エンジニアであり続けられるかは自分で決められる。

ISSN 2187-9443

USP MAGAZINE

2013 Autumn

for the sophisticated shell scripters

特集 ついにユニケージがAWSに登場！

「AWS Tukubai」を試す

シルネン・ブヤンジャルガルの海外開拓レポ

B.W.カーニハン先生にお会いしてきました!

TechLION再録

田中邦裕

エンジニアと経営と、日本と海外と

浮川初子氏に訊く、技術者哲学

From Editor

『UNIX という考え方』（マイク・ガンカーズ著）によれば、人間が作ることができるのは 3 つの段階からなるシステムしかありません。

その最初の段階 " 第 1 のシステム " は、1 人か、せいぜい数人の小さなグループが作った、無駄がなく俊敏なシステムです。

最小限のコストで高い性能を実現するそのシステムのコンセプトは、人間の創造力を刺激します。
やがて、そのコンセプトに惹かれて集まってきた多くの専門家が、次の段階 " 第 2 のシステム " を作り出すのです。

この " 第 1 のシステム " が誕生してから " 第 2 のシステム " に発展するまでは、ほとんどの場合、孤独な長い道のりを経なければなりません。

最初は誰にも理解されない。だが、あきらめないで根気強く、そのシステムの素晴らしさを説き続けているうちに 1 人、2 人、さらに長く続けていれば 10 人…100 人と、指数的に理解者が増えていきます。

B.W. カーニハン教授によって評価されたことは、ユニケージ開発手法が " 第 1 のシステム " の段階を終え、" 第 2 のシステム " に移行する時期が近いことの証しでもあるのでしょう。

しかし、さらに長い道のりを歩き続けて、" 第 3 のシステム " を作り出せたとき、それまでの努力が " 実った " と言えるのです。

USP MAGAZINE 編集部

Contents

※ 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

ついにユニケージがAWSに登場

「AWS Tukubai」を試す

USP MAGAZINE 編集部

シェルスクリプトによるシステム開発。データベースアプリケーションを一切使わず、テキストファイルだけで業務システムを組み上げる。しかも動作は高速で、開発期間も短い……。もしそんな噂を耳にしたのなら、その正体は usp Tukubai かもしれない。

一部の人に知られるのみで、ほぼベールに包まれた存在であった usp Tukubai が、2013 年 8 月、Amazon Web Services (AWS) に登場し、使いたい時に使いたい分だけ、誰でも気軽に利用できるようになった。

果たして usp Tukubai とはいかなるものであるのか。本特集では、これを提供する「AWS Tukubai」の使い方を紹介しつつ、その実力の程を探っていく。

第1章―3分でわかる「AWS Tukubai」

2013年8月、シェルスクリプトによる本格的システム開発環境「usp Tukubai」が、AWS上にAMI（仮想マシン）という形で登場したという報道がなされた。これを「AWS Tukubai」と呼ぶことにするが、それは一体何物なのだろうか。本章ではまず、AWS Tukubaiとはどういうもので、どんなメリットがあるのかを、3分程度で理解できるよう、簡単に紹介しよう。

Q1.AWS Tukubaiって何？

A1.シンプルで高速な開発環境「uspTukubai」のAWS提供版

これまで一部の企業にのみ供給され、一般ユーザーには「シェルスクリプトベースにも関わらず、動作が高速で、コストパフォーマンスも高いらしい」といった噂が聞こえてくるのみだったusp Tukubai。これをAWS上で**誰でも手軽に使えるようにした**ものがAWS Tukubaiである。

AWS上で供給するメリットは誰でも使えることのみならず、初期費用ゼロかつ**使用した時間分だけ料金を払えばよい**点にある。ゆえに、必要な時だけ使いたいというオンデマンドな用途に最適である。

Q2.昨年出た「Open usp Tukubai」と何が違う？

A2.平均20倍以上の処理速度と200以上のコマンド

AWS Tukuaiで提供されるビジネス版は、Open usp Tukubai（Open版）を大幅に超える200個以上のコマンドを収録。しかも**全てC言語で書かれれており、**Open版に比べ、平均20倍以上の速さ（編集部比）で動く。

例えば、Tukuabiコマンドで定番のself（SQLではSELECTによるカラム抽出に相当）を用いた100万行のカラム抽出では、同じAWSインスタンス上で実行したOpen版と速度比較してみたところ、**約97倍**となった。

コマンドの豊富さも見逃せない。Open版では現在52個のコマンドが提供されているが、AWS Tukubaiではその4倍以上の221個もある。**SQLで設計されたシステムの移植**に便利なtagシリーズコマンド、**Microsoft Excelファイルを読み書き**できるrexcel,wexcelコマンドの提供など、ビジネスシーンへの対応がより強化されている。

selfコマンドの速度比較

100万行のデータから一つの列を抽出する作業を、同じc1.xlargeインスタンスにて実行。

ビジネス版のみ提供される主なコマンド

名称	機能概要
bb	Bashコード美化（beautifier）フィルタ
calsed	文字列Aを文字列Bに変換（軽量sed）
fsed	特定フィールドを対象にしたsed
htable	HTMLの<table>内データ抽出
pad0	指定フィールドを0パディング
tag*	SQLライクなデータ処理コマンド群
*excelx	Microsoft Excelファイルを読み書き

他多数

Q3.AWS Tukubaiの利用料金は？

A3.初期費用0で、1時間0.6(注1)ドルから利用可能

AWS は、仮想マシン (VPS) 等を提供するサービス。VSP は何といってもハードウェア購入の**初期負担無し**に始められるのがメリットだが、AWS はありがたいことに、殆ど(注2)が従量課金制。つまり**「使った分だけ課金。使わなければ 0 円」**。一時的に Tukubai を使いたい、試してみたい、というオンデマンド用途には最適と言えるだろう。具体的な価格については、**表 1**、**表 2** を参照してもらいたい。

表 1．AWS Tukubai 仮想マシン（インスタンス）の主要ラインナップおよび価格

インスタンスタイプ*1	仮想CPU数*2	ECU*3	メモリ	ネットワークパフォーマンス	1時間使用料*4（単位米ドル）		
					usp Tukubai	EC2	合計
t1.mcro	64bit × 1	1	630MiB	非常に低	0.60	0.02(注1)	0.62(注1)
m1.small	64bit × 1	1	1.7GiB	低	0.60	0.06	0.66
m1.medium	64bit × 1	2	3.75GiB	中	0.60	0.12	0.72
m1.large	64bit × 2	4	7.5GiB	中	0.60	0.24	0.84
m1.xlarge	64bit × 4	8	15GiB	高	0.60	0.48	1.08
m2.xlarge	64bit × 2	6.5	17.1GiB	中	0.60	0.41	1.01
m2.2xlarge	64bit × 4	13	34.2GiB	中	0.60	0.82	1.42
m2.4xlarge	64bit × 8	26	68.4GiB	高	0.60	1.64	2.24
m3.xlarge	64bit × 4	13	15GiB	中	0.60	0.50	1.10
m3.2xlarge	64bit × 8	26	30GiB	高	0.60	1.00	1.60
c1.medium	64bit × 2	5	1.7GiB	中	1.20	0.145	1.345
c1.xlarge	64bit × 8	20	7GiB	高	1.20	0.58	1.78
hi1.4xlarge	64bit × 16	35	60.5GiB	10 ギガビット	1.20	3.10	4.30

*1 デフォルト設定のディスク容量はどれも8GiB。別途追加料金により増設可能である（EBS）他、各インスタンスタイプに応じて一定量の揮発性ディスク領域（インスタンスストレージ：電源断時に失われる）を無料で増設することも可能。
*2 Tukubai コマンドは並列処理に向いているため、仮想 CPU 数が多いほど性能が上がり易く、お勧めである。
*3 ECU とは、CPU 性能を相対的に示すための Amazon による評価値（例：c1.medium の CPU 性能は t1.micro の 5 倍）
*4 課金は、電源投入直後、及びその後 1 時間毎に行われる。従って、一瞬でも電源投入すると 1 時間分の課金が行われる。

表 2．AWS のその他主な課金条件 *1

項目	価格（目安）
EBS（不揮発ディスクスペース）使用料	1GB あたり月 0.1米ドル
EBS アクセス (IOPS) 使用料	スタンダードボリュームの場合、 一時間フル稼働させたとして概ね 0.036 米ドル *2
EC2 →インターネット宛データ転送料*3	最初の 1GB までは毎月無料 以降毎月 10TB までは 1GB あたり 0.12 米ドル

*1 詳しくは、Amazon EC2 料金表のページを参照されたい。 http://aws.amazon.com/jp/ec2/pricing/
*2 表の価格は参考値。詳細な条件によって実際の価格は変化する。
*3 インターネット→ EC2 宛のデータ転送は無料である。

注 1　AWS 新規登録から 1 年間は、最小マシン構成の t1.micro を選ぶとそのマシンの課金分 (1 時間 0.2 米ドル) が毎月実質無料になる。詳しくは「AWS 無料利用枠」で検索。

注 2　ただし EBS と呼ばれるディスクスペースは、稼働させていなくても**ディスクスペースを貸りているだけで料金が発生**する。その他、主な課金条件を**表 2** にまとめてあるので参照してもらいたい。

Q4.どうやって使うの？

A4.次章でじっくり解説！

AWS なら、自分でマシンを構築するより遥かに簡単に始められる。だが、一部に英語表記の手続き画面があるなど、AWSを使った事のない人には少々難しいかもしれない。そこでAWS初心者向けの始め方マニュアルを用意した。早速､次ページを開こう!

第2章―AWS Tukubaiをはじめてみる。

昨年のOpen usp Tukubaiに続き、AWS上でビジネス版usp Tukubaiが提供されたことで、より身近になったTukubai。ただ、使い始めるまでの手続きは、AWS初心者には少々難しいかもしれない。そこでAWS Tukubaiを使い始めるまでを手取り足取り紹介しよう。また、説明ビデオ(約8分)も公開されているので、そちらを参考にするものよいだろう。→ **http://www.youtube.com/watch?v=Yxop1Wrjc-M**

1. AWSを始める

1) AWSサインアップのため、ブラウザーから **http://aws.amazon.com/jp/free/** へアクセス。新規サインアップ者は1年間「無料利用枠」を使えるので、参考に読んでおこう。読んだら「サインアップ」開始だ。

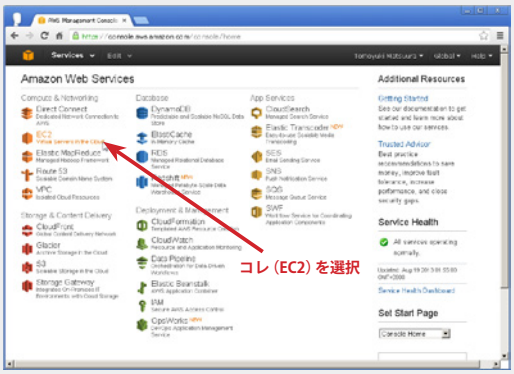
2) 指示に従い、名前や利用料金支払いのためのクレジットカード番号等を入力していく。イタズラサインアップ防止の為、**途中で電話による認証がある**。(こちらの声を音声認識するぞ。スゴい！)そしてサインアップ完了後、下の画面に。そうしたら「AWS Management Consoleの起動」を選択。

2. マーケットプレイスでTukubaiを取得

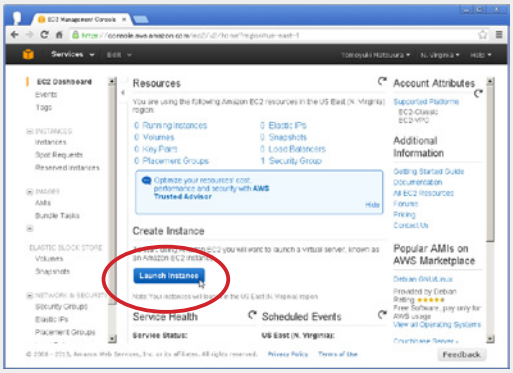
3) ここはAWSにサインインした直後の画面だ。(AWSを以前から利用していて、**サインアップ済の人はここから**始める)

この画面の右上の「アカウント / コンソール」内にある「AWS Management Console」を選ぶ。

4) 一番左の列にある「EC2」を選ぶ。(**Tukubaiは EC2の中で提供されている**のだ)

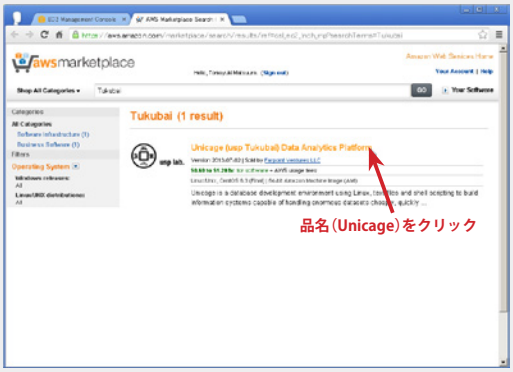


5) **EC2 ダッシュボード画面**に到着。ここから Tukubai 入り仮想マシン (= インスタンス) を取得し、作る。そのため「Launch Instance」をクリック。

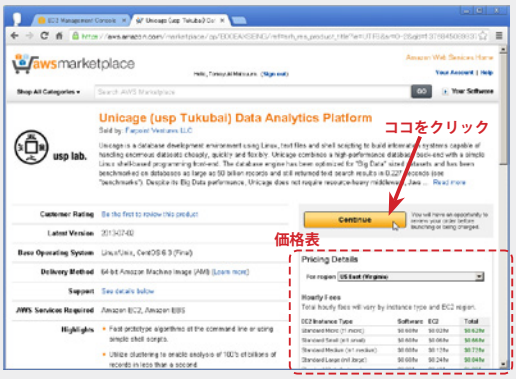


6) Tukubai 入り仮想マシンは **AWS マーケットプレイスで提供されている**。よって、左サイドの「AWS Marketplace」を選択した後、右上に「Tukubai」という検索キーワードを入れて、検索 (Go) する。

7) 見つかった！そこに表示されている品名 “Unicage (usp Tukubai) Data Analytics Platform” をクリック。

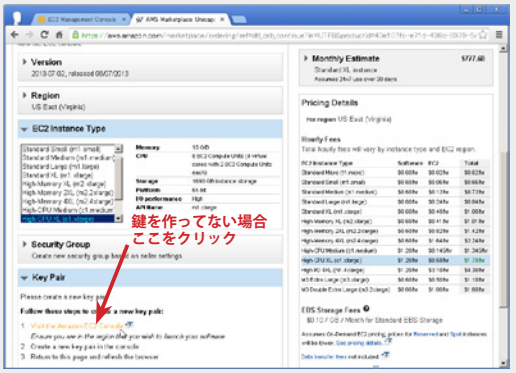


8) 商品詳細画面。左下にマシン構成、右下に価格表（仮想マシンが置かれる地域によって若干価格差あり）がある。確認したら「Continue」をクリック。



3. マシンスペックを決める

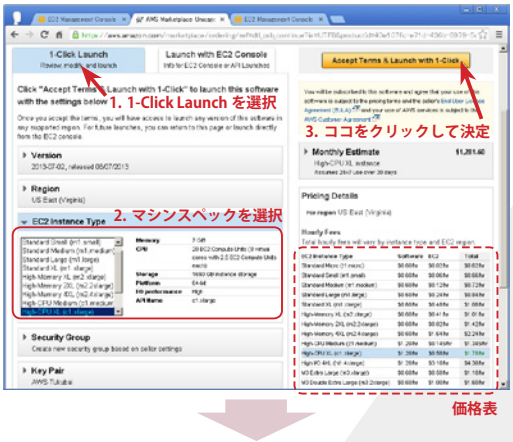
9) この画面で、仮想マシンのスペックを決める。だが AWS 用の公開鍵 / 秘密鍵を持っていない場合は、画面左下にある「Key Pair」項目内の「1.Visit the Amazon EC2 Console」にて、**予め鍵を作らねばならない**。（作ってある人は**手順 12** へ）



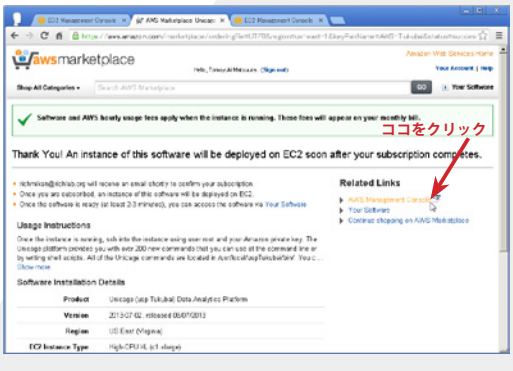
10) こんなふうにして鍵を作る。途中で入力を求められるのは鍵の名前であり、パスフレーズではない。だから分かりやすい名前を付けておこう。

11) 鍵が発行されると秘密鍵がダウンロードされるので適宜保管しておくこと。ダウンロード後、ブラウザーを**手順９の画面に切り替え、再読み込み(F5)させる**。これで AWS 用の鍵の登録は完了だ。

12) さて、仮想マシンのスペックを決める。左上に「1-Click Launch」と「Launch with EC2 Console」の２つの決め方が用意されているのだが、**AWS 初心者は前者が絶対オススメ**。ということで、「1-Click Launch」からマシンを選択する。すると右の価格表が自動的に選択されるので確認しよう。（ここで仮想マシン「Standard Micro(t1.micro)」を選ぶと、利用１年未満なら毎月最初の 750 時間は、EC2 の分($0.02/hr) が無料になるぞ）決めたら右上の「Accept Terms & Launch with 1-Click」をクリック。

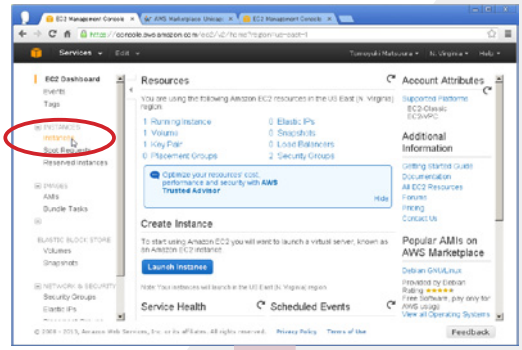


13) この画面に来たら、裏では仮想マシンが生成され、起動が行われている。つまり**課金が始まっている**ということ。さあ、「AWS Management Console」へ戻って使い始めよう。



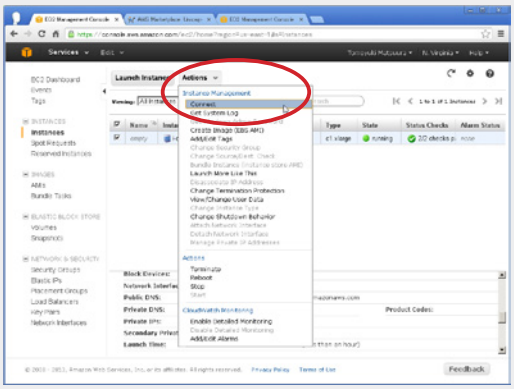
4. Tukubaiマシンを使う

14) 再び EC2 ダッシュボード画面。今度は左サイドメニューの「Instances」を選ぶ。

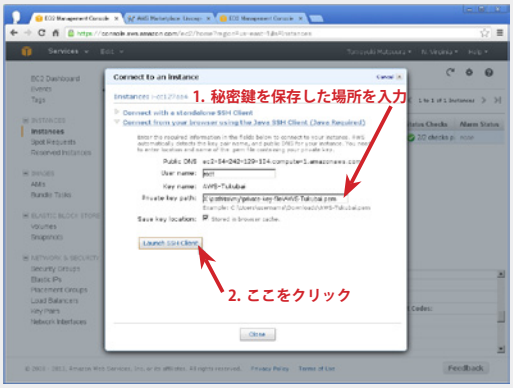


15) するともう動き出して (running) いる。ここでその仮想マシンをクリックすると、やがて画面底部に、**アクセス用のアドレス (Public DNS) 等が表示される**。このアドレスはメモしておこう。

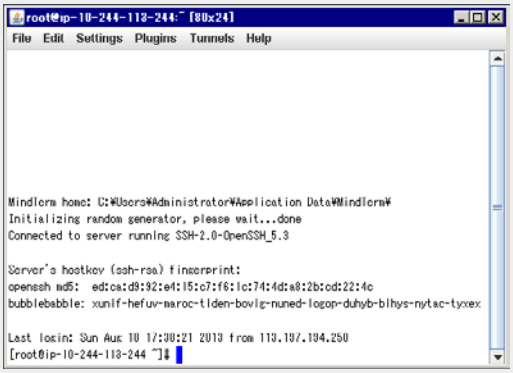
16) ログインしよう。まず AWS が用意しているターミナルによるログイン方法の解説から。これを使う場合は上部メニューの「Action」から「Connect」を選ぶ。



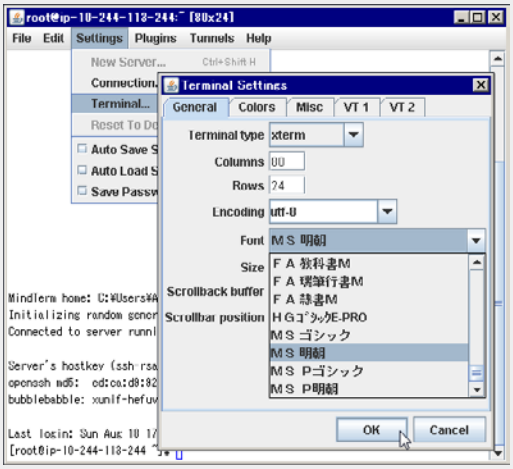
17) 先程ダウンロードした秘密鍵のパスを指定して「Launch SSH Clinet」を押す。するとJavaアプレット版のターミナルが起動する。(**要 Java Runtime Environment**)



18) ターミナルが起動し、ログイン成功。はじめまして！



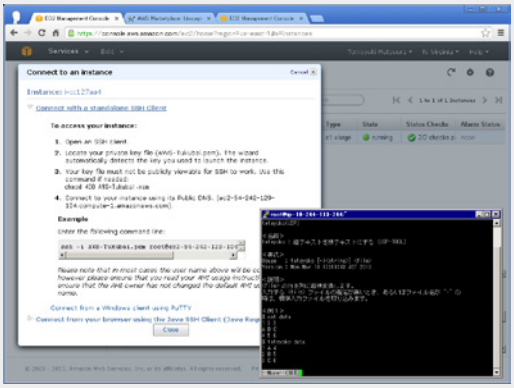
19)「Setting」内の「Terminal」の設定をいじれば、**日本語表示にだって対応**させることができる。



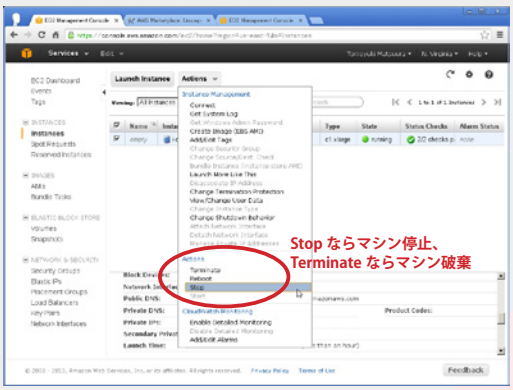
これで、Linuxマシンとして普通に使えるぞ！

他の SSH クライアントからログインする場合

16')手順 16 から……もちろん SSH やいつも自分で使っているターミナルソフトでログインすることもできる。**先程の公開鍵、および先程メモしておいた Public DNS のアドレスに対し、root としてログイン**すればよい。



20) 最後に、仮想マシンの止め方。EC2 ダッシュボード画面にて、対象の仮想マシンを選択 (チェック) している状態で、上部メニューの「Action」の中の「Stop」を選べば止まる。**止めないと 1 時間毎に課金され続けるので注意**。ただし、ディスク (EBS) 使用料は Stop しても課金され続ける。まあ 1GB あたり月 0.1 ドルではあるのだが……。これも止めたい場合は、Stop ではなく Terminate をして、その後 EBS を Delete する。もちろんその場合、ディスクの内容は破棄されるので注意！



AWS Tukubai を使い始める方法はおわかりいただけただろうか？　次章では、様々なデモを紹介しよう。

第3章―実力がよくわかる、Tukubaiデモ5選

「AWS に usp Tukubai が登場した」とはいうものの、そのusp Tukubai（Tukubai）は一体、どんなことができて、どれくらい速いのか？ そこで編集部では、今回独自に組んだベンチマークプや過去に収録した Tukubai レシピ、更にはリリース元の USP 研究所が提供しているデモプログラムや教材を動作させ、Tukubaiの実力の程を検証した。これらのプログラムは公開しているので皆さんも是非試してもらいたい。

準備．デモプログラムのインストール

AWS Tukubai の実力を体験するためにより抜いたデモプログラムを UEC サイトに用意した。まずはこちら（**https://uec.usp-lab.com/201309-uspmag-sample.tgz**）をダウンロードしてもらいたい。（ただし、作成したばかりの AWS Tukubai 仮想マシンにはダウンロードを行うための wget コマンドが入っていないため、先にインストールしておく）

```
$ yum -y install wget
$ wget https://uec.usp-lab.com/201309-uspmag-sample.tgz
```

これを解凍し、生成されたディレクトリの直下にある INSTALL.sh を実行すれば準備完了だが、**一つだけ注意！**

```
$ tar jxf 201309-uspmag-sample.tgz
$ cd TukubaiDemo
$ ./INSTALL.sh
```

Web アプリのデモを含んでいるため、前述の INSTALL.sh は Apache の設定 (httpd.conf) とファイアウォール (iptables) の設定を書き換える。それらを既に弄っている場合は、INSTALL.sh の冒頭に記されている説明に基づいて、手動でインストールを行う。

デモ１．帳票を作成する

まずは小手調べ……、というわけでもないが Tukubai コマンドの基本が短く凝縮された例を紹介したい。

解凍したディレクトリの中にある SALES_REPORT というディレクトリに入ってもらいたい。入ったらそこに、report というファイルがあるのでこれを実行してみよう。

```
$ cd TukubaiDemoディレクトリ/SALES_REPORT
$ ./report
```

これは同じディレクトリにある、商品原価・売価マスタ（PRICE）、売上（SALES）、部門マスタ（CATEGORY）、部門名マスタ（CATEGORY_NAME）の各ファイル（テーブル）に基づいて右のような帳票を出力するものだ。**SQL の世界で言うならビューに相当**するし、その場合は SELECT 文を使えば恐らく同じことができる。

実行を終えたら、report のソースコードを眺めてみてもたらいたい。15 個のコマンドがパイプで繋がれ、工場の製造ラインのごとく逐次実行されるようになっていることがわかる。やるべきことは上から下へと素直に書かれ、各行も人間にとって理解し易い処理単位になっている。このスタイルこそが Tukubai プログラミングの真髄なのである。

また、その**実行に要した時間にも注目**してもらいたい。売上データは約300万行あるが、c1.xlarge インスタンスなら約 3 秒で処理が終わるのである。同じデータの処理をリレーショナルデータベースでやったとしたら、果たして同程度の時間で終えられるだろうか。

リスト１．売上データに基づく帳票出力結果

```
売上レポート

部門 種別       数量     売上     粗利    利益率%
==========================================
001  野菜     327,537 175,521  40,401    23.0
002  果物     255,235 144,118  33,202    23.0
003  魚類     339,523 195,023  41,661    21.4
004  肉類     114,795  72,395  12,906    17.8
005  調味     217,198  81,404  20,373    25.0
006  雑貨     115,552  46,375  10,695    23.1
007  パン      49,472  38,238   6,614    17.3
008  冷凍      72,660  28,241   7,173    25.4
010  飲料     125,012  72,688  17,021    23.4
011  酒類      24,259  12,075   3,165    26.2
==========================================
@@@  ***   1,641,242 866,078 193,210    22.3
```

デモ２．1000 万件データ処理ベンチマーク

次は Tukubai コマンド１つ１つの処理能力を評価するためのベンチマークをしてみよう。このデモは、解凍したディレクトリ直下の BENCHMARK というディレクトリの中に用意されている。

```
$ cd TukubaiDemoディレクトリ/BENCHMARK
```

テストデータを生成

最初に、ターゲットとなるテストデータを生成しよう。まずは次のプログラムを実行してしばし待とう。

```
$ ./make_TESTDATA_ja.sh 10000000 > TESTDATA
```

引数の 0 の数は 7 つ、つまり 1000 万だ。これで TESTDATA というファイルに 1000 万行からなるランダムなデータが生成される。

データは、4 列で構成されており、1 行目から順に、id(受付番号)、pref(都道府県名)、balance(収支)、date(日付) である。ベンチマーク用のデータなので、あまり深い意味は無いことをご了承いただきたい。

基本中の基本「列の抽出」

全データの中から東京都の経費列のみを抜き出してみよう。SQL 的に言えば **SELECT balance FROM TESTDATA WHERE pref = '東京都';** 的な作業だ。

```
$ time cat TESTDATA | selr 2 '東京都' | self 1 > OUTPUT
real    0m1.182s
user    0m0.694s
sys     0m0.490s
```

編集部で試したマシンは c1.xlarge（Xeon E5410 2.33GHz、8 コア）であったが、結果はご覧のとおり 2 秒足らずであった。

抽出するだけでなく、もちろん値の加工もできる。例えば上記の値の合計を求めてみよう。これも SQL 的に言えば **SELECT SUM(balance) FROM TESTDATA WHERE pref = '東京都';** 的な作業だ。

```
$ time cat TESTDATA | selr 2 '東京都' | self 1 | tr -d , | sm2 0 0 1 1 | comma 1 > OUTPUT
real    0m1.188s
user    0m0.724s
sys     0m0.492s
```

CPU コアが 8 つのマシンであるため、tr や sm2 コマンドといった CPU に相応の負荷をもたらすコマンドが 2 個増えても並列実行で吸収できるということであろう。コマンドを実行するのに要した CPU 時間の合計値 (2 行目) は増えているので、仕事量はきちんと増えていることがわかる。

「ソート」の威力が凄まじい

ソートもまたビジネス版 Tukubai の威力が発揮される部分だ。msort という強力なソートコマンドが用意されており、これはソートをオンメモリで行う上に、マルチコア対応。早速試してみよう。

サンプルデータを、第一に 4 列目 (日付) の昇順、第二に 2 列目 (都道府県名) の昇順でソートしてみよう。SQL 的に言えば **SELECT * FROM TESTDATA ORDER BY DATE ASC, PREF ASC;** である。

```
$ time msort key=4@2 TESTDATA > OUTPUT
```

-p8 というオプションを付ければ 8 コアで処理する。

```
$ time msort -p8 key=4@2 TESTDATA > OUTPUT
```

結果を**表 3** にまとめた。1000 万行ソートが 13 秒で終わるというのは驚異的だと思う。ちなみに標準で入っている GNU 版 sort コマンドにも全く同じ処理をさせてみたが、その速度差 93 倍 !! …雲泥の差だった。

C 言語の鬼は伊達じゃない

Tukubai コマンドは、C 言語熟練者の手で徹底的にチューニングされているといい、ご覧のとおり速い。

コマンドのヘルプが man2 コマンドで見られるので、他のコマンドもベンチマークテストしてみるとよいだろう。ヘルプは、環境変数 LANG に ja_JP.UTF-8 を設定しておけば日本語版になる。例えば sm2 というコマンドの日本語 man は次のようにして見られる。

```
$ export LANG=ja_JP.UTF-8
$ man2 sm2
```

表 3．1000 万行ソート対決！

GNU 版 sort	msort	msort(8パラ)
20分16 秒98	22秒75	13秒15

c1.xlarge（Xeon 2.33GHz、論理 CPU 数 8）上での比較

デモ３．ルービックキューブと帳票、宝くじ

次に紹介するデモは、本誌 Vol.5 と 2012 autumn で紹介した、Open usp Tukubai 用レシピ３本だ。AWS Tukubai は、Open 版のものも当然動く。しかも速い！

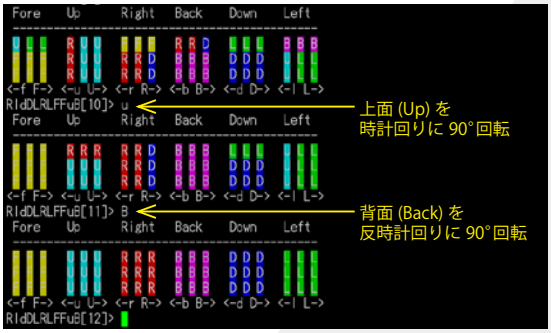
ルービックキューブシミュレーター

１つ目は色鮮やかなルービックキューブシミュレーターの紹介だ。解凍したディレクトリの中の RUBIK_CUBE というディレクトリに入り、rubik_cube というシェルスクリプトを実行してみよう。

```
$ cd TukubaiDemoディレクトリ/RUBIK_CUBE
$ ./rubik_cube
```

すると**画面１**のように、正面、上面、右面、背面、底面、左面の現在の色配置が表示される。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。小文字ならば反対方向に回転する。プロンプトにある英字の羅列は答えを示しており、一手目からその通りにタイプすると色が揃う。(終了は “q”)

このデモは Tukubai が**業務システムのみならず、数理操作にも活用できる**ことを示している。是非ソースコードも見てその様子を実感してもらいたい。



画面１．ルービックキューブシミュレーター

部門トレンド（帳票デモその２）

２つ目は帳票デモだ。本章の最初にデモした帳票よりもさらに複雑である。解凍したディレクトリの中の BUMONTREND というディレクトリに入り、同名のシェルスクリプトを、とにかく実行してもらいたい。この時、ターミナル画面の文字数は縦横共に大きめにしておくのがおすすめである。

```
$ cd TukubaiDemoディレクトリ/BUMONTREND
$ ./BUMONTREND
```

画面に出力されるのは商品部門・地域別の売数推移の帳票だ（**写真１**）。ここまで綺麗に表示できるなら、大掛かりな**帳票エディター等使わずにプレーンテキストで出して印刷するので十分**に思えてこないだろうか。

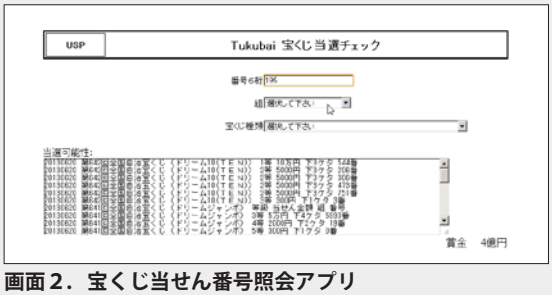


写真１．画面に出力された帳票を紙に印刷

宝くじ番号照会 Web アプリ

３つ目は、ちょっとイジワルなジャンボ宝くじ当選番号照会 Web アプリだ。まずは、次のように打ち込んで、LOTTERY ディレクトリの中の LOTTERY.DB というシェルスクリプトを実行してもらいたい。

画面２．宝くじ当せん番号照会アプリ

```
$ cd TukubaiDemoディレクトリ/LOTTERY/SHELL
$ ./LOTTERY.DB
```

これは宝くじの公式ページ HTML から、当せん番号の抽出を行うものである。終わったら次に、Web ブラウザーから次の URL を開いてもらいたい。

```
http://第2章の手順15で得られたアドレス/TukubaiDemo
/LOTTERY/CGI/LOTTERY.CGI
```

すると**画面２**のように当せん番号照会画面が表示される。ここで、実際のジャンボ宝くじの番号や組番号、抽せん回を入力すればよい。Ajax でリアルタイムに、現在までの入力で当たっている可能性のある金額が表示される。**大抵は、最後の桁まで入力し終えたところでドボン（０円）**になるという、イジワル仕様だ。

デモ 4. 個性的 Tukubai コマンド 59 連発！

AWS Tukubai には 200 種類以上ものコマンドがあるが、ここではページの許す限りそれらを紹介しよう。

まずは、解凍したディレクトリの中の SHORTDEMO ディレクトリに移動しよう。準備はいいかな？

```
$ cd TukubaiDemoディレクトリ/SHORTDEMO
```

コマンド 59 連発！

★ CSV ファイルを読みたい！…sed,AWK でも頑張ればできるが、fromcsv なら超簡単。値として改行やカンマ、引用符（""）が含まれてても大丈夫。サンプルの sample.csv を生で見てから、fromcsv に掛けてみよ。

```
$ cat sample.csv          ←まずは元の中身を確認
$ fromcsv sample.csv      ←そしたら使ってみよう！
```

★ Excel ファイルを読み書きしたい！…確かに Excel ファイルがシェルスクリプトで一気に弄れたらさぞ便利だろう。それなら rexcelx と wexcelx。第 1 引数から順に、シート番号、セル範囲（左上）、セル範囲（右下）、ターゲット Excel ファイルと指定すれば OK。試しに、AWS Tukuabi 価格表を記したサンプル pricelist.xlsx を読んでみよ。

```
$ rexcelx 1 B5 F20 pricelist.xlsx | keta
                  見難いので桁揃えしている↑
```

1 番目のシートの、B5 ～ F20 の範囲にある値を抜き出した。「本当に合ってる？」と思うなら、元の pricelist.xlsx を Excel で開いてみればいい。

★バイナリコード混じりファイルに困っている？…そうそう、HTTP POST で届くデータや破損したテキストファイルでたまにそういうことがあって、表示が乱れて厄介だ。そういう時は utf8nude。これに掛ければ、UTF-8 に準拠しないバイナリデータを除去する。

試しにビープ音を鳴らすコード（0x07）を混ぜたサンプルテキスト beep.txt を utf8nude に掛けてみよ。

```
$ cat beep.txt          ←まず元の中身を確認（音が鳴る）
$ cat beep.txt | utf8nude   ←これは鳴らない！
```

★余計なことしない sed…例えば、テキストファイル NAME_TMPL.txt 内の "##NAME##" の部分をシェル変数 a に代入された文字列に置換したいと思った場合、

```
$ cat NAME_TMPL.txt | sed "s/##NAME##/$a/g"
```

と書くのはアウト！ナゼなら、a に "/" や "&" や "\" が含まれていたら誤動作するから。もちろん予めエスケープしてやればいいんだけど、なんて面倒臭い！そんな時には calsed が超便利。grep に対する fgrep みたいなもので、文字に対して一切特殊な反応をしない。先の例ならこう書けば解決。あぁ便利、しかも速い！

```
$ cat NAME_TMPL.txt | calsed "##NAME##" "$a"
```

★ SQL ライクに列を指定したい…リレーショナル DB から乗り換える時、気になるのは列（カラム）の指定を基本的に列番号で行う点だろう。「列名で指定できないの？」実はできる！ Tukubai には tag で始まるコマンド群（現在 51 個）があり、それらを使えば OK。対象となるデータの 1 行目を列名と見なし、その名前で指定できるのだ。例えば self コマンドに対応するのは tagself コマンド。試しに ID、姓、名の 3 列からなるサンプル名簿 members.txt から姓だけ取り出そう。

```
$ tagself "姓" members.txt
```

★このマシンの CPU は何コア？…並列処理が得意な Tukubai を使いこなすなら知っておきたいところ。でも、どうやって知ればいいのか。知ってたとしても調べ方は OS によってバラバラで面倒。それなら corenum コマンド。実行すれば素直にコア数を返す。

```
$ corenum
```

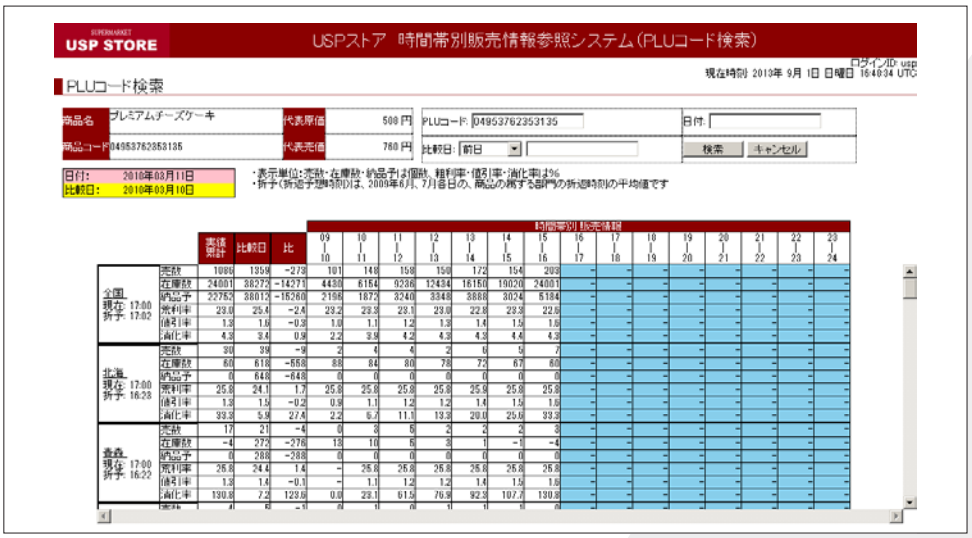
★巨大ファイルの分割…ローテーションをさせ忘れて巨大なログファルイができてしまった！　とりあえず 10000 行毎に分割したいと思った時、さあどうやる？ while 文!? いやいや多分遅くて日が暮れる。そんな時は gyocut コマンド。下記のように書け、動作も速い！

```
$ seq 1 100000 > many.txt      ←10万行のfile作成
$ gyocut -10000 few.%02d many.txt
      ↑few.01～10という10個のファイルに分割される
```

★ファイル末尾の改行コード…これが有ったり無かったりすると煩わしいので無ければ付けたいと思うことがある。そんな時は kaigyo コマンド。たったそれだけのためのコマンド？…そう。でも結構便利よ。

```
$ echo -n hoge          ←これは鬱陶しい！
$ echo hoge | kaigyo    ←kaigyoコマンド付ければ
$ echo -n hoge | kaigyo ←これも鬱陶しくない。
```

デモ 5．サンプル Web 帳票システム「USP ストア」



最後に紹介するデモは、Web ブラウザーから帳票を閲覧する業務システム「USP ストア」である。

このアプリケーションは、シェルスクリプトによるシステム開発方法を学ぶために制作するサンプルアプリケーションであり、**USP 研究所の「教育講座」の K3 クラスのカリキュラム**になっている。（興味のある方は、本誌 2013 summer 号を参照されたい）今回は特別にそのプログラムを収録させたもらった。

デモの使い方

Web ブラウザーから次の URL を開いてもらいたい。

```
http://第2章の手順15で得られたアドレス/TukubaiDemo
/USPSTORE/CGI/JIKANTAI.CGI
```

すると本ページ冒頭のような画面が表示される。

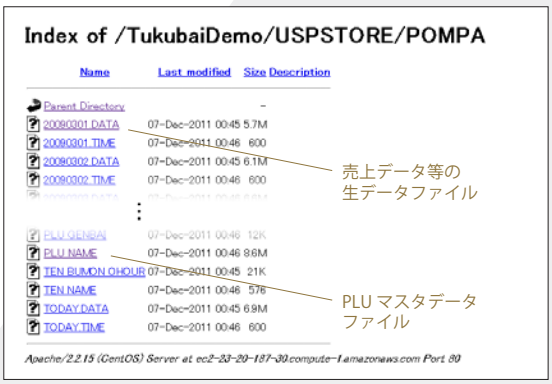
この画面は、PLU（プライスルックアップ）と呼ばれる商品コードを入力し、時間帯別販売状況に関する帳票を閲覧するものである。PLU を指定する他、閲覧した日付の指定、及び販売状況を比較したい過去の日付を設定するようになっている。

従ってこのシステムは、まず各商品の PLU コードを知らなければ検索できないのだが、何も入力しなければ「PLU コード=04953762353135、プレミアムチーズケーキ」が自動的に入力されるようになっている。

また用意されている売上データもサンプルであるため、ここには 2009 年、2010 年の 3 月頃のものしかない。そこで、次の URL へアクセスすることで生データを見られるようにしてある。（**画面３**）

```
http://第2章の手順15で得られたアドレス/TukubaiDemo
/USPSTORE/POMPA/
```

これを見ながら、様々な PLU コードや日付を入力した場合の動作を見てみるとよいだろう。



画面３．各種生データを見られる

個性が強いが万能な Tukubai

このように Tukubai は、データベースアプリを凌ぐデータ管理、帳票作成、数理計算、そして Web アプリケーションまで何でもこなす。

また、ソースコードや格納データを覗けばそれがいかに実現されているかもわかるだろう。個性は強いが、何でもこなす Tukubai に、是非親しみを感じてもらえればと思う。

アプリの開発の目的は「情報の収集と共有化」だ。例えば、日頃の商品販売情報を収集する事で、売れ筋商品や季節変動がわかる。経費削減や業務改善案を共有できれば、社内全体に利点がある。さらに、営業マンが営業ノウハウを共有すれば、営業力の強化になる。「情報」を資産と見て、情報資産を運用する道具としてアプリがある。

ところで Linux には、無料のプログラム言語やアプリ開発環境がある。これらを使わないのはもったいない。そこで今回はアプリ開発の話を書く事にした。

グループウェアと Web 掲示板

2000 年、Linux でサーバーを構築した。丁度その頃、「ナレッジマネージメント」という言葉が流行っていた。情報共有化の事で、それを実現させるのはグループウェアだという。IT を活用して、業務改善や企業業績が向上すれば、IT 先進企業として雑誌に掲載される。そうなれば、私は有名人になれるかもしれない。

そんな欲求も働き、まずはサイボーズのお試し版を Linux サーバーにインストールしてみたが、ほとんど利用されなかった。私のいる本社では、社員全員が同じ部屋にいるため、連絡事項や会議室予約機能などは、ホワイトボードやメールで事足りる。敢えてグループウェアを使う利点がなかった。利点が示せない状態では稟議を挙げても却下されるし、実際、却下された。

最初からグループウェアでは敷居が高い。そこで予算をかけず、かつ、気軽に使ってもらえる物から導入する事にした。まずは Web 掲示板を思いついた。掲示板を使って、本社や営業所にいる社員同士が、経費削減や業務改善を提案し合う事で、情報共有ができる。そして便利さを知ってもらう事で、利用する人が増えるかもしれない。すぐに無料の Web 掲示板の CGI プログラムをダウンロードして、使用することにした。最初は書き込み自由で行なったので、仕事以外の世間話や、面白い内輪ネタを書いてくれる人が何人かいた。それが功を奏したのか、読み書きする人も増えてきた。

しかし、あまりにも度が過ぎる書き込みが出てきたため、業務以外の内容が禁止になり、結局、掲示板は使われなくなってしまった。

全員が節度を保ちながら、利点を感じつつ、利用しやすい環境を整えなくてはならないという、難しい課題にぶつかってしまった。

役員予定表

営業所からの電話で「○○本部長の予定は？」という問い合わせがある。

本社の壁の張り紙に役員予定表があったが、社内向け Web で閲覧できたら便利だと思った。

しかし、そんな物はネットで無料配布していないので、自作アプリの挑戦になった。CGI のサンプル集を見ると Perl が多かったので、Perl の本を片手に見よう見まねで、自作 CGI を作成した。

だが、実際に使いはじめると、予定表を作成している担当者から「使い勝手が悪い」とか「張り紙も作っているので、二度手間」と言った意見が出てきて、長続きしなかった。現在でも張り紙が続いている。

PostgreSQL と PHP に出会う

2001 年、オープンソース・データベースの PostgreSQL と出会う。

データ検索システムが構築できると思った。しかし、PostgreSQL と Web と連動させるためには、仲介するプロ

グラム言語を学ぶ必要があった。

ところが、シェルスクリプトやPerlのサンプル集がなかった上、Javaはサンプル集を見た瞬間、難しいと思い、及び腰になってしまった。

そこで、メーリングリストでお勧めの言語を聞いたところ、「PHPが手っ取り早い」と勧められた。

PHPはオブジェクト指向言語だ。だが実際には、オブジェクトの概念を知らなくても問題はなく、手続き型言語として扱えるため、初心者でも触りやすい言語だ。特に、一度でもC言語を勉強した人なら、覚えやすい。

まずは商品名検索システムから作る事にした。基幹業務システムはAS400で運用している。

以前から商品コードブック代わりに、AS400上やWeb上で商品検索できれば便利だと思っていたが、当時のAS400では高価なミドルウェアが必要であり、予算面で不可能だった。

そこで、AS400の商品マスターをCSVファイルに落とし込み、それをPostgreSQLに格納した。

Web上で手軽に、商品名の一部から商品コードを検索したり、反対に商品コードから商品名の検索が可能になった。コードブックから探す手間を省けるようになり、社内での評判は良かった。

Webデータ検索システム構築

勤務先では、あるデータを顧客に提供していた。顧客から電話で問い合わせがある度に、データー覧表からデータを探して、FAXで送っていた。

1日40～50件の問い合わせがあり、処理に手間がかかっていた上、365日対応を謳っていたため、休日は持ち回りで電話当番をしていた。私も当番に入れられていたので、休みたいのに休日出勤をした事が何度もあった。そこでWeb上でのデータ検索システムを作る案件が浮上した。

当初、外注する予定だったが、私は「折角、PostgreSQLやPHPがあるのにもったいない」と思った。そこで外注依頼する前に、検索システムの原型を作れば良いと考え、突貫工事で作成した。

自社開発すれば経費削減になるので、褒めてくれるかと思いきや、逆に上層部から「勝手な事をするな」と言われてしまった。しかし、既に原型ができてしまったため、私が作った物が採用された。

検索に使うデータは、データベース化されていなかった。そのため、データベースに格納するための入力作業が必要だった。担当部署の同僚が、目を赤くしながらデータ入力や確認作業に追われていたが、データベース化する事で、管理しやすくなった。

検索方法、データの規格などの関係で、担当部署の意見を聞きながら検索システムの改良を繰り返し、完成させた。

ところが、Web検索サービスを開始したものの、利用者は伸び悩んだ。

考えられる原因は、当時パソコンを使っている顧客が少なかったこと、さらに電話だと「○○のデータをくれ」で済むが、検索システムだと、パソコンを起動させ、Webに接続し、データ検索するという手間がかかることだった。

顧客は利点がないと使ってくれない。この当たり前な事に気づき、携帯でも閲覧できるようにしたり、電話でのデータ問い合わせがある度に、ネット検索可能の宣伝チラシをFAXで配布したりした。

その結果、ネット検索する顧客が増え、当番制で行なっていた休日出勤を廃止する事ができた。

カレンダー式掲示板の導入

外出予定や来客予定などの連絡事項は、メールが主流になると、メールの整理を行なっても、見落とす問題が出てきた。他にも、複数人にメールを送る際に、送り漏れも発生する。

そこで、各人がカレンダー式の掲示板に予定を書き込めるようになれば便利だと考えた。

カレンダー式の掲示板を探してみると、PHPで作られたプログラムを発見した。しかし、そのままでは使えないので大幅に改造した。

導入した結果、連絡事項や社員の予定がカレンダーに書き込まれるようになり、各人の予定が把握しやすくなった上、メールの見落としなどが減った。まさに情報の一元管理と共有化の効果だった。

AS400との連動

2006年、基幹業務で使っているAS400を買い換えた。その時、AS400に接続するClient AccessのLinux版がIBMのサイトで無償配布されているのを知った。ODBC接続でAS400とLinuxが連動可能になるため、ODBCを活用すると、Web上でAS400のデータ検索ができる。（図1）

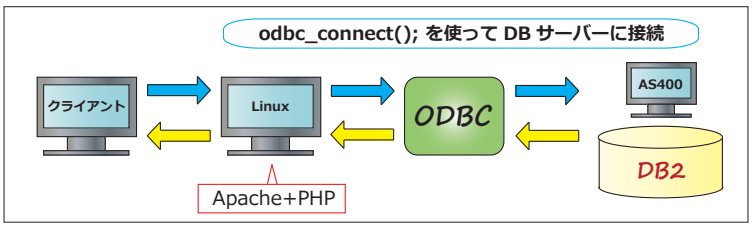


図1　webからのAS400のデータ検索

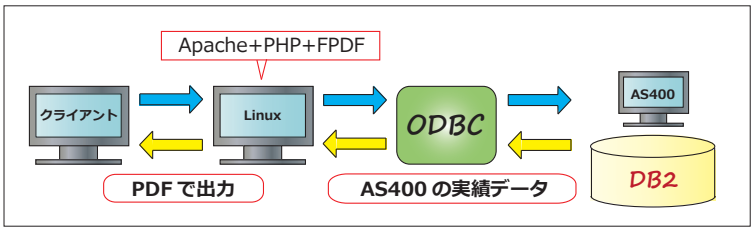


図 2　PDF 帳票生成

これによって高価なミドルウェアの購入の必要性がなくなった。しかし、ClientAccess について日本語の資料がない。英語と格闘になった上、設定の方法は手探りだった。七転八倒の末、連動が可能になり、わざわざ CSV ファイルに落とし込み、PostgreSQL へ格納する手間もなくなった上、更新し忘れによるデータの不整合の問題が解消された。

現時点でのデータ閲覧が可能になる事で正確な販売履歴などが検索・閲覧できる上、伝票検索を行なう事で伝票漏れや記入間違いを発見しやすくなり、在庫管理にも役立った。

AS400 内のデータを補完する形で PostgreSQL を使う事で、顧客情報の書き込みなどを行なえるようになった。

また、PHP の無料 PDF 生成ライブラリの FPDF を使う事で、AS400 の実績データを基に PDF 帳票作成も可能になった。AS400 で帳票を作成した場合、ファイル化するためのツールは高価だったため、紙で印刷する必要があった。だが、PDF 帳票生成が可能になった事で、メールでの実績表の配布が可能になった。（図 2）

Web アプリの注意点

Web アプリの場合、外部の不特定多数の人が接続できるため、セキュリティー対策は必須になる。私自身、最初は Web アプリの脆弱性を知らなかったため、何ら対策を行なっていなかったが、Web アプリの脆弱性のセミナーや本を読んで以来、可能な限り、対策を取る事にしている。

主な対策法を 2 つ挙げてみた。1 つ目はクエリーの無害化だ。無害化対策をしていない掲示板で、書き込み中に、HTML 文の終了を意味する </hrml> タグを記入し、それをサーバーに送ると、掲示板が尻切れトンボになる場合がある。これは、掲示板に書き込んだ </html> を、ブラウザが HTML 文の一部と判断するからだ。掲示板が途中で表示終了になる程度なら笑い話で済むが、場合によってはタグ記号を悪用したクロスサイトスクリプティングという問題が発生するおそれがある。対策方法は、タグで使われる「<」「>」などの文字を、「<」や「>」といった特殊文字に変換する処置を施す事だ。幸い、PHP には変換関数があるので、それの活用をお勧めする。

2 つ目はデータベースと連動させる場合の、SQL インジェクト対策だ。

もし、何の対策も行なっていなければ、SQL 文を含んだクエリーを送り込まれ、データベースの中身を覗き見される危険性がある。「'」の文字を「''」に変換するなどして、SQL インジェクトを防止する必要がある。

活用されなかったアプリの話

アプリ開発をしたものの活用されなかった例はいくつもある。共通した原因は、使う側が利点を感じなかった事だ。上層部の指示で、簡単な Web 営業日報や、Web 目標管理表などを作ったのだが、あまり活用されなかった。

業務命令で、営業マンが日報への書き込みを行なっても、部門長といった閲覧・評価する側が、それに対して何らかのコメントをしない限り、営業マンは「読んでくれていない」と思うし、Web 日報以外にも、紙で営業日報を書いていたら、二度手間になるため、活用する気が失せてしまうという結果になった。つまり、閲覧・評価側の立場の人が、営業マンの報告を読むだけでなく、営業マンと対話するという姿勢でコメントをする必要がある上、既存の手書き日報の廃止という事をしなければ、使われないシステムになってしまう。

シェルスクリプトについて

私の勤務先では基幹業務を AS400 で稼動させている。データを Linux 上の CSV ファイルへ容易に落とし込む仕組みがないため、残念ながらシェルスクリプトで業務アプリを作る機会がない。だが、シェルスクリプトは手軽で便利な道具だ。ファイルそのものをデータベースとして扱い、コマンドで必要なデータの取り出しや集計ができる。Linux の勉強会で、シェルスクリプトを使って家計簿を作っている主婦に会った事がある。

私自身、業務アプリではないが、Linux のログを見たり、ログの集計を行なうのにはシェルスクリプトを使っている。

機会があれば、シェルスクリプトで業務アプリを組んでみたいと思う。

シェルスクリプト大喜利 第十回

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。

秋といえば季語的にも虫の声。時代と共に自然が減ってあまり色々な虫の声が聴けなくなったと嘆くことなかれ。今はインターネットで、実に様々な虫の声が聴ける時代でもあります。「コオロギ 鳴き声」で検索したらコオロギ類の鳴き声をまとめているサイトを見つけたんですが、コオロギと名の付く虫の声が 28 種類もありました。どれも聞き覚えがあって「アレとかソレも、みんなコオロギだったのか」と驚かされました。

さて、これまた驚きの連続、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 とかもなきにしもあらず ??

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

<第一問>

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

すっかり恒例に~~なった~~した第一問の整数数列問題。えぇ〜、今回で 5 回目でございます。

スミス数ってのは、元の数の各桁の数字の和と、素因数分解してできた数字の各桁の数字の和が同じになるものを言うんだそうです。例えば、666。各桁の和は 6+6+6 で 18。一方素因数分解すると 2*3*3*37 なので、2+3+3+3+7 でやはり 18、というわけです。

さて今回はまず、非常にオシい解答からご紹介。

◎おとどさんの解答

```
1 #!/bin/sh
2 for i in $(seq 1 $1); do
3     n=$(factor $i | sed 's/¥([0-9]¥)/¥1 /g')
4     [ $(plus ${n%:*}) -eq $(plus ${n#*:}) ] && ec
  ho $i
5 done
```

Tukubai には、plus という、与えられた引数の総和を返すコマンドがあるんですけど、それを使ってくれてます。factor の返す元数と素因数のアラビア数字を 1 文字ずつ分割し、元数と素因数の区切りである ":" の両端で分けて plus コマンドにそれぞれ流し込んで和が等しいかどうか見てるのです。

いやぁ、plus なんてコマンドがあるのによくぞ気

が付いてくれた！　エラい、……だけどこの解答、不正解なんだなぁ。**スミス数は素数を含めない**のだ。理由は素数はそれ以上素因数分解できないから。なので、factor コマンドの後に素数を除外する 1 行が必要なのだ。Tukubai コマンド使ってくれてるのでオマケしたいところではあるけど、残念ながら**段位ならず**。

◎熱ゃ!お茶!さんの解答

```
1 #!/bin/sh
2 for n in $(seq 1 $1); do f=$(factor $n); echo ${f
  #* } | fgrep ' ' >/dev/null && [ $(($(echo $f | t
  r : '¥n' | sed '1s/¥([0-9]¥)/¥1+/g' | sed '2s/ *¥
  ([0-9]¥)/-¥1/g' | tr -d '¥n' | sed 's/+-/-/'))) =
  '0' ] && echo $n; done
```

投稿者コメント「ワンライナーを目指してみました。for 文は消せませんでしたが」ということです。コードを読んでみると、factor が返す文字列の ":" の前後で行を分け、数字を 1 桁ずつ分解すると共に、上の行には "+" を付け、下の行には "-" を付け、元に戻して全部足し合わせて 0 になるかどうかを見てますね。

うーん、これは面白いやり方だね。ワンライナーらしいごちゃごちゃ感。そしてちゃんと事前に素数を候補から除く処理もあるし、申し分ナシだ。よし、**一段授与**。これで二段だ。

◎K さんの解答

```
1 #!/bin/sh
2 g () { echo $(( $(echo $* | sed -e 's/¥([0-9]¥)/
  ¥1+/g' -e 's/$/0/' ) )); }
3 seq 1 $1 |
4 while read i; do
5   factor $i
6 done |
7 while read j f; do
8   i=${j%:}
9   if [ -n "$(echo $f | grep ' ')" ]; then
10    if [ $(g $i) -eq $(g $f) ]; then
11      echo $i
12    fi
13  fi
14 done
```

続いては常連の K さん。いつもありがとうございます。さて、求め方は factor の返す文字列を ":" で分けて、両側の数字の総和が等しいかどうか見るというのはやはり同様ですね。ただし、総和を求めるのがシェル関数になっているのが特徴的。

これは、熱ゃ!お茶!さんの解答と比較して速かった。前者は処理全体を一つのループで囲んでいるけど、こちらはループが小分けになっていて効率的に動くのかな？　いいねぇ、**一段授与**。これで八段だ。

そして最後に今回の最優秀解答。

◎tnazuka さんの解答

```
1 #!/bin/sh
2 seq 1 $1 | xargs -n 1 factor | tr -d : | awk 'NF>
  2{print $1;print "X",$1;$1="";print "Y" $0}' | se
  d '/^[XY]/s/¥([0-9]¥)/¥1 /g' | awk 'NR%3==1{print
  ;next} {for(i=2;i<=NF;i++){t+=$i};print t;t=0}' |
  awk 'NR%3==1{n=$1;next} NR%3==2{x=$1;next} {if($1
  ==x){print n}}'
```

これは素晴らしい!!　for、while 等のループ文ナシに見事にワンライナー。しかも他を圧倒して速いです！　コメントによると「元の数値を、各桁に分解されるものとは別にストリーム内に保持することで、シェル変数とループ文から解放されました」とのことです。

いやぁ、それにしてもこれは速い！　最初の factor の実行のところも xargs でやってしまってるし、そうやって**ループ文を無くすとこんなにも速くなるのか!**　これは**三段授与**。それくらいの衝撃はあるね。これで六段！

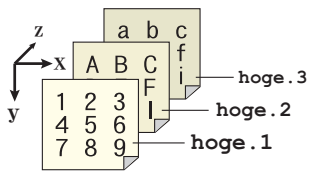
◇　◇　◇

さて、つづいて第二問！　問題文がちと長いですが。

<第二問>

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。



3D プリンターも珍しくなくなって、時代は 3D ！　UNIX コマンドも 3D 対応しなきゃ、というわけでこのお題です。要素は 3^3 個だし回転方向は x-z 平面だけ、と限定的なお題にしたんですが、それらを一般化した解答まで届いちゃったりして、恐れ入りました。

◎tomi さんの解答

```
1 #!/bin/bash
2 col='¥7 ¥4 ¥1  ¥8 ¥5 ¥2   ¥9 ¥6 ¥3' #  90° 用
3 #col='¥9 ¥8 ¥7  ¥6 ¥5 ¥4   ¥3 ¥2 ¥1' # 180° 用
4 #col='¥3 ¥6 ¥9  ¥2 ¥5 ¥8   ¥1 ¥4 ¥7' # 270° 用
5 paste -d' ' hoge.* | sed 's/¥([^ ]*¥) *¥([^ ]*¥)
  *¥([^ ]*¥) *¥([^ ]*¥) *¥([^ ]*¥) *¥([^ ]*¥) *¥([
  ^ ]*¥) *¥([^ ]*¥) *¥([^ ]*¥)/"$col"/' | tee >(
  sed 's/ .*//' > newhoge.1) | tee >(sed 's/.* ¥
  (.*¥)  .*/¥1/' > newhoge.2) | sed 's/.*  //' >
  newhoge.3
```

なんと AWK 版、sed 版、Tukubai 版の 3 つを送ってくれました。どれも 3 ファイルを横 (y 方向) に連結したうえで列を入れ替えるということをやっ

ているわけですが、列を入れ替える操作は AWK や Tukubai(self コマンド) だと何の面白味も無くあっさりできてしまうのであえて sed 版を採用しました。実行するとカレントディレクトリーにある hoge.{1,2,3} を元に newhoge.{1,2,3} に回転後の値を書きだします。

うーん、アタクシの気まぐれなのかもしれないけど、列の並べ替え後、新しい 3 つのファイルになる区切り位置に 2 連続 ,3 連続の半角スペースを入れてるあたり、この sed 版はいいね。AWK のように列番号をダイレクトに指定できない sed でも、これでウマいこと左、中、右で三分割してる。よし、**初段授与**。

◎K さんの解答

```
#!/bin/sh
for z in 1 2 3; do
  awk '{for(x=1; x<=3; x++) print x, NR, "$z", $x}'
$1.$z
done |
case $3 in
 "xz90" |"zx270") sort -k 1n,1  -k 2n,2  -k 3nr,3 ;;
 "xz180"|"zx180") sort -k 3nr,3 -k 2n,2  -k 1nr,1 ;;
 "xz270"|"zx90" ) sort -k 1nr,1 -k 2n,2  -k 3n,3  ;;
 "yz90" |"zy270") sort -k 2n,2  -k 3nr,3 -k 1n,1  ;;
 "yz180"|"zy180") sort -k 3nr,3 -k 2nr,2 -k 1n,1  ;;
 "yz270"|"zy90" ) sort -k 2nr,2 -k 3n,3  -k 1n,1  ;;
 "xy90" |"yx270") sort -k 3n,3  -k 1n,1  -k 2nr,2 ;;
 "xy180"|"yx180") sort -k 3n,3  -k 2nr,2 -k 1nr,1 ;;
 "xy270"|"yx90" ) sort -k 3n,3  -k 1nr,1 -k 2n,2  ;;
esac |
awk 'BEGIN{x=1; y=1; z=1; out="'$2'."z}
     {if (x != 3) {
        printf("%s ", $4) > out
        x++
      } else {
        printf("%s¥n", $4) > out
        x=1
        if (y != 3) {
          y++
        } else {
          y=1
          z++
          out="'$2'."z
        }
      }
     }'
```

来ました、一般化。これは回転軸を一般化してくれてます。引数 1,3 で入出力ファイル名 (拡張子無) を、引数 2 で回転方向と量を指定するようです。コメントによれば、更に X,Y,Z のサイズも一般化したかったそうですが、さすがにそれは難しかったそうです。

いやぁ、なかなかやるね！　回転を実現するやり方も、3^3 個の要素を全て縦に並べ、回転の都合に合わせてソートしてて面白い。よし、**一段授与**。おっと九段だ！

そして、一般化した解答がもう一つ来ました。

◎$C_6H_8O_7$（クエン）さんの解答

```
#!/bin/sh
f=${1%.*}
n=$(ls $f.* | wc -l | tr -Cd 0-9)
files=$(yes $f. | head -n $n | awk 'BEGIN{ORS=""}{print $0 NR}')
case $2 in
  90)  a=-1; b=-$n; c=$((n*n+1)) ;;
  180) a=$n; b=-1 ; c=0          ;;
  270) a=1 ; b=$n ; c=0          ;;
  *)   exit 1                    ;;
esac
fields=$(awk -v n=$n -v a=$a -v b=$b -v c=$c '
         BEGIN{
             for (x=n; x>0; x--) {
                 for (y=0; y<n; y++) {
                     printf("$%d,", a*x+b*y+c);
                 }
             }
         }')
paste $files | awk '{print "${fields%,}"}' > $f.0
for file in $files; do
    fields=$(awk -v x=${file##*.} -v n=$n '
             BEGIN{
                 for (y=1; y<=n; y++) {
                     printf("$%d,", n*(x-1)+y);
                 }
             }')
    awk '{print "${fields%,}"}' $f.0 > $file
done
rm $f.0
```

こちらは K さんが難しいと言っていた X,Y,Z のサイズを一般化してます。ただし、回転方向は固定なようで、第二引数で "90","180","270" を指定するみたいです。また、こちらは第一引数で指定したファイルに上書きするので 90 で 4 回実行すれば元に戻ります。

これまたいいねぇ！どうやってサイズを一般化してるのかと思えば、AWK で AWK のソースコードを作ってるのか。よし、**一段授与**！　これで三段だ。

この 2 人が協力したら完全な一般化バージョンも完成するかもしれないな。いずれにせよ、これで UNIX コマンドの 3D 化も安泰だ。（ホント？）

◇　◇　◇

さて最後、変り種の三問目ですよ。

<第三問>
rev コマンドって知ってますか？各行の文字位置を左右逆にするものなんですけど……。これの使い道を教えてください。

これ、皆さん使ったことありますか？　アタクシ、何に使ったらよいのかさっぱりわからなかったのでお題にしてみたのですが、お蔭様で色々来ました。

◎イタローさんの解答

```
sl | rev
```

コメント「rev を使えば忌々しい汽車を破壊できます」……って、おいおい！ **こっちの画面まで破壊されるじゃないか!!** それだったら、"sl | head -c 1" とかで瞬殺する方がいいような……。いや〜けしからんなー。これは久しぶりに**一段強制返還**だ！　はい、初段ね。

◎tomi さんの解答

```
1 #!/bin/bash
2 expand $1 | ycat -O - <(sed 's/.*//g' $1) | rev
```

こちらは、引数で与えられたファイルの文字の並びを左右入れ替えるものだそうです。単なる rev と違って、全行がちゃんと右揃えになります。ちなみに第二問の Tukubai 版の応用でできてしまったそうです。

なるほど Tukubai の ycat を使うとこんなに簡単にできてしまうのか。ファイルの文字位置が反転して気持ちーなー。しかも、この後更に rev に通せば元のテキストを右揃えしたことになるじゃないか！　よし、**一段授与**。これで二段だ。

◎ゆきとさんの解答

```
1 echo "山本山" | awk '{print;print}' | sed '2s/¥(.
  *¥)/echo "¥1" | rev/e' | uniq | wc -l | awk '$1>1
  {printf "non-"}{print "palindrome"}'
```

これは回文 (palindrome) か否かを判定するワンライナーだそうです。但し sed は GNU 版専用とのこと。

はい、回文判定への応用は、アタクシも来ると思ってたんだよね。AWK の対応次第だけどマルチバイトにも対応してるし、いいね。**一段授与**。これで三段だ。

◎猿二号さんの解答

```
1 cat <<LIST | rev | sort -k 2,2 | rev
2 three,3 This_is_the_third.
3 one,1   This_is_the_first.
4 two,2   This_is_the_second.
5 LIST
```

コメント「列の文字列を右から左にソートしたい場合に rev は重宝しますよ。わりと有名では？」ということで、恥ずかしながら「なるほど！」と思いました。

この解答は今回一番実用的なものかのしれないな。もしかしたら rev コマンドの作者もこういう用途を想定してたのかもしれない。まぁ、この解答の例文からいくと数字が複数桁になったらダメになっちゃうけどね。でも教えてくれてありがとう、**一段授与**。これで四段。

◇　◇　◇

といったろ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、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「ズッカーマン数」を列挙するシェルスクリプトを書いてください。

二、今回の rev コマンドの解答を見ていて、このコマンドはテキストファイルの全行を右揃えするのに応用できることを気付かされました。そこで問題。与えられたテキストファイルをセンタリングするシェルスクリプトを書いてください。

> センタリング（中央寄せ）とは、
> こうやって
> 各行内の中央に文字列を寄せる編集操作ですよ

三、このシェルスクリプト大喜利で過去に、head にムチャクチャ巨大な数の行数指定を与えたり、あるいは mkdir でムチャクチャ深いディレクトリを掘るなどした珍解答がありました。
そこで、そんなふうにして何かムチャクチャ巨大な事やらせるシェルスクリプトと、結果何が起こるかを教えてください。（バージョン番号や、GNU 版か BSD 版か等、再現に必要な環境も教えてください）でも、**致命的な損害を与える解答はダメよ!**

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 **11 月 25 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！

USP MAGAZINE 2013 autumn Vol.10
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 9 月 20 日
定価 500 円（本体価格 476 円）