使ってみませんか？ー便利なソフト利用法ー

Perl 入門
ーものぐさ太郎の UNIX 環境ー

日本原子力研究所那珂研究所
清 水 勝 宏

1．はじめに

最近ではベクトル計算機，パラレル計算機，可視化計算機，PC クラスター，ワークステーション，パソコンとその種類は増え，コードの目的／特性に合わせて計算機を使いこなす必要があります．１ヶ月ぶりにある計算機を使おうとして，さてどのディレクトリで作業を行っていたかさっぱり思い出せないという経験はないでしょうか？作業領域へ移るため，何階層も cd を使ってディレクトリを移動することが，面倒と思われたことはないでしょうか？ファイル編集で名前の入力をよく間違えることに苛立ちを覚えられたことはないでしょうか？

このような物忘れ，ものぐさ太郎の UNIX 環境とは一体どんなものでしょう．

久しぶりに，この計算機を使うので，pcd －10 とキー入力．最後にアクセスしたディレクトリ10個の絶対パス名がリスト出力されるので，番号を選択してそのディレクトリに移動．直ぐに作業に入れます．コード開発では，ソースの編集や計算実行のため，ディレクトリを行ったり来たりすることになりますが，移動には行き先のディレクトリ名を入力するだけです（どこにいても）．それも２，３文字で十分です．特定できた時には直ぐに移動し，該当するものが複数ある場合，候補が画面に出力され数字で選択します．ファイル編集であれば，pedの後，うろ覚えのファイル名で編集が行えます．ped，pcd のコマンドが実行されると，その履歴情報が保存されるので，そこからファイル，ディレクトリの選択も可能です．ついさっき修正したものなら，どのディレクトリにいても，ped －1 でよいのです．

こうしたツール開発に，これまで C shell を使ってきましたが，インターネットの CGI（Computer Graphic Interface）で有名な Perl が学びやすく，使いやすいので，スクリプト作成の具体的な方法を解説し，MT-UNIX 環境をここに紹介します．

自分でスクリプトを作るのは面倒，とりあえず使ってみたいという方は，学会ホームページから tar ファイルを取得して，４章の手順に従って設定を行って下さい．

2．スクリプト作成

それでは，スクリプトの書き方について説明します．ここでは，正確なファイル名を入力しなくても，簡単に目的とするファイルが編集できるスクリプト（pedt）を作成します．ファイル名 plinpt.f を編集する場合，コマンド pedt の実行は以下のように行われます．

```
% pedt pli
current directory : /home/pjsonic/src/soldor
    1: plinit.f  2: plinit0.f  3: plinpt.f
    4: plinpt_sc.f
>> enter number (2) ==> 3
target file : plinpt.f
```

エディターが起動し，ファイル編集が可能となる．

主プログラムに相当する exam.pl が21行，サブルーチン tblst.pl が25行ですので，コーヒーでも飲みながら，１時間程度おつきあい下さい．Perl のスクリプトを作成する前に，以下の規約は覚えて下さい．

- スクリプトの開始行は，必ず#!の後に Perl のフルパス名を指定します．which perl で調べられます．
- ２行目以降，文はセミコロン「;」で終わります．
- Perl が扱える変数には，スカラー変数（$で始める），配列変数（@で），連想配列（%で）があります．
- C shell では，変数に値を代入するときには$をつけないで，「set no＝5」のように書き，それを使用（出力）するときに「echo "number＝$no"」と表記しますが，Perl では，値を定義するときも「$no＝5」と書きます．
- C 言語と同じように配列変数の添え字は，０から始まります．@var の最初の要素に数値を代入するときは，「$var[0]＝3;」とします．
- 端末に結果を表示するときには，print 関数（C shell では，echo）を用い，ダブルクォーテーション「"」で囲み，改行はバックスラッシュ\n とします．これ以降，例を表示するときには，「¥n」と表示します．表示の細かなフォーマットの指定には，printf 関数を用います．その指定方法は C 言語と同じです．

まず，ソースプログラムを開発しているディレクトリに移動しましょう．そして，以下に示すスクリプト exam.pl を作成します．

J. Plasma Fusion Res. Vol.81, No.6 (2005) 458-463

exam.pl [1]：テキストファイル一覧

```
01  #!/usr/bin/perl
02  $pwd = `pwd`;
03  print "current directory : $pwd";
04  @tbls = `ls -a`;
05  @tbtx = ();
06  foreach $fnam (@tbls){
07      chomp($fnam);
08      push(@tbtx,$fnam) if( -T $fnam );
09  }
10  print "@tbtx¥n";
```

説明の都合上，文番号を表記していますが不要です．

01. 開始行は，必ず＃！の後にPerlのフルパス名を指定．
02. UNIXのカレントディレクトリを出力するpwdのコマンドが実行され，それが，perlの変数＄pwdにセット(代入)されます．UNIXのコマンドは，バッククォート「`」で囲みます．
03. カレントディレクトリを表示します．変数＄pwdには改行が含まれますので，改行¥nは不要です．
04. lsコマンドの内容が，配列@tblsにセットされます．
05. テキストファイル名を格納する配列変数@tbtxを初期化します．
06. foreachは，（ ）で囲まれた配列@tblsの要素を一つずつ取り出して，スカラー変数＄fnamとして，｛ ｝でくくられたブロック内を反復処理します．このループ制御は，配列のインデックスを用いて，for($i=0; $i<=$#tbls;i++){$fnam=$tbls[$i]; と同じ．
07. chomp関数を用いて，文字列＄fnamの最後の改行文字を取り除きます．
08. push関数は，配列の最後にスカラー変数を付け加えます．if文は，fortranの論理if文と異なり，論理式は実行文の後に書きます．if文の括弧内（-T $fnam）は，ファイルテスト演算子と呼ばれるもので，テキストファイルなら真です．
09. foreach文の反復処理の最後です．
10. 配列変数@tbtxを表示します．改行文字は取り除いていますので，最後に改行¥nが必要です．

ファイルexam.plを保存して，実行してみましょう．

```
% chmod 755 exam.pl
% exam.pl
```

ここで，%は，端末からのプロンプト（コマンド入力待ち状態を示す）です．1回できちんとlsのコマンドの出力が出た人は相当注意深い人です．多分，以下のようなメッセイジが出たはずです．

```
syntax error at exam.pl line 3, near "print"
Execution of exam.pl aborted due to
                    compilation errors.
```

多くの場合，文末に「;」をつけ忘れたことによるものです．

実行すると，たくさんファイル名が横方向に表示され，目的とするファイルを見つけにくいと思われたはずです．それでは，この点を改良します．配列変数を番号と一緒にリスト出力し，番号で選択する事は汎用性があるので，ライブラリー（ファイル名tblst.pl）として作成します．

tblst.pl：配列変数を番号と一緒にリスト出力し，選択

```
01  #!/usr/bin/perl
02  sub tblst{
03  local (@tbl) = @_;
04  $ii = 0;
05  $line = "";
06  foreach $fnam (@tbl){
07      $ii++;
08      $cnam = substr($fnam,0,12);
09      $line = $line . (次の行を続ける)
              sprintf(" %2d: %-12s",$ii,$cnam);
10      if( length($line) > 80 ){
11          print "$line¥n";
12          $line = ""; }
13  }
14  print "$line¥n" if( length($line) > 0 );
15  exit if( $ii <= 0 );

16  print ">> enter number (2) ==> ";
17  $no = <STDIN>;
18  chomp($no);
19  unless( $no > 0 && $no < $ii+1 ){
20      print " Wrong number! ¥n";
21      exit; }
22  $file = $tbl[$no-1];

23  $file;
24  }
25  1;
```

02. サブルーチン名（tblst）の定義を行います．｛ ｝のブロックの範囲（24行目迄）が実行されます．
03. サブルーチンに渡された変数は，特殊配列変数@_に格納されます．これをローカル変数@tblとすることを宣言します．これ以降，このサブルーチン内では@tblとして変数を参照します．
04. 配列の要素を指定する番号を0に設定します．
05. 配列のリストを書き込む文字列を初期化します．
06. 配列変数@tblの要素に対して，反復処理します．
07. 添え字＄iiに1加えます．$ii = $ii +1としても同じ．
08. 縦方向にファイル名を揃えるため，名前は最大12文字迄出力します．文字列演算子substrを用いて，＄fnamの最初から，12文字を＄cnamとします．
09. 文字列＄lineに添え字とファイル名を新たに追加します．文字列の結合は，ピリオド「.」で行います．C言語のsprintf関数と同じように，数値，文字列の出力のフォーマット指定ができます．
10. 文字列＄lineの長さが80字を超えると｛ ｝で囲まれた文を実行します．

15. 配列に要素がなければ，プログラムを終了します．
16. メッセージを表示してユーザのキー入力を促します．
17. 入力の読み込みには，<STDIN>とします．
18. chomp 関数で入力の改行文字を削除．
19. 不適切な入力の場合，メッセージを出力して終了．
22. ファイル名$fileを指定された番号でセット．@tblの添え字は0から始まることに注意．
23. 決めた値$fileを呼び出し側に戻すときには，サブルーチン最後の文とします．
25. ユーザが定義したスクリプトをライブラリーとして用いるときには，この「おまじない」を忘れないこと．

このライブラリーの呼び出しは，次のように行います．

exam.pl [2]：ファイルの選択

```
01  #!/usr/bin/perl
      exam.pl[1]  9行目まで省略
10  ##print "@tbtx¥n";
11  require "tblst.pl";
12  $file = &tblst(@tbtx);
13  print "taget file : $file¥n";
14  system("emacs $file");
```

10. 配列@tbtxを表示するコマンドをコメントアウト．最初の文字が#で始まる行はコメントとなります．
11. require 関数を用いて，ライブラリーをインクルードします．
12. サブルーチンで決まったファイル名を$fileとします．ライブラリーの呼び出しには，&tblstとします．
13. 確認のため，編集するファイル名を画面に出力．
14. C言語のsystemと同じように，「"」ではさまれたUNIXのコマンドが実行されます．ファイル編集に用いているエディターがViの場合，system("vi $file").

tblst.plを実行モジュールにして，実行(exam.pl)しましょう．目的とするファイル編集ができたでしょうか？

さて，いよいよ大詰めです．正確なファイル名を覚えていないとしても，名前の先頭の2～3文字は覚えています．それでは，ファイルの候補を絞る機能を追加します．

exam.pl [3]：最終的なファイル選択のスクリプト

```
01  #!/usr/bin/perl

a1  if( $ARGV[0] ){
a2      $head = $ARGV[0]; }

02  $pwd = `pwd`;
03  print "current directory : $pwd";
    @tbls = `ls -a`;
    @tbtx = ();
    foreach $fnam (@tbls){
        chomp($fnam);
a3      if( $head ){
a4          next if( $fnam !~ / ^ $head/ ); }
        push(@tbtx,$fnam) if( -T $fnam );
    }
10  ##print "@tbtx¥n";

a5  if( $#tbtx < 0 ){
a6      print "No found file with [$head]¥n";
a7      exit; }

11  require "tblst.pl";
12  $file = &tblst(@tbtx);
13  print "taget file : $file¥n";
14  system("emacs $file");
```

追加した行は，a1～a7の7行です．

a1. コマンドラインからの引数は@ARGVの配列に格納されます．最初の引数が定義されていたら，
a2. ファイル名の先頭の変数として，$headを定義．
a4. ファイル名$fnamの先頭が$headにマッチしなければ(!~で指定)，次のループを繰り返します（nextはCのcontinueに相当）．マッチした時の実行には「=~」とします．先頭を調べるため，「^」で指定しています．こうした正規表現はC shellと同じです．
a5. ファイル名を絞ったことで，候補のファイルがない場合にはメッセージを出力．$#tbtxは配列の最大の添え字です（C shellでは個数でしたが）．
a6. メッセージを表示して，プログラムを終了します．

ファイル名の一部を入力して新しく追加した機能を確認した後，この作成したスクリプトを任意のディレクトリで使用できるようにします．今後のことを考えて，perlのスクリプトを格納するディレクトリを作成することをお勧めします．

ファイル編集のコマンドとする手順

```
01  % cd
02  % mkdir myPerl
03  % cd myPerl
04  % mv ~/pjsonic/src/soldor/*.pl .
05  % emacs exam.pl (exam.plに2行追加)
        #!/usr/bin/perl
06      $home = $ENV{"HOME"};       # <==add
07      push( @INC, "$home/myPerl" ); # <==add

08  % emacs .cshrc (.cshrcに1行追加)
09      alias pedt '~/myPerl/exam.pl¥!*' #<==add
10  % source .cshrc

    % cd ~/jt60/src/simdp
    % pedt inp
        ファイルの選択&ファイルの編集
```

04. 作成したスクリプトを新しいディレクトリに移します．
05. exam.plはライブラリーとして，tblst.plを用いていますが，それを探すパスを指定する必要が有るので，2行追加（最初に）します．
06. C shellでは，ホームディレクトリは「~」で表せますが，

Perl にはこの機能が有りませんので，環境変数から取得しなければなりません．環境変数は連想配列%ENVに格納されており，数字で指定するのではなく，名前で指定します．

07. 特別な配列変数@INC には，ライブラリーを探すディレクトリパスが格納されています．push 関数を用いて，あなたのスクリプトがあるパスを追加します．
08. エイリアス（別名）を登録するため，環境設定ファイル.cshrc に 1 行追加します．
09. エイリアスの名前を pedt とします．「¥!*」は，引数をスクリプト exam.pl に渡すためのおまじないです．
10. 修正した．cshrc を有効にします．

それでは，あなたの任意のディレクトリに移動して，pedt を用いましょう．僅か50行足らずのスクリプトで，編集作業が格段に改善されるはずです．

3．C shell と Perl との違い

これまでshellでツールを開発してきた人のために，ここでは，なぜ Perl なのかを説明します．

pedt は，あるディレクトリの ls で得た情報（ディレクトリに含まれる内容）をフィルターに掛け（テキストファイルのみ），それを画面に見やすく表示，ユーザが指定したファイルの編集を行います．複数のファイル名を見やすく表示するには，format 指定が必要ですが，shell にはないのです（もっとも awk という難解なコマンドを使用すれば可能ですが）．細かなformat制御や文字列操作のため，C言語を使わざるを得ません．この場合，実行するには，コンパイルが必要となります．Perl では，C 言語の printf 関数が含まれ，Cと同様のループ制御，条件判断等が使えます．そして，実行のためのコンパイルの必要がないので，すぐにテストができます．コンパイルは数秒でしょうが，この間，集中していた思考が中断されます．この違いが大きく開発のスピードに影響するのです．

C shell には，サブルーチンの概念がなく，また子のシェルから親へ情報を伝える術はありません．子の処理した結果をファイルに書き出し，親はそれを読みとって処理することになります．しかし，Perl では，スクリプトの中の変数は，通常 C 言語のグローバル変数であり，どの階層のサブルーチンであっても使用可能です．このことが，ツールの開発を一変させます．単一の機能を持ったスクリプトを組み合わせることで，大きなツールの開発が可能となります．これが，どれだけツールの開発を，メインテナンスを楽にするか，コード開発した経験のある人なら理解できるはずです．

しかし，このグローバル変数には，注意が必要で，思わぬ動作をする場合があります．そのサブルーチン内で使うだけの変数には，local と定義するよう習慣（特にループ制御変数）づけて下さい．サブルーチンのテストではうまく行ったのに，いざツールに組み込むとうまく動作しない．「なんで？！」となった時には，このグローバル変数のことを思い出して下さい．Perl バージョン 5 には，Fortran の implicit none に相当する（use strict 'vars';）宣言文があります．この宣言を行うと，変数の型（our/my）を宣言せざるを得ず，この種のトラブルは確実に避けられます．

C shell でよく使うフィルター grep，sed，sort，uniq 等は，ファイルからのデータ抽出ですが，Perl では，同じ機能をもった関数があり，スクリプトの変数に対して行うことができます．Perl の正式名称は，The Pratical Extraction Report Language（実用的情報抽出レポート言語）の略で，もともとファイルのテキスト操作，情報抽出して簡単にレポートを作成するために開発されたものなのです．そのため，こうしたフィルター，文字列操作が充実しています．ここでは，説明しませんが，ファイル操作（オープン，リダイレクション）も簡単に行えます．スクリプトを作ってみようと思われた方は，Webには Perl 入門のホームページがたくさんありますので，さらに詳しい情報（変数の scope の概念，Perl 5 で導入された my 変数等）はそこから得て下さい[1]．

4．MT-UNIX 環境

ここでは，MT 環境について，簡単に紹介します．

学会のホームページより，PerlM.tar.gz（サイズ24K バイト程度）を取得し，使用する計算機のあなたのホームディレクトリにコピーします．PerlM のディレクトリが既にある場合には，名前を変更して下さい．

```
01  % gzip -d PerlM.tar.gz
02  % tar xvf PerlM.tar
03  % cd PerlM
04  % perl setup
    Let's start to use MT-unix Tool.
        New-command (pls, pcd, ped, pfnd, plk)

05  % echo "source ~/PerlM/palias" >> ~/.cshrc
06  % source ~/.cshrc
```

05のコマンドは，エイリアスを定義したファイル palias を環境設定ファイル．cshrc に 1 行追加します．06のコマンドは，この修正を有効にします．これでセットアップは終了．シェルとして，bash を使っている方は，ファイル palias_bash をお使い下さい．

(1)　まずは，pls でディレクトリの内容を調べましょう．

```
% pls
current directroy : /grp02/j3859/PerlM
[directory] 6
    Myst  Tips  exjt60  pjsol2d  test
    Prj
[text file] 2
    palias  setup
```

ディレクトリとファイルが分離されて表示され，UNIX コマンドの ls よりもわかりやすい表示となっています．

(2) ディレクトリの移動（pcd）

```
% pcd -d
```

と入力すると，現在のディレクトリ以下にあるディレクトリの絶対パス名が表示され，新規にニックネイム（最下位のディレクトリ名）が登録されます．これ以降ニックネイムで直接移動が可能です．

```
% pcd -t  登録されたニックネイムの一覧が表示
% pcd T   Tips のディレクトリに移動
% pcd run 計算実行のディレクトリへ移動
      1 runA : /grp02/j3859/PerlM/exjt60/runA
      2 runB : /grp02/j3859/PerlM/exjt60/runB
      3 runC : /grp02/j3859/PerlM/exjt60/runC
>> select number (2) ==> 3
new dir: /grp02/j3859/PerlM/exjt60/runC
ディレクトリ移動の履歴を見るには
% pcd -10
      1 runC  : /grp02/j3859/PerlM/exjt60/runC
      2 Tips  : /grp02/j3859/PerlM/Tips
      3 PerlM : /grp02/j3859/PerlM
>> select number (2) ==><return key>
    Command was canceled.
```

(3) ファイルの編集（ped）

ファイルの編集は以下のように行います．

```
% pcd test
% ped    ディレクトリのテキストファイル一覧
current directory : /j3859／PerlM/test
  1: .P_fpinp     8: Tpgtvl.pl  15: Ttiminod.pl
  2: Tpath.pl     9: Tpinp.pl   16: Ttimjpn.pl
  3: Tpbspc.pl   10: Tplast.pl  17: Ttimunx.pl
>> select number (2) ==> 2
target file : Tpath.pl in /j3859/PerlM/test
エディターが起動
% ped ex      名前の頭に ex の付くファイルを編集
% ped -1      ped を用いて最後に編集したファイル
% ped -5      最後から 5 番目以内のファイルから選択
% ped new.f   新規にファイル new.f を作成
```

(4) ファイル，ディレクトリの探索（pfnd）

2002年の PSI 会議で発表したシミュレーションを再開する必要が出てきたとして，あなたはその作業領域を探し出すことができますか？会議は 5 月に開催されたので，2002年の 4 月から60日の間に計算しているはず．この期間に作成した入力ファイル（inppls）を含んだディレクトリの親を探索します．MT-UNIX では以下のように行います．

```
% pcd            ホームディレクトリへ移って
% pfnd
*** Welcome to [pfnd] - command ***
>> enter type (d:dir/f:file/<rtn>:quit) ==> f
>> enter command (t/n/m/e/g/l/s/q) ==> t
 selection [time]
>> date (2002_3_5/<rtn>:now/quit)==> 2002_4_1
>> date (2002_5_31/60d/-2d/-5h/q)==> 60d
  contains 3690-active files (total:70470)
>> enter command (t/n/m/e/g/l/s/q) ==> n
  selection [name]
>> enter top name (mon/<rtn>:quit) ==> inppls
    contains 159-active files (total:70470)
```

7 万個のファイルから159迄絞れました．出力ですが，親（m）はまだ159と多いので，親の親（m 2）を出力

```
>> enter command (t/n/m/e/g/l/s/q) ==> l
>> enter type (f:full/t:time/n:name/
                    m:mother - dir/q) ==> m2
    mother - level = 2 number of dir = 3
      1 ./ORG38/exe/sol2d/exesc/runA
      2 ./ORG38/exe/sol2d/exesc/runB
      3 ./ORG38/exe/sol2d/exesc/runC
    contains 159 - active files (total:70470)
```

そうだ exesc のディレクトリだった！

数値積分の必要があるけれど，確か数年前に Gauss－Legendre の積分公式を使って，kinetic thermal force の数値計算を行ったはず．でも，ディレクトリ，ファイル名なんだっけ？こんな時も，記憶の断片 Gauss を手がかりに pfnd コマンドが探し出してくれます．

(5) コード解読のツール（plk，pinq）

最近の計算機の能力向上に伴い，モデリングの精密化が進み，コードのサイズは増大の一途です．2，3 千ステップならともかく，2，3 万ステップとなると，自分で開発したコードでさえ，コモン変数がどのルーチンで，どのような計算式で評価され，それがどのインクルードファイルに含まれ，何次元でその配列サイズは，これらを正確に覚えておくことは不可能です．あやふやな記憶に頼って，コードの改良を行うことはできません．常にコードの中身を調べながら行わねばならず，コード全体のサイズ増大とともに，その確認が負担となります．まして，他研究所で開発されたコードを導入したとき，コード解読は非常に忍耐のいる作業となります．

通常，変数名のトレースには，grep が用いられますが，以下の点が不満，というより不備です．

- 単語単位での検索のオプション-w のない計算機が存在．
- 巨大なコードに対して，表示される行数が多く，肝心の情報が埋もれてしまいます．
- 文字列が現れた行しか表示されないので，継続行がある場合には，それは意味不明です．

この点を改良したツール plk を作成しました（現在の所 Fortran90の継続行には未対応です）．ディレクトリ monte のファイルを対象に，eion の単語を探索する時，monte の親ディレクトリで以下のように用います．

```
% plk eion -d monte
### /j3859/sol2d/src/soc/monte ###
io ntinpt.f: namelist /untinp/
io       > lntmd, nntl, dtntl, imxnt,
io       > eioni, eion, famp1, famp2
io ntout.f: write(n6,'(5x,"eion =",f8.3,
io       >"eioni =",f8.3)') eion, eioni
       ntsrca.f: zwe =-zsi*eion*cev
def ntwrad.f: eion = 40.0
```

ここで，行の先頭の，incはインクルードを，ioは入出力を，defは定義を表します．

実行後，pdspと入力すると，ヒットしたファイルの中身をエディターで詳しく調べることができます．巨大なコードの場合，扱うディレクトリが多いので，これをprojectデータとして登録しておくと，pinq eionと簡単に調べることができます．

5．Tipsの紹介

スクリプトは小さな単一の機能をもつ部品を組み合わせて作ります，ここでは，汎用性のある基本的部品（tips）を紹介します．ファイルの中の一部を修正したいということはよくあります．通常sedが用いられますが，これを使いこなせる人はそう多くはいません．namelistによる入力データに特化して，値を変更するツールがTpinp.plです．

```
% pcd test
% Tpinp.pl
--- Usage ---
Tpinp.pl 'mxcpu = 125, .....' inppls > inpnew
Tpinp.pl 'cftr="dtprf_03",..' inppls > inpnew
Tpinp.pl 'pfmag(1)=7.0d22, ' inppls > inpnew
```

Usage以下に現れた3行のメッセージは，テストデータです．マウスで1行を切り取り実行して下さい．そして，新入力ファイルinpnewの値を確認して下さい．

シミュレーションのパラメータサーベイで便利なスクリプトTrunも実行してみて下さい．リスタートファイル名を間違えたり，入力値を間違えて，長い待ち時間の後「しまった！」ということがなくなるはずです．

Tpextm.pl	大量の配列データをコンパクトに表示
Tpvlst.pl	配列データを縦方向に整列して表示
Tpvsel.pl	配列データを表示して選択させる
Tvindx.pl	配列を縦方向に表示する時の添え字
Tpuniq.pl	配列の中で重複するデータは削除
Tpls.pl	ディレクトリの内容を表示
Tpgtdr.pl	現在の位置以下のディレクトリ名を取得
Tpmdir.pl	何代も遡って親のディレクトリ名を表示
Tpfsfx.pl	ファイル名のサフィックス番号の最大値
Tpspltv.pl	文字列を分解して，名前，数値データに
Tpgtvl.pl	namelistのファイルより数値を取得
Ttimunx.pl	2005_05_14の表示をUnix timeに
Ttimjpn.pl	Unix time(秒)を日本時間に変換
Ttiminod.pl	ファイルの最終修正／アクセス時間

6．おわりに

昨年編集委員として，Cshellで昔作成したKSTOOLの一つncd（pcdに相当）を紹介することとなりました．Perlの勉強に良い機会と考え，8年ほど前に購入した「Perl (Ver.4) 入門」[2] を本棚から引っぱり出し，勉強を始めました．KSTOOLの中のコマンドをPerlで書き直していくうち，スクリプトの記述が簡単なのに感激しました．KSTOOLでは，複雑な処理や表示はC言語で記述しているので，すっかりCを忘れた著者には改良はもはや不可能と諦めていたからです．Perlは理解しやすく，複雑な処理や表示が可能なC shellに代わるコマンドインタプリタです．

そして，Perlの可能性を確かめるため，

- 記憶の断片からファイルを探し出すpfnd,
- 巨大コードの解読をサポートするpinq,
- パラメータサーベイを簡単確実に実行するTrun

を新たに開発しました．

その開発中に，一冊の本と出会いました．「UNIXという考え方」[3]です．計算機ユーザ，物理コードのユーザをサポートされる方には，是非読んでいただきたいと思います．プログラムはかくあるべきとの考えが，経験に基づき示されています．構築したシステムが時とともにどのように変遷していくか，そして開発者の論理に立った巨大なシステム（多機能主義），移植性に注意を払わなかったシステムのたどる道が示されています．私には，拘束的インターフェイス，独自技術症候群の問題点が気になりました．そして，「全てのプログラムはフィルターとして設計」は，目から鱗でした．80%はでき上がっていましたが，この観点から，pfnd，pinqは作成し直しました．

あなたは，端末の前で繰り返し同じ作業を行っていないでしょうか．コンピュータができる作業を．こんな時，Perlによるスクリプト作成を考えてみて下さい．Perlは習得が，C shellやC言語よりはるかに簡単です．一度習得すれば，纏まった手順の処理が，コマンド入力だけでコンピュータが確実に実行してくれます．

参考文献

[1] http://www.site-cooler.com/kwl/perl/
http://www.tohoho-web.com/wwwperl1.htm.
[2] Ellie Quigley著「Perl入門」トッパン社（1997）.
[3] Mike Gancarz著「UNIXという考え方」オーム社(2004).

MT-UNIXに関してのトラブルや，使用方法についての質問（例えばTrun）があれば，メイル（kshimizu@naka.jaeri.go.jp）でお知らせ下さい．