたい。

この時、seniq １２とすれば、銘柄・日付毎の始値を抽出

*8 本誌はシェルスクリプトをメインに扱う雑誌でありながら、Ｃシェルによるソースコードが登場するのは何とこれが初めてである！

リスト３．seniq…一部のフィールドだけを見て uniq 処理を行うコマンド（なんとＣシェルで書かれている）

```
#!/bin/csh -ef
#
# seniq >>> 指定フィールドのみでuniq処理を行う。
#         (seniq >> select field and uniq)
#
# Usage : seniq f1 f2 ... < infile > outfile
#
# Written by N.Tounaka (usp-lab)

#プロセスｉｄの取得
set tmp = $$

#作業ディレクトリの設定
set d   = /tmp

#コマンド書式の表示
if($#argv == 0) then
  echo2 "Usage : seniq f1 f2 ... < infile > outfile"
  exit 1
endif

#入力ファイルの同定
if(! -e $argv[$#argv]) then
  cat - > $d/$tmp-01
  set file = $d/$tmp-01
  set dif  = 0
  goto NEXT
endif

set m = 1
while(1)
if(! -e $argv[$m]) then
    @ m ++
else
  set file = "$argv[$m-$#argv]"
  @ dif  = $#argv - $m
  @ dif ++
  break
endif
end

NEXT:
#ファイルの存在チェック
if(! -e $file) then
  echo2 "Error : $file ファイルがありません"
  exit 1
endif

@ n = $#argv - $dif

#セニーク編集をするＡＷＫスクリプト
cat << FIN > $d/$tmp-02
NR==1 {
  a="$argv[1-$n]"; n=split(a,x," ")
  m=1
  for(i=1;i<=n;i++) {
          if(x[i]~/¥//) {
    start=transnumber(substr(x[i],1,index(x[i],"/")-1))
    last =transnumber(substr(x[i],index(x[i],"/")+1))
    for(k=start;k<=last;k++) { y[m++]=k }
    }
```

次ページへ続く→

```
62      else  y[m++]=transnumber(x[i])
63    }
64    m--
65    ore=nre=¥$0; old=new=makerecord()
66  }
67
68  {
69    new=makerecord()
70    if(old != new)  { print ore; old=new; ore=¥$0 }
71  }
72
73  END { if(flg==1) exit; print ore }
74
75  function makerecord() {
76    s=""; for(i=1;i<=NF;i++)  {
77      for(j=1;j<=m;j++) if(i==y[j]) { s=s " " ¥$i;
    break }
78      }
79    return substr(s,2)
80  }
81
82  function transnumber(q) {
83    gsub("NF",NF,q)
84    pp=index(q,"-")
85    if(pp != 0) q=substr(q,1,pp-1)-substr(q,pp+1)
86    return q+0
87  }
88  FIN
89
90  #実際の編集
91  awk -f $d/$tmp-02 $file
92
93  #一時ファイルの除去
94  #rm $d/$tmp-*
95
96  #正常終了
97  exit 0
```

できるというわけだ。

このように、ある項目の先頭行だけを取り出してもらいたいという需要はよくあることである。しかしながら、これを既存 Unix コマンドやその簡単な組み合わせで実現することは難しく、そうした経緯で生み出されたコマンドだったのだ。

道具になれなかった理由とは

seniq 自体は淘汰されたが、同様のものが "getfirst" という名前の Tukubai コマンドとなってリリースされた。

「名が体を表していない」ために名前が変えられたということだ。確かに getfirst という名の方が直感性に優れている。そして、兄弟コマンドとして "getlast" もリリースされた。

また一致を確かめたいフィールドの指定方法も、個々に列挙するのではなく範囲指定に変更された。これは前章で述べた sm2 の書式の決まり方と同様の理由であろう。

その４．sm3 コマンド

Tukubai コマンド一覧を見ると、sm2 の次が sm4、sm5 になっており「sm1 や sm3 は何処へ行った !?」と思うことだろう。そう、それら欠番コマンドは道具になれずに淘汰されていったコマンドなのだ。

sm3 は、ソートせずに sum-up を行うコマンドであった。

これもソースファイルが長過ぎて掲載できないのだが、機能だけでも是非紹介しておきたい。

使い方と動作例

例えば 2012 年日本シリーズのスコアがイニング毎に記録されたファイル (NipponSeries2012.txt) があったとしよう。

```
> cat NipponSeries2012.txt
第1戦 1回表 日ハム 0
第1戦 1回裏 巨人 0
第1戦 2回表 日ハム 0
第1戦 2回裏 巨人 0
      :
第2戦 1回表 日ハム 0
第2戦 1回裏 巨人 1
      :
      :
第5戦 2回表 巨人 2
第5戦 2回裏 日ハム 1
第5戦 3回表 巨人 3
第5戦 3回裏 日ハム 1
      :
      :
第6戦 9回表 日ハム 0
第6戦 9回裏 巨人 0
>
```

この表から 1 戦毎のスコア合計を求めたい場合、sm3 があれば次のようにして一発で求めることができる。

```
> sm3 1 1 3 3 4 4 NipponSeries2012.txt
第1戦 巨人 8
第1戦 日ハム 1
第2戦 巨人 1
第2戦 日ハム 0
      :
第6戦 巨人 4
第6戦 日ハム 3
>
```

現在リリースされている Tukubai コマンドの範囲でやるなら、delf でイニングのフィールドを消し、更に試合番号とチーム名フィールドでソートしてから sm2 に掛ける必要がある。

```
〉delf 2 NipponSeries2012.txt | sort -k1,2 | sm2 1
2 3 3
第1戦 巨人 8
第1戦 日ハム 1
第2戦 巨人 1
第2戦 日ハム 0
      :
第6戦 巨人 4
第6戦 日ハム 3
〉
```

道具になれなかった理由とは

上記の代替例で示したとおり、簡単な記述で他のコマンドに置き換えられたからだ。このように単機能でないコマンドはよほど頻繁に用いられない限り、淘汰される運命にある。

ちなみに、もう一つの欠番コマンドである sm1 について軽く紹介しておくと、これは横方向の sum-up を行うためのものだったという。これは先程の seniq コマンドと同様の運命を辿り、ysum（横サム）コマンドとなって現在に至る。

その５．dayslash コマンド

これは淘汰されたコマンドではなく、将来 Tukubai コマンドへ追加すべき候補として現在有力視されているものだ。

8 桁の年月日文字列（YYYYMMDD）を与えると年と月と日の間にスラッシュ（/）を挿入するというものだという。

使い方と動作例

例えば第 1 フィールドに年月日、第 2 フィールドにその日が誕生日である人の名前が書いてあるファイル (birthdays.txt) があったとする。

これを、第一フィールドを “yyyy/mm/dd” というフォーマットに直したい旨を示した dayslash コマンドに与えると、年月日の間にスラッシュが入る。

```
〉cat birthdays.txt
19410809 Alfred_Vaino_Aho
19420101 Brian_Kernighan
19420806 Peter_Jay_Weinberger
〉dayslash yyyy/mm/dd 1 birthdays.txt
1941/08/09 Alfred_Vaino_Aho
1942/01/01 Brian_Kernighan
1942/08/06 Peter_Jay_Weinberger
〉
```

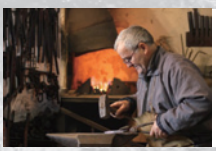
道具になれるかもしれない理由とは

確かに、YYYYMMDD 形式の年月日値にスラッシュを挿入するだけなら AWK でだってできる。

```
〉awk ´{print substr($1,1,4) "/" substr($1,5,2) "/"
substr($1,7,2), $2}´ birthdays.txt
1941/08/09 Alfred_Vaino_Aho
1942/01/01 Brian_Kernighan
1942/08/06 Peter_Jay_Weinberger
〉
```

それでもこのコマンドが有力視されているのはこのようにして日付フォーマットを変更したいという需要が多く、その度にこのような長い AWK の記述を強いられるのが苦痛だからだという。

しかし、そういう思いでこれまでいくつものコマンドが作り出され、消えていった。果たしてこのコマンドは、そんな淘汰の波を乗り越え、将来 Tukubai コマンドに追加されることになるのか、注目してみたい。



コマンドは、臆さず作れ。大人気ない大人にならぬ為に

今でこそ道具の何たるかを語っている USP 研究所も実は、こんなふうにして色々と知られざるコマンドを作っていた。

機能として洗練されていなかったり、或いはソースコードの書かれ方として洗練されていなかったり……。そんなものを公開するのは恥ずかしいとも言われたが、お願いして今回それらを敢えて公開してもらった。編集部としては、コマンド自作の文化を広めるには必要なことだと考えたからだ。

当たり前のことだが、初めから洗練された道具など誰にも作れない。茶道で用いる茶筌も、一人前に作れるまでに 10 年以上かかるという（→ 24 ページ記事）。プログラミングとて同じであり、コマンド自作によるシステム構築法である Tukubai もまた然りである。

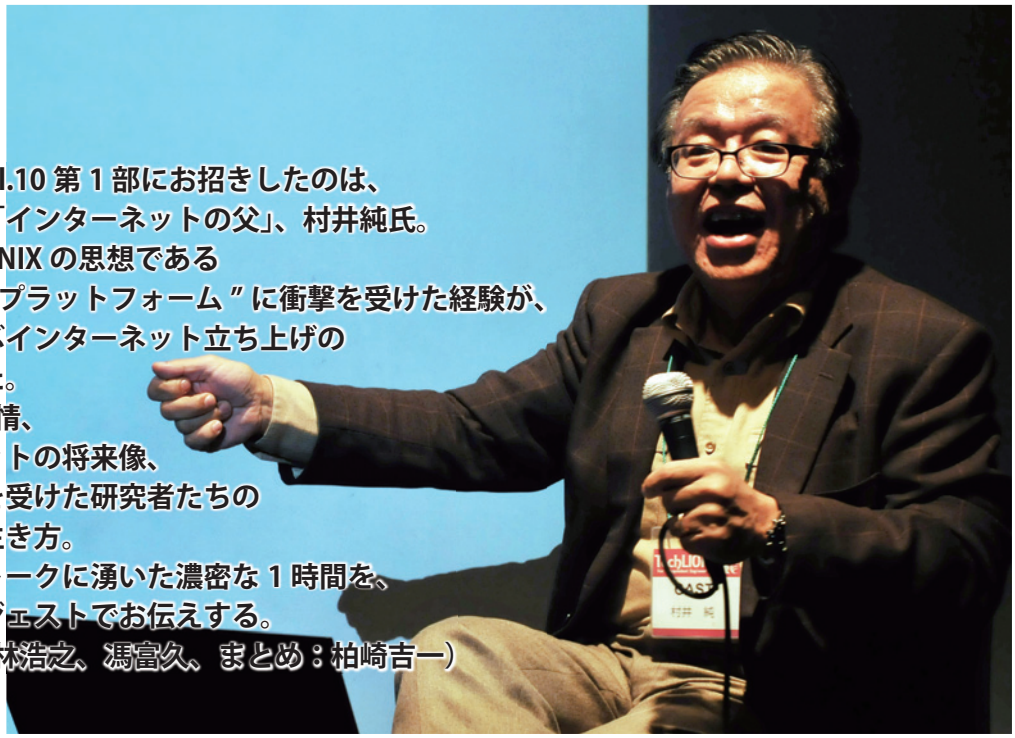
様々な言語でプログラミングを慣れ親しむのと全く同じようにして、Tukubai にも慣れ親しんでもらいたい。即ち、「コマンド自作」を特別視せず気軽に始めてもらいたい。これこそが、真に Unix を使いこなすということなのだから。

最初はそれで、不恰好なものが出来ることだろう。しかし気にすることなど全く無い。手を動かさなければ、いつまでたっても大人気ないコードを生みだすプログラマーを卒業することはできないのだ。

TechLION
For Independent Engineer

村井純（慶應義塾大学環境情報学部教授）が語る

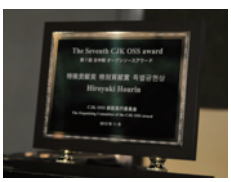
Unix and me
～UNIXとインターネットだぁ！



**TechLION vol.10 第 1 部にお招きしたのは、
日本が誇る「インターネットの父」、村井純氏。
大学時代、UNIX の思想である
“ オープン・プラットフォーム ” に衝撃を受けた経験が、
大学間を結ぶインターネット立ち上げの
原点にあった。
UNIX への愛情、
インターネットの将来像、
米国で刺激を受けた研究者たちの
ユニークな生き方。
縦横無尽のトークに湧いた濃密な 1 時間を、
本稿はダイジェストでお伝えする。
（聞き手：法林浩之、馮富久、まとめ：柏崎吉一）**

馮：法林さん、机の上のプレートは何でしょうか。

法林：先日、「第 11 回 北東アジア OSS 推進フォーラム」にて「日中韓 OSS アワード特別貢献賞」を受賞した際に頂いた楯です。日中韓への貢献というと、せいぜい餃子の王将で食べたとか、麻雀やった程度しか思い当たる節がないですが、嬉しいですね。先日、沖縄で受賞者挨拶をやりました。英語版のプレゼン資料を用意して臨んだのですが、ビデオに雄姿を納めなかったのが心残りです。



馮：じゃあ、今日も資料は英語で。

法林：勘弁して下さい（笑）。馮さんが応援している読売ジャイアンツもアジアチャンピオンになったし、おめでたいこと続きです。さて、今日の TechLION はなんと、Ustream で中継します。前々回の TechLION の第一部では砂原秀樹先生をお招きしましたよね。本日も JUS のすごい大先輩が TechLION に来てくださいました。村井純先生、どうぞ。

村井：どうも、こんばんは。

法林：僕が JUS に参加した頃、村井先生が会長だったようです。ところで先生、もうかなり飲まれましたか。

村井：（手に持ったビール瓶を覗いて）まだ 1cm くらいです。改めて乾杯（**会場：**かんぱーい）。僕は昔、「ミスター・インターネット」って言われたんだけど、法林さんは、「ミスター・UNIX」と言われていたよね。

法林：いや、お恥ずかしい。

村井：そういえば先日、USP 研究所の人に会っ

た時もさ、嬉しかったね。こんな連中がまだいるんだ！って。実は、私もそれで食べていましてね。某バージョンの BSD なんて、カーネルのコードを、目をつぶって vi できましたよ（**会場：**笑）。何行目に何書いているかとか、頭の中にあって。**夜中に、割り込みが入る夢でハッと目が覚めてね。寝ている間にデバッグしちゃう。**

法林：その村井先生が、どうして、インターネットを作ることになったのでしょうか。

村井：色々な資料を持ってきたけど、昔話の前に、まず現在のインターネットを取り巻く状況を聞いてほしい。

最初はインターネットもラジオも相手にされない

村井：来年 1 月にスイスで恒例のダボス会議が開かれます。そこで議論するアジェンダをまとめるための会合が今月、ドバイで開かれた。世界中から集まる CEO や大統領に向けて専門家が、アジェンダを書くんだけど、俺に与えられたお題は、「Future of Internet」だった。インターネットにおける来年の課題は、突き詰めると、ITU（国際電気通信連合）がインターネットを持っていくかどうかなんだよ。**要は、インターネットを国や電話会社が奪うのか、それともみんなのモノにしておけるのか、という勝負だよね。**

法林：インターネットの初期から変わっていない構図のようですね。

村井：ただ誤解してほしくないけど、俺は電話会社と喧嘩するつもりはまったくない。むしろ、インターネットにとって電話はインフラなので、すごく大事です。そもそも、最初の頃のインターネットは、コミュニケーションの手段としては、まだまだ不十分だったし、陽の目を見ると思われていなかった。（スクリーンを指して）これ、何のことを言っているか知っている？

馮：「相手が特定されないメッセージにお金を払うはずがない」とありますね。なんだろう。

村井：（会場から、『ラジオ！』の声）そう、正解。1920 年代、ラジオ放送が始まった頃に、こう言われたんだよね。

馮：広告で稼ぐビジネスモデルがまだなかった頃ですね。

村井：ラジオ放送が市場を確立する前は、商業化の成功を予測した人は少なかった。**広告収入のビジネスモデルも後から気付けばなんていうことないけど、着想に至るまでは人の持つ発想力って、この程度のものなんだよね。**じゃあ、これはどうかな。

法林：「640K はすべての人にとって未来永劫、充分なメモリーだ」。これ言ったの誰だろう（**会場：**笑）

村井：1981 年、ビルゲイツ。

法林：本気で言ったのかな？

村井：セールストークだろうね。インターネットの世界でも、「70Mbps の LTE が出たから、もう光ファイバーのアクセス回線は要らないですよ」なんて言う輩がいる。嘘つけですよ（**会場：**笑）

法林：これは凄いですね。「電子メールよ、さようなら。FAX よ、こんにちは」

村井：それはね、俺が 1986 年ごろ、電子メールネットワーク（JUNET）を作り始めて少し有名人になっていた頃に、ガツンと食らった一言だったね。

「OS のソースを書け」が研究テーマの大学時代

村井：その話をする前に、75 年ごろ、学生時代の話を少しします。コンピュータとの出会いがなければインターネットは作れなかった。でも、俺はもともとコンピュータが嫌いだった。高校時代、コンピュータ使っている連中と言えば、おおむね計算や数学が好き。だいたいメインフレームは偉そうだよ。中心にドーンと鎮座して、周りに人が群がって端末を操作する。**俺はそれを見て本来、立場が逆だろうと思った。**人間が真ん中、その周りに色々な用途のデータやソフトウェアを搭載したコンピュータがあるべき。そうなると自然にコンピュータをつなぐ必要が出てくる。何らかのネットワークが不可欠である。これが卒論の骨子だった。

コンピュータと本格的に関わり始めたのは、慶応の工学部で 3 年生の時だった。ある先生から「OS のソースを書け」という仕事を割り振られた。Ustream で言っちゃっていいのか分からないけれど、要は**リバースエンジニアリング**ですよ（**会場：**笑）

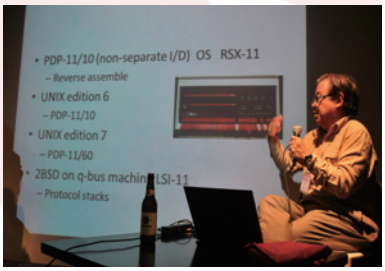
法林：いきなり大学での研究がそれですか（笑）

村井：はじめは、まるで訳がわからない。当時使っていたのが、PDP-11/10 というマシンでした。PDP-11 には、最初 RSX-11 という OS が載っていたんだけど、挙動を見ているうちに、OS というものが次第にわかってきた。16 ビット区切りでデータを書いていく。メモリを示す 16 個のスイッチがあって、ピピピと光る。紙テープがあって、それを読み込ませて「走れ！」「おー、動いた」という感じでね。要するにビットパターンを見て、マニュアル読んでいくと、インストラクションパターンがバレバレなんですよ。でも、あんまりやんないほうがいいですよ、リバースエンジニアリング（**会場：**爆笑）

それでも、OS がわかって C 言語が理解できて、C で書かれた UNIX が出てくると全然違う状況になってきた。

つまり、OS というのは、その頃、あくまで H/W ベンダーがマシンを動かすために付属で作ったもの。それに対して、7th Edition や 4BSD といった UNIX は、ハードウェアとはまったく切り離されてリリースされている。**これは、OS 部分を H/W ベンダーから乗っ取っているわけですよ。**

とはいえ、H/W ベンダーがマシンに合わせてベストチューンした OS に比べれば、UNIX を載せたマシンは軒並み遅いわけ。だけど、ユーザーは自由になれる。問題はそこ、つまり、UNIX はプラットフォームをオー



プンにしたということです。ベンダー側に首根っこを押さえられるのではなく、「ユーザーサイドで何を作りたいのか？」を選択できる自由が手に入る。新しいモノを創造できる標準的な H/W とのインターフェース。OS の中で、**UNIX が一番好きだったのは、オープン・プラットフォームだったから**です。

口説くべき相手を考えるようになった一言

村井：それで、この 7th Edition には、UUCP（UNIX 同士をつなぐ通信プロトコル）が埋め込まれていた。

84 年頃、時代はまだ、電話にコンピューターをつなぐのがイリーガルで、モデムも 500 万円くらいした。こっそりこれ（UUCP を使ったコンピューター接続。やがて JUNET と呼ばれるもの）の研究をしていた。「村井先生がやっているの、何ですか？」との問いに「研究だ」と答えつつね。

悪いことをしようと思ったら……いけね、新しいことをやろうと思ったら大学ですよ（**会場：**笑）。企業には儲かってもらわないと困る。リスクを取れる、特に社会を変えるような挑戦をできるのが大学だと俺は思う。というわけで、**企業の人は新しいことをやるならば大学にお金を払ってください**（**会場：**笑）ただし、情報処理学会で JUNET の説明をしたら、「電子メールは重要だと思うけど、村井、それは、地下でやれ」と。当時いた東工大でも「研究として認めません」の一点張り。なぜ認めてくれないか」と尋ねたら、「東大でやってないから」と言われた。へー、そうなんだと思って、折よく声をかけてくれた東大にすぐ移った（笑）。だからコンピューターを専用線でつなぐ WIDE プロジェクトは東大で始めたんです。

東大へ行ったその頃（1986 年）には、JUNET はすでに 100 組織くらいをつないでいた。北大から九大まで結んで全国制覇、という具合でね。だから関係者の間では既に「俺が電子メールだ」くらいのちょっとした有名人になっていた。

法林：まさに「ミスター電子メール」（笑）

村井：電子メールもあの頃は命がけですよ。ところがその勤め先（東大）のセンター長で後藤英一先生という偉い人がいたんだが、ある日「村井君、ちょっときて」と呼び出され、部屋にあった新しい FAX を見せられながら、**「電子メールよ、さようなら、FAX よこんにちは」**と言われた。FAX なんてとっくにどこにでもあったのに先生には珍しかったんです。

電子メールはキーボードから一生懸命文字を打つ必要があるけど、FAX は手で書けば文字がそのまま送れる。ショックだったね。「俺が認められているのは一部の世界だけだ。この人を満足させていない」。マーケットとして捉えた時、口説くべき相手は誰だろうと考えるようになった。**ソフトウェアって、いいものを書くから認められるのではなくて、人と社会に受け入れられるから認められる。**そのことに気付いたのは、この時だね。

IPv6 の普及に必要なのは 4.2BSD のムーブメント

村井：WIDE をやっていた 80 年代の後半、専用線を使って大学などを結ぶ際に、ハブ（IX）が必要になった。そこで岩波書店に協力してもらったんだよ。

法林：そう言えば、岩波書店に NOC があったと聞いたことがあります。

村井：東大と民間企業をつなぐなんてとんでもないと言われたんだけど、「岩波書店だったら OK」となった。岩波書店ってアカデミックな雰囲気があるから（**会場：**笑）。89 年以降、そこに設備を置かせてもらい、大学間をつないでいった。

法林：理由がようやくわかりました。

村井：ところで去年、バークレイで、「グローバルインターネットはどこから来たか」というテーマの話をした。『それは、ココ（バークレイ）だよ、4.2BSD からだよ』と言ってさ。俺も 4.1BSD あたりの頃から、BSD を作ってた CSRG の連中と一緒に研究をやっていた。この 4.2BSD に DARPA の TCP/IP が実装されて、間違いなくここからインターネットが本格的に広まった。それまで世界の研究機関に広まっていた 4.1BSD を 4.2 にバージョンアップすると TCP/IP が動き出す。**黙ってもインターネットにつながっちゃうわけ。**

この時、バン・ジェイコブソンという仲間が、通信パケットの経路を暴き出す traceroute っていうけしからんツールを発明してこれが大流行。そしたら世界中のあちこちで落ちるネットワークが続出。どうやら 4.2BSD が抱える ICMP（TTL）処理のバグにあったということがわかったんだよね。つまり、当時みんな、その 4.2BSD のソースコー

ドを読んで自分の環境用の実装を作っていたということ。リファレンスコードである4.2BSDの作法を忠実に守っていたことの裏返しだった。

後で、IPv6の仕様を作った時、これを爆発的に普及させるためには4.2BSDのTCP/IPと同じことが起こらなければならないと気付いた。でもそう思った92年頃、CSRGにいた仲間たちは、実質的に残ってなかった。みんなインターネット・ビジネスにかかりきり。「BSDに関わっていた研究者で、まだ大学に居る人」は世界中見回しても、俺だけなんだよ（笑） みんな大金持ちで俺一人が貧乏、そんな感じだった。じゃあどうしようか、といってやり始めたのが、やがてKAMEになってIPv6のリファレンスになっていった。

いまなお続く CSRG時代の交友

村井：その後も、CSRGにいた仲間とは、交流が続いています。ビル・ジョイ（BSDまとめ役・vi,csh作者）、カーク・マキュージック（UFS等開発者）、エリック・オールマン（sendmail作者）などは、そうですね。

馮：みんな名の通っている人ですね。

村井：UUNETを作ったリック・アダムスとも仲はいい。彼もUNIXマニアで、相当悪い奴だけど（笑）

デニス・リッチーはUNIXのファイルの抽象化に貢献したことで知られる。ただ彼が、パリティビットをフラグに使っていたのを見て、俺は、「**日本語には16ビット必要なんだ**」と思わず言った。

法林：漢字表記のためですね。

村井：ベル研でこの話をした後、1年後くらいにまた呼ばれていったらまだその研究をしていた。画面を見ると漢字が出ている。「他の研究はどうしたんだ」とデニスに言ったら「ずっとこれだけやっていた」。すごい研究所だよね。「だから、ベル研って、ノーベル賞受賞者を出すんだ」って思った。

そろそろ、時間だね。これで俺の学生時代から、2002年くらいまでの話をしたけど。

質問者A：入社以来、IPv6をずっとやっています。ヒューマンセントリックと言われますが、どうご覧になりますか。

村井：いろんなことがありましたが、まずアドレス空間は可変長にするのが本来、正しい。どれだけノード増えるか分からないんだから。でも、IPv6決めるとき、アドレスは固定長になった。**理由は、世界中のプログラマが泣くから。**可変長にすると、プログラマがわずかだけどそれだけ作業に手を取られる。ただ、それは一番大事なことだったとおもう。プログラマを見て、意思決定をした。ソフトウェアの開発者がいなければ、インターネットは持たない、それが最後の決め手だった。

質問者B：村井先生、またご登場いただけますか？

村井：いつでも（**会場：**大拍手）

法林・馮：本日は本当に有難うございました。

第2部「ITサファリパーク」もゲストのトークに会場が沸く

大いに盛り上がったTechLION第1部「獅子王たちの夕べ」の熱気そのままに、第2部「ITサファリパーク」がスタート。高野直子さん、片山暁雄さん、増井雄一郎さんによるトークが繰り広げられた。

第2部からは村井先生はコメンテータとして参加。ここでも名言の数々が。

販売ブースも大盛況！

サインをGETする人も。

恒例の、出演者同士の記念写真で、ファイティングポーズ!!
左から）馮さん、片山さん、村井先生、高野さん、増井さん、法林さん。

てっくんからのお知らせ

①TechLIONの次回予告は裏表紙を見てね♪
②地方遠征もあるよ♪
③詳細はWEBサイトをみてね♪（スタッフのブログも頻繁に更新されているよ）

http://techlion.jp/

2013年もエンジニアが集う！

シェルスクリプト大喜利 第七回

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。三か月のご無沙汰、シェルスクリプト大喜利（略して sh 大喜利）のコーナーです。

冬到来、アタクシのギャグにピッタリな寒い季節になりましたよ。……年明けにテックライオンやるんだってねぇ。テックら行っテックラぁｲｵﾝ！……シェル使った大喜利やるんだってね。よし、投稿任シェルです‼

ほーら、寒くなってきたでしょ。まるで大○商会の CM みたいじゃないか。部屋を暖めても無駄無駄ぁ！さて、**今ので何人読者が帰っていったか**な。（あ、ウソウソ！帰らないで～！！）

極寒の話はこのへんにして……、めげずに残ってくれたホントは心暖かな読者の皆さんのために、それで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？

五、Open usp Tukubai(http://uec.usp-lab.com/)も使用 OK！

但し、それなりに見応え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、それじゃ始めましょう。

本番開始

<第一問>

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

前回の「幸運数」に続き、整数数列繋がりで今度は「完全数」というわけですよ。自分自身を除く約数の総和が自分自身と同じ値になるってヤツなんですけどね。

幸運数同様、これまた投稿数が一番多かったんです。もしかして、皆さん整数とか数列好きだったりしますか？それじゃ解答いってみましょう。

◎321516さんの解答

```
1 #! /bin/sh
2 awk 'BEGIN{for(x=1;x<='$1';x++){for(y=1;y<x;y++){p
  rint x,y,x%y}}}' | grep ' 0$' | sm2 1 1 2 2 | awk
  '$1==$2{print $1}'
```

お、Tukubai コマンド使ってますな！いつも投稿ありがとうございます。ひょっとすると、Open usp Tukubai の Github リポジトリにシェルスクリプト版 Tukubai を pull リクエストしてる方かな？？？

さてと。この解答はオーソドックスに解いてるね。最初に AWK で総当たりに剰余を求め、それが 0 だったら約数だから次の grep でフィルタリング。次に sm2 で数ごとに約数の総和を出して、最後にそれが元の数と等しいものだけ表示する……と。いいね、sm2 コマンドの使い道も示してくれてありがとう。しかも今回の投稿の中で最速だったのだ。途中で経由させている Open 版の usp Tukubai コマンドは中身 Python なのにね。やるなぁ。

よし、**一段授与**。前回は二段まで授与してるからこれで三段ね。

◎emasaka さんの解答

```
#!/bin/bash
awklike() { local _0; while read _0; do eval "$1 && $2"; done }

numsum() { local n=0; awklike : '((n += _0))'; echo $n; }

grep_divs_of() { awklike "(($1 % _0 == 0))" 'echo $_0'; }

divs_sum() { seq $(($1 / 2)) | grep_divs_of $1 | numsum; }

seq $1 | awklike '[ $(divs_sum $_0) = $_0 ]' 'echo $_0'
```

添付のコメントによると、ループや条件分岐をなるべく使わないようするため、シェル関数を定義しそこに相当の処理を詰め込んだそうです。

というわけで、Bash を活用した投稿がきたぞ。殆ど Bash の機能で済ませているわりに、確かにループや条件分岐の構文も殆ど無しにしているのがいいね。まぁ、そうすると欲が出るもので seq さえ使わなければ Bash で閉じたスクリプトになっていたのに、なんて思ってしまうが、そうなるとさすがに本格的にループ構文が必要になってしまうからね。

ありがとー、**一段褒与**してこれで三段だ。

◎K さんの解答

```
#!/bin/sh
seq 1 $1 |¥
while read n; do
    seq -f "$n %g" 1 $(awk -v N=$n 'BEGIN{print int(N**(1/2))}')
done |¥
awk '$1%$2==0{print $0, $1/$2}' |¥
awk -v N=$1 '$2==$3{a[$1]+=$2}{a[$1]+=($2+$3)}END{for(i=1;i<=N;i++){if(a[i]==2*i)print i}}'
```

こちらの投稿にも添付のコメントがあって、実は並列化に挑戦していたとのこと。時間切れで残念ながら投稿には至らなかったそうですが、**それ興味深い**です！

さて、肝心の投稿コード。約数の総和を求めるアルゴリズムがアタクシの想像してなかったものなので感心した！1 から int(√n) までの商を求めれば約数の総和が計算できるのか。なるほどね！ただルートの計算に処理時間を要しているようで最速には至らなかったのだが。（そこは競ってない？）

でも面白いから**二段褒与**。これで三段だ。

◎東京 awker さんの解答

```
#! /bin/bash
eval echo -ne {1..$1}¥¥ {1..$1}¥¥¥¥n | awk '$1%$2==0' | awk '$1==$2{print $1,t;t=0} $1>$2{t+=$2}' | awk '$1==$2{print $1}'
```

第一問で最後に紹介する投稿。アタクシねぇ、この投稿には感心したのですよ。なぜなら、検証対象数 X とそれの約数 Y の組み合わせを作るために他の投稿者が皆ループ文（for や while）を使っているのに、使ってないんですよ！

Bash の echo 拡張オプションとブレース展開を使ってうまいことやって、結果ループ文なし。そしてペンネーム通りにあとは AWK の連携プレイ。添付コメントは特に無しで不言実行ってことか！それもカッコいいね。

投稿で唯一ループ文使わなかったところに敬意を表して**三段褒与**！現在四段だ。

◇ ◇ ◇

それでは第二問へ行きます。URL エンコードの話なんですが、スマートな投稿と力技な投稿があって面白い！

<第二問>

与えられた文字列を URL エンコード（RFC3986）するシェルスクリプトを作ってください。勿論使っていいのは Linux UM に載っているコマンドと Tukubai コマンドだけですよ。

URL エンコードというのは、Web ブラウザのアドレス入力欄でよく見かける、パーセント記号 "%" やプラス記号 "+" 等が入ったアレを生成するフィルターのことですね。

URL デコーダーについては、Tukubai コマンドの登場した今、難しいことはありません。cgi-name コマンドというのがこれをやってくれます。もちろん使わなくてもわりと簡単にできそうです。何せプラス記号と %xx のキャラクターコードを元に戻せばいいのですから。ところがエンコーダーはちと厄介。変換対象の記号が多数ありますからね。

果たしてどんな解答が寄せられたのでしょうか。

◎tnazuka さんの解答

```
#! /bin/sh
export LC_ALL=C
awk '
BEGIN{
  for(i=0;i<256;i++)
    chr2asc[sprintf("%c",i)]=sprintf("%%%x",i);
  chr2asc[" "]="+";
  for(i=48;i<57;i++)
    delete chr2asc[sprintf("%c",i)];
  for(i=65;i<91;i++)
    delete chr2asc[sprintf("%c",i)];
  for(i=97;i<123;i++)
    delete chr2asc[sprintf("%c",i)];
  delete chr2asc["-"];
  delete chr2asc["."];
  delete chr2asc["_"];
  delete chr2asc["~"];
  ORS="";
}
{
  for(i=1;i<=length($0);i++){
    l=substr($0,i,1);
    print (l in chr2asc) ? chr2asc[l] : l;
```

```
24   }
25 }'
```

添付のコメントは「いろいろ考えたあげく、正攻法です。捻りがきいてなくてすみません」とのことです。いえいえ、参加してくれただけでありがたいですよ。

というわけで普通に解くとこんな感じですな。**初段授与**。

◎sed 内 ker さんの回答

```
#! /bin/sh
sed 's/ /+/g'                                   |
sed 's/%/%25/g'                                 |
sed 's/"/%22/g'                                 |
sed 's/#/%23/g'                                 |
sed 's/¥$/%24/g'                                |
sed 's/&/%26/g'                                 |
sed "s/'/%27/g"                                 |
sed 's/(/%28/g'                                 |
sed 's/)/%29/g'                                 |
sed 's/¥*/%2a/g'                                |
sed 's/+/%2b/g'                                 |
sed 's/,/%2c/g'                                 |
sed 's!/!%2f!g'                                 |
sed 's/:/%3a/g'                                 |
sed 's/;/%3b/g'                                 |
sed 's/</%3c/g'                                 |
sed 's/=/%3d/g'                                 |
sed 's/>/%3e/g'                                 |
sed 's/?/%3f/g'                                 |
sed 's/¥[/%5b/g'                                |
sed 's/¥[/%5c/g'                                |
sed 's/¥¥/%5c/g'                                |
sed 's/¥]/%5d/g'                                |
sed 's/¥^/%5e/g'                                |
sed 's/`/%60/g'                                 |
sed 's/{/%6b/g'                                 |
sed 's/|/%6c/g'                                 |
sed 's/}/%6d/g'                                 |
sed -n 'l'                                      |
sed 's/¥(¥¥[0-7][0-7][0-7]¥)/¥
¥1¥
/g'                                             |
sed -n './p'                                    |
sed '/^¥¥/y/01234567/ABCDEFGH/'                 |
sed '/^¥¥/s/A/000/g'                            |
sed '/^¥¥/s/B/001/g'                            |
sed '/^¥¥/s/C/010/g'                            |
sed '/^¥¥/s/D/011/g'                            |
sed '/^¥¥/s/E/100/g'                            |
sed '/^¥¥/s/F/101/g'                            |
sed '/^¥¥/s/G/110/g'                            |
sed '/^¥¥/s/H/111/g'                            |
sed '/^¥¥/s/¥¥0¥([01]¥{4¥}¥)/¥¥¥1 /'            |
sed '/^¥¥/s/0000/A/g'                           |
sed '/^¥¥/s/0001/B/g'                           |
sed '/^¥¥/s/0010/C/g'                           |
sed '/^¥¥/s/0011/D/g'                           |
sed '/^¥¥/s/0100/E/g'                           |
sed '/^¥¥/s/0101/F/g'                           |
sed '/^¥¥/s/0110/G/g'                           |
sed '/^¥¥/s/0111/H/g'                           |
sed '/^¥¥/s/1000/I/g'                           |
sed '/^¥¥/s/1001/J/g'                           |
sed '/^¥¥/s/1010/K/g'                           |
sed '/^¥¥/s/1011/L/g'                           |
sed '/^¥¥/s/1100/M/g'                           |
sed '/^¥¥/s/1101/N/g'                           |
sed '/^¥¥/s/1110/O/g'                           |
sed '/^¥¥/s/1111/P/g'                           |
sed '/^¥¥/y/ABCDEFGHIJKLMNOP/0123456789abcdef/' |
sed '/^¥¥/s/¥¥¥([0-9a-f]¥) ¥([0-9a-f]¥)/%¥1¥2/' |
sed ':line;N;$!b line;s/¥n//g'                  |
sed 's/¥$$//'                                   |
sed 's/¥$/%0d%0a/'
```

無駄に長い！！いくらなんでも真面目にと解いてここまで長くはならんでしょう。添付のコメントによれば「メニーコア時代。長いけどきっとこれが有利になります」とのことで、**お、おぅ…**って感じですかねぇ。

ただこれだと 16 進数の英字が大文字になって厳密にはちょっと違うけどまあいいか。**一段授与**、これで三段かな。

◎emasaka さんの解答

```
#!/bin/bash
echo -e $(echo -n "$1" | od -An -tx1 -v -w99999 |
  tr ¥  % | sed s/%20/+/g |
sed 's/%¥(2[de]¥|3[0-9]¥|4[^0]¥|5[0-9af]¥|6[^0]¥|7
  [0-9]¥)/¥¥x¥1/g')
```

今の投稿から一転して、おー！これは短い。添付のコメントは「第 3 のビールに浸ってるので省略」とのこと。はい、ありがとうございます。(‥;)

うーん、アイデアがいいね！厄介なエンコードは一旦全部してしまってから、必要無い文字だけデコードして戻すことで、こんなに短くでるのか。

スバラシイ！！**二段授与**！これで折り返しの五段だ。

◇　◇　◇

さあ、最後の問題いってみよう。

<第三問>

"a" というテキストファイルがあります。この中身を画面に表示する方法（一番簡単なのは恐らく cat a）を、いろいろ教えてください。数で勝負するもよし。珍しい方法を披露して勝負するもよし。

このお題、今日のメインイベントに据えていたんですけど……、あれれ？数で勝負する投稿がありませんでした。ちょっと残念。やっぱり数列問題の方が面白い？？

◎イタローさんの回答

```
・head -n 18446744073709551615 a
・tail -n 18446744073709551615 a
・cut -c 1-18446744073709551615 a
・fold -w 9223372036854775807 a
```

「上下右に切るコマンドに抗って切られないようにしました。でも 3 番目はメモリ不足で力尽きるでしょう」とのこと。

なんだこの数字は？と思って調べたら、最初の 3 つは符号無し 64 ビットの最大値、最後のは同じく符号ありでの最大

値でした。へー、これが head、tail、cut、fold の最大値なんだ。

マメ知識として面白いね。よし、**初代褒与**！

◎猿二号さんの回答

```
・nl a | awk ´{sub(/^[[:blank:]]*[0-9]+[[:blank:]]/," ",$0);print}´
・nl a | sed ´s/^[[:blank:]]*[0-9]¥{1,¥}[[:blank:]]//´
・nl a | uniq | sed ´s/^[[:blank:]]*[0-9]¥{1,¥}[[:blank:]]//´
・nl a | sort | uniq | sed ´s/^[[:blank:]]*[0-9]¥{1,¥}[[:blank:]]//´
```

コメント「愚かにも cat を rm して使えなくなってしまったら、行番号の入っちゃう nl で開き、後で行番号を消せばいいんじゃないかと。ついでに行番号があるなら sort しても uniq しても変わらないよねということで地味に数稼ぎました」とのこと。

うーん確かにちょっと面白い。だが、cat が使えない時の回避策というコンセプトだったら「はじめから後ろの AWK や sed 単独で事足りるぞ！」というツッコミを入れたくなっちゃうぞ。いや、もちろんそういう趣旨のお題じゃないからいいんだけど。

ありがとう！**一段褒与**。これで三段だね。

◎ATM53 さんの解答

```
・write `whoami` < a
・wall a
・shar a | sed -n ´/^X/s/^X//p´
・diff a /dev/null | tail -n +2 | sed ´s/^..//´
・diff3 /dev/null /dev/null a | tail -n +5 | sed ´s/^..//´
```

雑多に思いついたものの寄せ集めたそうです。ペンネームからするとネットワーク屋さんなんでしょうかねぇ。

さて、最初の 2 つは本文の手前にメッセージが付くのでどうかと思うんだけどまあ面白いね。最後の diff* もよく考えてくれたと思う。でも特に目を引いたのは 3 番目だね！これ、見慣れないコマンドだなぁと思ったら、何と **Linux JM には確かに載っているのに最近の CentOS には収録されてない** みたいだ。よくこんなコマンド見つけたもんだ！

珍しいコマンド探してくれたんで**二段褒与**しちゃおう。

◎$C_6H_8O_7$（クエン）さんの解答

```
・join0 key=1 a a
・cjoin0 key=1 a a
・self 0 a
・self 1 NF a
・tateyoko a | tateyoko
・filehame -IHOGE a /dev/null
・juni a | delf 1
・rank a | delf 1
・ycat a
・tcat a
・up3 key=1 a /dev/null
```

$C_6H_8O_7$（クエン）さんの投稿は Open usp Tukubai を使ったものですね。

しかし！ダメなのがいっぱいあるじゃないか。self の 2 番目、それから tateyoko、juni、rank の解答、これらは半角スペースが 2 個以上連続している場合、1 個になっちゃうぞ。でもまだまだ普及してないから難しいよね。懲りずに使ってね。

誤答もあるけど Tukubai 利用に感謝して、**初段褒与**！

◇ ◇ ◇

これにて本日の大喜利はお開き！読者の皆さん、投稿してくれた皆さん、今回もまた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）までに存在する今度は「累乗数」を列挙するプログラムを書いてください。速さを追求した解答、または Tukubai を使う等して（使わなくても可）コードの美しさや面白さを追求した解答を待ってます。

二、本誌 38 ページ、「やわらかマッドサイエンティスツ」の記事にあるリスト 5（同一行内文字ソート）。Perl で書かれてて、大喜利司会のアタクシとしてはシェルスクリプト版が欲しいなと思いました。というわけでこれをシェルスクリプトで書いてください。面白いのがいいなー。

三、for や while を一切使わずに、九九の表を書いてください。

```
01*01=01  01*02=02  01*03=03 …… 01*09=09
02*01=02  02*02=04  02*03=06 …… 02*09=18
                    :
09*01=09  09*02=18  09*03=27 …… 09*09=81
```

といった感じです。（今回の投稿もヒントになるね！）

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 **2月 26 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

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してます。そして、考えてくれた方にも段位を授与します。自分で出題して回答するのも、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！

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 2012 年 12 月 20 日
定価 400 円（本体価格 381 円）