MT-Tool 解説

2005 年 6 月２１日作成
日本原子力研究所那珂研究所
清水勝宏

０．はじめに

コード開発において，ディレクトリの移動，ファイルの編集，実行は日々何回も行うものです。こうした作業を効率的に行うためのツール（MT-Tool）を開発したので、ここに紹介します。Perl で開発しているので，どんな計算機（パソコンも）にも移植可能です。

名前の２，３文字の入力で、直接そのディレクトリに移動したり(pcd)，ファイルの編集（ped）が可能です。何年も前に作成したプログラム，計算実行のディレクトリを記憶の断片（作成時期，コメントに書いた解法等を手がかりに）からファイル，ディレクトリが探索（pfnd）できます。また，巨大コードの変数トレースをサポートする pinq，パラーメータサーベイを簡単確実におこなう Trun のツールがあります。

MT-tool を利用するだけでしたら、Perl の知識がなくても利用できます。自分流にカスタマイズするのであれば、学会誌 6 月号の記事「Perl 入門」の知識で十分です。自作される場合には、制御構造（if/elsif/else）、数々の関数（grep、push、split、sort）、パターン照合演算子 （=~ s/xx/yy/等）、ファイルハンドル等の知識が必要となります。Web には Perl 入門のホームページがたくさんありますので，それらの情報はそこからえてください。特に文献[1]に示すホームページは良く纏まっています。MT-tool のスクリプトを参考に、具体的な記述方法、機能を確かめると、Perl の理解が進むと思います。本格的にシェルを作成しようとされる方は、変数の scope の概念を、Perl５で導入された my、our の考え方をきちんと勉強される事をお勧めします。

1. セットアップ

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，使用する計算機のホームディレクトリにコピーします。ディレクトリ PerlM が既にある場合には，名前を変更して下さい。プロンプト%が先頭にある行が、あなたが入力すべきコマンドを示します。

```
01 % gzip -d PerlM.tar.gz
02 % tar xvf PerlM.tar
03 % cd PerlM
04 % perl setup

     Add the 1-line to [.cshrc].
a1       echo "source ~/PerlM/palias" >> ~/.cshrc
     Please enable new-.cshrc
a2       source ~/.cshrc

   Let's start to use computer under the MT-unix environment.
        New-command (pls, pcd, ped, pfnd, plk)

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

05 のコマンド入力には，a1 のメッセイジをマウスで cut&past してください。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 に登録します。06 のコマンドには，a2 のメッセイジを用います。修正された.cshrc を有効するコマンドです。シェルに C shell を用い、ファイル編集に emacs を用いている方は、これでセットアップは終了です。

・シェルとして bash を使っている方は，ファイル palias_bash を使います。05 のコマンドで、
`echo "source ~/PerlM/palias_bash" >> ~/.cshrc` とします。
・ファイル編集のエディターとして、vi あるいは mule を用いている方は、palias の 3 行目
`setenv PEDT  emacs   # CHOOSE from (vi/emacs/mule)`
の下線部を変更してください。

2. 各コマンドの説明

2.1 ディレクトリの内容の表示（pls）

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 よりも分かりやすい表示となっています。tar ファイルやオブジェクトモジュールがある場合には、[nonT file]に分類されて表示されます。

2.2 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。

```
% pcd  -d       ニックネイムの新規登録
get nick-names of subsequent directory.
    from current directory : /home/g9/j3859/PerlM
  /home/g9/j3859/PerlM
  /home/g9/j3859/PerlM/Myst
  省略
Total number of directory = 23      PerlM 以下に 23 個のディレクトリが存在
new dir: /home/g9/j3859/PerlM       現在いるディレクトリの表示

% pcd  -t       登録されたニックネイムの一覧が表示
  Myst     inc      pjsol2d  runC     sonic
  PerlM    monte    rand     size     src
  Prj      montei   runA     soldor   ssl
  Tips     mplnk    runB     soldori  test
  exjt60   org
Total number of nick name = 22  [.]
new dir: /home/g9/j3859/PerlM
```

登録の後、ニックネイムの２，３文字の入力で、該当するディレクトリに直接移動できます。

```
% pcd  T        Tips のディレクトリに移動
   1  Tips       : /grp02/j3859/PerlM/Tips
new dir: /grp02/j3859/PerlM/Tips      移動後のディレクトリの確認

% pcd  run      ディレクトリ（runC）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
```

それでは、あなたがコード開発をおこなっている上位のディレクトリに移って、

```
% pcd       引数が何もないと、ホームディレクトリに移動
% cd  project-name
% pcd  +d   +d は、ニックネイムの追加登録を意味します。
% pcd  -t   ニックネイムのリスト出力
```

（注：-d は、既に登録されているニックネイムをクリアーし、新規に登録を行います。）

ニックネイムでディレクトリをビュンビュン飛び回ってください。それでは、これまでの移動の履歴を調べましょう。

```
% 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>  ディレクトリの移動を行わないとき
  Command was canceled.
```

2.3 ファイルの編集（ped）

ファイルの編集は以下の様に行います。

```
% pcd test
% ped          引数が無いと、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
```

その他のファイルの指定方法

```
% ped ex          名前の頭に ex の付くファイル（現在いるディレクトリにある）を編集
% ped -1          ped を用いて最後にアクセスしたファイル
% ped -5          最後から５番目以内にアクセスしたファイルから選択
% ped new.f       新規にファイル new.f を作成
```

ディレクトリ、ファイルの履歴情報は、標準として最大３０個まで記録します。この数をもっと増やしたい場合には、PerlM/Tips/pinit.pl のファイルで

```
## max-record
$MT_mxrcf = 30;     記録するファイルの最大数
$MT_mxrcd = 30;     記録するディレクトリの最大数
```

を修正します。

2.4 ファイル，ディレクトリの探索（pfnd）

（1）修正時期でファイルを探す

2002 年の PSI 会議で発表したシミューレ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-Tool では以下の様に行います。

```
% 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
 Executing find-command.                        多少時間（１～２分）がかかります。
  [.MT_myfndfl] made at 2005_05_11/22:57:29
    contains 70470-active files  (total 70470)
                                                 え！　７万個もファイルがあるの？
                                                 修正時期で絞って
>> enter command (t/n/m/e/g/l/s/q) ==> t
 selection [time]
>> enter date (2002_3_5/<rtn>:now/quit) ==> 2002_4_1
>> enter date (2002_5_31/60d/-2d/-5h/-15m/q) ==> 60d
  contains 3690-active files  (total:70470)
                                        入力ファイル名 inppls でさらに候補を絞ろう
>> enter command (t/n/m/e/g/l/s/q) ==> n
 selection [name]
 ntfldp  ntptrc  outsta  plntwg  pxpres  shsub
  @@@     FigNTL  Sp_EI   cputim  inppls  mspvel
>> enter top name (mon/<rtn>:quit) ==> inppls
  contains 159-active files  (total:70470)
```

７万個のファイルから、159 迄かなり絞れたので，リスト出力

親のディレクトリ（m）は 159 と多いので，親の親(m2)を出力

```
>> enter command (t/n/m/e/g/l/s/q) ==> l  (エルです)
>> 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！
>> enter command (t/n/m/e/g/l/s/q) ==> q  終了
```

ファイルの選択で、５時間以内にアクセスしたファイルを選択する場合

```
>> enter command (t/n/m/e/g/l/s/q) ==> t
>> enter date (2002_3_5/<rtn>:now/quit) ==> <rtn>       現在の時刻より
>> enter date (2002_5_31/60d/-2d/-5h/-15m/q) ==> -5h    ５時間前
```

(2) コメントよりファイルを探す

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？ でも大丈夫。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。

```
% pcd
% pfnd
*** Welcome to [pfnd]-command ***
>> enter type (d:dir/f:file/<rtn>:quit) ==> f
  Use Find-file created 2 days ago ?
(<rtn>:y/n) ==><rtn>                    古いファイルを探すから 2 日前の情報で良いか

  [.MT_myfndfl] made at 2005_05_11/22:57:29
    contains 159-active files  (total 70470)
>> clear flag ? (y/n:<rtn>) ==> y    前回の選択のフラッグはクリアーしてと。

contains 70471-active files  (total 70471)
>> enter command (t/n/m/e/g/l/s/q) ==> e  拡張子で選択する
 selection [extension]
 Example of extension
  BAK   c   cdf    f    ff     h   （あなたのファイルの拡張子が出力）
>> enter extension (f/c/tar/<rtn>:quit) ==> f  ソースだから，拡張子 f で絞ろう。
  contains 4897-active files  (total:70471)
```

キーワード Gauss で grep をかけてと。5000 近いファイルでも 2～3 分程度。

```
>> enter command (t/n/m/e/g/l/s/q) ==> g
 selection [grep]
>> enter input of grep (aimas) ==> Gauss
  contains 12-active files  (total:70471)
```

オッ！ 12 件ヒットした。
メニュー画面（show のコマンド）で調べよう

```
>> enter command (t/n/m/e/g/l/s/q) ==> s
メニュー画面
### enter [1] at the head of line ###
1   ./ORG38/imp/src/kinfti/gasleg.f
1   ./ORG38/sol2d/src/ssl2/mtsolv.f
   ./ORG38/sol2d/src/random/gauss.f
```

これではないかと思われるファイル名の行の先頭に 1（いち）を入力。メニュー画面を保存終了すると，選択されたファイルのエディター画面となるので，探していたものかどうか調べます。オリジナルファイルをコピーして表示しているので、ファイルを壊す心配はありません。選択したファイルを全て見終わると，

```
>> show another files (<rtn>:y/n:quit) ==> n
```

が出てくるので，探し当てた場合には，n とします。

pfnd のコマンドを終了しても、ヒットしたファイルの情報は保存しているので、pdsp コマンドで直接（pfnd を実行しなくても）、再度ファイルを調べることは可能です。

```
% pdsp
メニュー画面
### enter [1] at the head of line ###
    ./ORG38/imp/src/kinfti/gasleg.f
    ./ORG38/sol2d/src/ssl2/mtsolv.f
1   ./ORG38/sol2d/src/random/gauss.f
```

pfnd はファイル/ディレクトリの探査が強力なツールなので，「エーット，あれは何処だっけ？」となった時お使い下さい。

2.5 コード解読のツール（plk）

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 の無い計算機があります。
・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。本当に知りたいのは、計算式であったり、ＩＯ文（read/write/namelist）であったり、include file 名であったりと、場合により異なります。
・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様に用います。

```
% 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。
複数行に渡った文が出力されるので、eion の入力が ntinpt.f のファイルで、namelist untinp で行われることが理解できます。

探す変数が基本的な変数の場合、表示される行数が多くなり、そこから必要な情報を探しだすのに一苦労です。こんな時、行の先頭にある情報を使えば、必要な情報のみが出力されます。

```
% plk eion -d monte | grep ^def
def ntwrad.f:    eion = 40.0
```

monte のディレクトリにいる場合、grep のコマンドと同じような使い方もできます。

```
pwd
/j3859/sol2d/src/soc/monte
% plk eion *.f
% plk eion ntout.f
```

実行後，pdsp と入力すると，pfnd と同様に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もできます。

2.6 巨大コードの開発／ユーザの方に（prj）

コードの巨大化に伴い，ソースは数多くのディレクトリから成ります。変数調査のためいちいちディレクトリを移動する煩わしさから解放されるため，コードを構成するディレクトリを指定したデータ（project データ）を用いて単語検索を行うのが，pinq です。その説明の前に、project について説明します。

project データは、~/PerlM/Prj/sol2d のファイルを例に作成します。

```
cmt   sol2d-code   2005/02/08
cmt     soldor/neut2d + radiation model

inc   ~/PerlM/pjsol2d/inc/size
inc   ~/PerlM/pjsol2d/inc/soldori

src   ~/PerlM/pjsol2d/src/sonic
src   ~/PerlM/pjsol2d/src/soldor
src   ~/PerlM/pjsol2d/src/rand

dir   ~/PerlM/exjt60/runA
dir   ~/PerlM
```

行の先頭 3 文字が cmt は、コメントを、inc がインクルードの、src がソースのディレクトリを現します。dir は、ソースコードとは関係なしに、よく用いるディレクトリ、例えば、実行のディレクトリを指定します。（sol2dB の例は、ディレクトリの指定を最小限にしたい面倒くさ

がりの方の例です。*によって、以下のディレクトリを対象とすることができます。）
コード開発を行っているとプロジェクトが幾つもできてきます。そこで、prj コマンドでその指定を行います。

```
% prj
===== prj-data ====================
   1: sol2d         2: sol2dB
>> select number (2) ==> 1
Prj-name : sol2d
cmt   sol2d-code   2005/02/08
cmt     soldor/neut2d + radiation model   確認のため、コメントが出力
```

project データを作成し、prj コマンドでその指定を行った場合には、pcd -p を実行してください。プロジェクトに関わるディレクトリのニックネイムが新規登録されます。追加登録には、pcd +p です。これにより，２，３文字で一意的にディレクトリが決まり移動がよりスムースに行われます。

```
% pcd -p
>> Prj-name : sol2d  O.K. (<rtn>:y/n) ==> <rtn>
 Wait a moment till register nick-names.
    using prj-data sol2d
  /home/shimizu/PerlM
  /home/shimizu/PerlM/exjt60/runA
  省略
Total number of directory = 11
new dir: /home/shimizu/PerlM/Prj
```

2.7　巨大コードの解読（pinq）

prj コマンドで、project データを指定しておくと、単語の検索に、ディレクトリの指定（移動）は不要となり、どこにいても簡単に調べることができます。

```
% pinq eion
Prj-name : sol2d
cmt   sol2d-code   2005/02/08
cmt     soldor/neut2d + radiation model

  in progress  [/home/shimizu/pjsonic/soc/soldor]
  in progress  [/home/shimizu/pjsonic/soc/monte]
```

調べた結果は，いろんな形で見ることが出来ます。

```
% pinq -c   :  UNIX の cat コマンドで
% pinq -e   : エディターで
% pinq -o   : inc, def, io, 参照の順で outline のみ
% pinq -s   : pdsp を用いて（メニュー画面でファイルを指定）
% pdsp      : メニュー画面でファイルを指定
```

3. Tips の紹介

今回紹介したツールをみて、自分だったらこういう仕様にするとか、こんなツールを作ってみようと思われたかもしれません。そのような人のために、汎用性のある基本部品（tips）を紹介します。この tips をドライブするテストプログラムが test のディレクトリにあるので、それを動かして、その機能を理解してご利用ください。

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

3.1 Tpvlst.pl

これは、配列変数を番号と一緒にリスト出力し，番号で変数を選択する機能をもつ tips です。ユーザに選択させるときに良く用います。

```
% pcd test
% Tpvsel.pl
length of string = 6,  number of column = 4
case lfix = 0
   1: January   4: April    7: July    10: October
   2: February   5: May     8: August  11: November
   3: March    6: June     9: September  12: December
>> select number (2) ==> 2
 Your choice : [February]

case lfix = 1
   1: Janua+   4: April    7: July    10: Octob+
   2: Febru+   5: May      8: August  11: Novem+
   3: March    6: June     9: Septe+  12: Decem+
>> select number (2) ==> 5
 Your choice : [May]
```

表示形式は、グローバル変数　$MT_lnstr = 6（文字数）、$MT_ncolm = 4（コラム数）で指定します。文字数が$MT_lnstr を超えたときの処理を指定するのが、引数$lfix です。$lfix=1

とした後者の方が見やすい表示になっています。このとき 6 文字を超えた場合、6 文字目が+となっていることに注意。例では、月を格納した配列変数@month の選択を行います。
そのコールは　$i = &pvsel($lfix, @month);　です。$i に選択された変数の添え字がセットされます。

3.1　Tpextm.pl

pfnd のコマンドの説明に示しているように、名前でファイルの選択を行うとき、現在対象としているファイルの名前が表示されます。ユーザはそれを参考に名前で絞ります。この時表示されるファイル名は非常に大量なので､これをコンパクトに出力するために作成した tips です。Tpextm.pl とキー入力して、要素の多い配列データ@tbdat（Jun_10 といった文字列が２００個格納）をどのように表示しているか確かめてください。

```
$no = &pextm($mlen,@tbdat);
```

としてコールすると、文字数$mlen(=3)として、配列を作成し直し、ソートし、重複名を削除して出力しています。$no は出力した変数の数がセットされます。

3.3　Tpinp.pl

ファイルの中の一部を修正したいということは良くあります。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。

```
% Tpinp.pl
--- Usage ---
Tpinp.pl 'mxcpu = 125, itend=180, rcy_al = 4*0.999d0, limp = 3' inppls > inpnew
Tpinp.pl 'cftr = "  dtprf_03", cftw = "dtprf_07", ' inppls > inpnew
Tpinp.pl 'pfmag(1)=7.0d22, flxni=3.0d22, flxqe=20.0d6 inppls > inpnew

and then,  diff -w inpnew inppls     for check
If not work well, cat .MT_fpinp
```

Usage 以下に現れた 3 行のメッセイジは，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。オリジナルの入力ファイル inppls との比較を行うのであれば、diff -w inpnew inppls とします。

namelist の数値変更のためには、文字列解釈をする必要があります。それには、split 関数が威力を発揮しますので、簡単にその使い方を説明します。次の文字列$strg としますと、

```
$strg = "lstep = 2, mxcpu = 180, lpls = 3, lntl = 3, limp = 2,";
@wrds = split(" |=|,", $strg);
```

この実行により、配列@wrds には、変数名、値が交互に格納されます。

```
@wrds = lstep 2 mxcpu 180 lpls 3 lntl 3 limp 2
```

3.4 Trun.pl

シミュレーションのパラメータサーベイで、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！」という経験はないでしょうか。対話形式で入力ファイルを作成し、パラメータサーベイを簡単に確実におこなうツールが Trun です。

```
% Trun.pl
>> enter  read-rst file (<rtn>:dtprf_15/spc:ini-cal) ==> <rtn>
     start calculation from dtprf_15
>> enter write-rst file (<rtn>:dtprf_16/dtprf_07) ==> <rtn>
     restart file [cftr = "dtprf_15",  cftw = "dtprf_16",]

  name     = default when <rtn>
-------------------------------------------------
  mxcpu    = 180        >> enter (<rtn>/new-V) ==> 30
  itend    = 2000       >> enter (<rtn>/new-V) ==> 500
  flxni    = 1.5d22      >> enter (<rtn>/new-V) ==> <rtn>
  pfmag(1) = 0.5d22      >> enter (<rtn>/new-V) ==> 3.0d22
  rcy_al   = 0.980d0, 0.980d0, 2*1.0d0    >> enter (<rtn>/new-V) ==> <rtn>

*** New parameter ***
  mxcpu    = 30
  itend    = 500
  flxni    = 1.5d22
  pfmag(1) = 3.0d22
  rcy_al   = 0.980d0, 0.980d0, 2*1.0d0
  cftr     = "dtprf_15"
  cftw     = "dtprf_16"

 input data O.K.  (y/n) ==> <rtn>
    Please answer y or n.
 input data O.K.  (y/n) ==> y

New input data [inppls_new]
```

(a)

(b)

(c)

(a)まず、読み込みのリスタートファイルの指定を行います。例では、リスタートファイルの名前は、dtprf としています。現在いるディレクトリの中にある dtprf の名前を持つファイルから、suffix の最大のファイルを、リスタートファイルの標準として表示されます。このデータから計算を始めるときは、リターンキーを入力します。最初から計算するときには、スペースを入力します。書き出しのリスタートファイル名も標準のものが表示されます。

(b)この後、パラメータサーベイを行う入力データを会話形式で指定します。巨大コードの場合、namelist に現れる変数は非常に多いものですが、一連のパラーメータサーベイにおいては、変更すべき入力値は数個程度です。これだけを会話形式で入力するので間違いは無くなります。また、標準値が示され、それを用いるときには、リターンキーの入力で済みます。新しい値で計算する変数に対してのみ、その数値を入力します。

(c)新しい入力データが表示されますので、確認を行います。この確認は重要なので、<rtn>は受け付けませんので、y もしくは、n と入力します。y と確認が行われると、新しい入力ファイル（inppls_new ）が作成され、表示されます。

実際に Trun を実行してみて、使えそうだと思われた方は、以下の手順に従ってあなたのコードの実行シェルとしてください。

１．適用可能かのチェック（特殊な使い方をしない限り問題無いはず）

(1) 入力ファイルには、同じ変数名が複数回現れない。ただし、最初のカラムが!、#、*で現れる行は、コメントとするので、ここに現れても問題ない。

(2) リスタートファイルの名前は、XXX_no の形式である。ファイルの名前の前に１つ以上のスペースがある場合（" XXX_no"）、ファイルは使わないものとする。

(3) 文字列は、「"」で挟むものとし、その間には、コンマ「,」は無いものとする。

２．あなたの実行ディレクトリに移動して、cp ~/PerlM/test/Trun.pl exinp.pl

３．入力ファイル作成のシェル exinp.pl の修正

ped exinp.pl
以下の部分をあなたの入力ファイルに対応して修正

```
### User option ###
$file = "inppls";   # namelist  （入力ファイル名）

                                （パラメータサーベイする変数名のリスト）
@parm = ( "mxcpu", "itend", "flxni", "pfmag(1)", "rcy_al" );
                  # name of important variables in namelist

                                （リスタートファイルに関連）
## If no use restart, $rsta ==> ##$rsta (comment)
$rsta = "dtprf_";    # restart file name              リスタートファイル名
$nfrd = "cftr";      # name of read-file in namelist  読み込み
$nfwt = "cftw";      # name of write-file in namelist 書き出し
if( $rsta ){ @parm = ( @parm, "$nfrd",  "$nfwt" ); }

$fnew = "$file" . "_new";       （作成した入力ファイル名）
```

入力データファイル inppls の一部を以下に示しますので、それとの対応を見てください。

```
!  flxni = 3.0d22,  flxqi = 12.0d6,  flxqe = 12.0d6,   （コメント）
 &usonic
   lstep = 2, mxcpu = 180, lpls = 3, lntl = 3, limp = 1, &end
&uplinp
   nion = 1, aion(1) = 2.0d0, aza(1) = 1.0d0, fcna(1) = 1.0d0,
   nftr = 35, cftr = "dtprf_05",
   nftw = 36, cftw = "dtprf_06", ndsk = 500,
   itend = 2000, dtmax = 1.0d-4, dtmin = 1.0e-9,
   flxni = 1.5d22,  flxqi = 6.0d6,  flxqe = 6.0d6,  &end
```

リスタートファイルを使わない場合には、
##$rsta = "dtprf_";   # restart file name
として、この行をコメントとしてください。

4，exinp.pl の実行

chmod 755 exinp.pl
exinp.pl
新しく作成した入力ファイルをチェック

5．exinp.pl で、サブミット迄行う場合

以下の文を exinp.pl の最後にいれる。
system("csh sub.sh");
サブミットシェル sub.sh の中で、入力ファイル名を、$fnew の名前とすることを忘れずに。

3.5 timnod.pl, timunx.pl, timjpn.pl

UNIX では、ファイルについての情報（大きさ、変更日付等）は、i ノードに格納されています。この情報を元に、ls コマンドでは、ファイル、ディレクトリの内容が表示されています。ls より得られる日付では、秒単位での比較はできません。Perl には、i ノードの情報を取得する関数 stat があるので、これを用いてファイルの最終修正時刻を取得するのが、timinod.pl です。その時刻は 1970 年 1 月 1 日からの秒数です。

```
$mtim = &timinod( $file, "m" );    最後に変更された時刻
$atim = &timinod( $file, "a" );    最後にアクセスした時刻
$ctim = &timinod( $file, "c" );    最後にファイルの状態を変更した時刻
```

UNIX 時刻（秒数）は、比較するときには良いのですが、それがいつかはわからないので、西暦、月、日、時刻（日本時間）に直すのが、timjpn.pl です。

```
% Ttimjpn.pl
  Unix Time at now = 1119505860
  Jpn  Time at now = 2005_06_23/14:51:00
```

西暦_月_日で表された時間を UNIX 時刻に変換するのが、timjpn.pl です。

```
% Ttimunx.pl
>> enter date  (2005_01_10/<rtn>:today) ==> 1980_6_23
  date     : 1980_6_23
  unix time : 330534000 (sec) from 1970_01_01
```

ファイル探索 pfnd コマンドで、アクセスした期間でファイルを絞るときに作成しましたが、その用途は広いと思います。

4. おわりに

昨年編集委員として，C shell で昔作成した KSTOOL の一つ ncd（pcd に相当）を紹介する事となりました。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%は出来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るのです。

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。