6.自分専用の環境をつくる。

ここまでで大体UNIXで仕事ができるようになりました。そうするともっと快適に計算機を使いたくなります。例えば印刷したいときはいちいち-Pオプションを使わなくても一番近くにあるプリンターに出力してもらいたいものです。その他こまごましたことも含め、自分にあった環境の整備がしたくなります。

6.1.環境とは？

UNIXで言う環境には、様々なものがありますが、コマンドやファイルの検索順序やプリンター、端末などの装置の設定、特定のコマンドで使うファイルやパラメータの標準値などが考えられます。自分専用のコマンドを用意することなども含まれます。このうち例えばcshのaliasなどはすでに見てきました。そういった自分専用の環境を表すデータベースはどこに作られるのでしょうか。すでにcshの解説のところでシェル変数については一部を紹介しました。それはcshが使うパラメータであってシェルによって起動されたコマンドに渡されるわけではありません。UNIXで使われるコマンドは別にcshだけから呼び出されるのではないので、シェルに依存したパラメータを使うことはできません。代わりに環境変数と呼ばれるデータベースを使用します。

6.1.1.環境変数

では実際にどのような環境変数が定義されているか見てみましょう。printenvというコマンドを使います。

```
% printenv
HOME=/home/iwashi
SHELL=/bin/csh
TERM=sun-cmd
USER=iwashi
PATH=/home/iwashi/bin:/usr/local/bin:/usr/ucb:/bin:
/usr/bin:/usr/bin/X11:/usr/new:.
MAIL=/usr/spool/mail/iwashi
%
```

これは環境変数の極く一部ですが、初めてログインしたときなどは場合によってはこの程度しか定義されていないかも知れません。これらの環境変数は非常に大事なものなので不注意に書き換えると何もできなくなってしまう可能性があります。とりあえず見るだけにしておいてください。ここに書かれた変数は普通cshはそのコピーをシェル変数として使っています。どちらかを設定すれば両方が同じ値を保持します。

すでに定義されている環境変数はそのままおいておきますが、自分で新しい環境変数を設定してみましょう。setenvというコマンドを使います。

```
% setenv MY_DATA_PATH ~/data
% printenv MY_DATA_PATH
/home/iwashi/data
%
```

MY_DATAPATHという変数が定義されました。この変数は自分のプログラムの中でgetenvなどを使って調べることができます。不必要になった変数は

```
% unsetenv MY_DATA_PATH
%
```

で削除することもできます。環境変数はもちろんシェルに対するコマンド文字列の中でその時の値を使うことができます。例えば次のように使います。

```
% myanalysis $MY_DATA_PATH/run20001.dat
```

こうするとmyanalysisというプログラムには/home/iwashi/data/run20001.datというファイル名が渡されることになります。場合によって環境変数と他の文字列の境目がはっきりしないと正しく変換されないときがあります。その時は変数名を{}でくくります。makeのときは()でした。cshでは()は別の働きをします。

環境変数はUNIXのコマンドやプログラムが参照するデータベースですからそのコマンド毎にどのような環境変数を参照するかが決まっています。例えばXウインドウでは、どのX端末をXサーバに使うかを表す、DISPLAYという環境変数を参照します。コマンドmanはマニュアルページを探すディレクトリをMANPATHという変数から調べます。。自分の身近にあるプリンターをPRINTERという変数に書いておくとlprで-Pのスイッチが必要なくなります。変数を設定する必要があるかどうかはmanなどでそのコマンドを調べて見つけてください。

環境として重要なのはPATHです。コマンドをどこから探してくるかを表しています。例えば他の人が使っているコマンドが自分では使えないとき、PATHにそのコマンドのあるディレクトリが書かれていないことが考えられます。PATHでは検索するディレクトリを:で区切って並べます。PATH変数に例えば/etcを追加しましょう。

```
% setenv PATH ${PATH}:/etc
```

これは、現在設定されているPATHの値と:/etcをくっつけて新たにPATHを定義しています。先ほど説明した環境変数の引用を使っています。

6.1.2.端末の設定

環境変数以外にも重要な環境があります。自分が使っている端末のタイプや受け付けるエスケープシーケンス(制御文字)はスクリーンエディターなどを使うとき正確にシステムに知らせておく必要があります。少しでも間違っているとviが変な動きをしてしまいます。環境変数にTERMという変数があって端末の型を知らせるようになっています。UNIXでは/etcにtermcapという大きなファイルがあって、様々な端末の型が定義してあります。TERM変数に書いてある型をtermcapから探してスクリーンの制御のためのエスケープシーケンスを決定します。

他の計算機にtelnetでログインすると端末の型が相手の計算機に知らされますが、そこのtermcapに自分の型がないことがよくあります。またvt100互換の端末などを使っている場合にも必ずしも完全互換でない場合があったりします。そういう場合にはスクリーンエディターはもちろん使用できなくなりますが行編集のためのキーが使えないことなども起こります。このような場合termcapの中にある端末の型にTERMをセットしてやればよいのですがほとんど呪文なのでまずわかりません。該当するものがないときはtermcapを更新すればよいのですが普通のユーザには許されていません。そこで自分の端末の機能やエスケープシーケンスを別途教えてやる必要があります。sttyコマンドを使います。

```
% stty erase ^H
```

この場合例えば自分のキーボードでBSキーを押すと一文字消して欲しいのに^Hと表示してしまう時に消去文字を^Hに変更するものです。あるいは画面に入る行数より多くの行をmoreやmanがスクロールしてしまうとき

```
% stty rows 24
```

としてやると次からは24行毎にページをめくってくれます。どのような設定ができるかは`man stty`とやってみてください。

6.2.シェルスクリプトを書く。

ファイルにコマンドを書き並べておいてそれを呼び出して実行させるコマンドプロシジャをUNIXではシェルスクリプトと呼びます。シェルスクリプトではプログラミング言語のように繰り返しや条件分岐などを記述することができます。

UNIXのコマンド言語インタープリター(シェル)にはいくつかあることはすでに述べました。標準のシェルはsh(ボーンシェル)で、システムの立ちあげなどの手続きはshで書かれています。システム管理を行なう人はshを知っていなければなりませんが、自分で書いて自分で使うにはcshを知っていれば十分です。ここではcshのスクリプトの書き方について説明します。

6.2.1.シェルスクリプトの文法

まずコメントは#で始まり、行末までです。普通にcshからスクリプトを呼び出すときはなくてもよいのですが、第一行目はそのシェルスクリプトをどのシェルを使って実行するかを教えるために使う場合があります。例えばUNIXの標準シェルであるshを使うアプリケーションがcshで書かれたスクリプトを実行する場合、そのままではshのコマンドと解釈するためシンタックスエラーになります。しかし第一行目に次のように書いておけばshがcshを呼び出してスクリプトを実行するため正常に働きます。

```
#! /bin/csh
```

UNIXの場合色々なシェルが使われるために導入された工夫です。誰から呼び出されるかわからない汎用的なスクリプトはこれに従うべきです。他人に見せるときもcshで書いてあるのだということが一目瞭然です。個人で使う分には必要ないでしょう。

スクリプトにおいても変数を使います。すでにシェル変数や環境変数については説明しました。参照は両方とも同じ$を使います。変数名の区切りを明確にするためには変数名を{}でくくります。シェル変数と環境変数は文法上は区別がつきませんので通常シェル変数は小文字の、環境変数は大文字の名前を付けます。変数の定義の仕方はこの二つで違います。

```
# シェル変数の定義
set testvar = test
# 環境変数の定義
setenv TESTVAR test
```

スクリプトの中で制御のために使う変数はシェル変数を使うべきです。環境変数はシェルが呼び出したプロセスに渡されますが、シェル変数は渡されません。シェルスクリプトからさらにシェルスクリプトを呼び出すときには環境変数を使う必要があります。シェル変数に値を与える方法にはもう一つ@があります。setは右辺の文字列が文字列として代入されるのに対して、@では右辺が値として評価されその結果が代入されます。次をやってみてください。

```
% set testcount = 1
% echo $testcount
1
% set testcount = ( $testcount + 1 )
% echo $testcount
1 + 1
% @ testcount = $testcount + 1
% echo $testcount
3
%
```

setでは+ 1が文字列としてそのまま変数に代入されているのに対し、(その意味では+は文字であって演算子ではありません。)@では右辺が式として評価されその演算結果が代入されています。

式の値の評価は繰り返しや条件分岐に使われます。繰り返しはwhileです。

```
# whileを使う
@ testcount = 0
while( $testcount < 10 )
    echo $testcount
    @ testcount = $testcount + 2
end
```

これを実行すると端末に0から8まで2おきに数字を表示します。whileの括弧の中はC

言語とほぼ同じ演算子が使えます。条件分岐にはifやswitchがあります。ifの使い方は

```
# ifを使う
if( $#argv < 1 ) then
    echo "using default file for input"
    myana default.dat
else if( -e $1 ) then
    echo "using $1 for input"
    myana $1
else
    echo "input file $1 not found"
endif
```

if then、else、endifの構造は説明の必要はないでしょう。式の表現には初めて出てくるものがあります。$#argvは、argvという変数に含まれる文字列の数を表します。argvは特別のシェル変数で、コマンド文字列に与えられた引数の集合を表します。C言語のプログラミングのところで解説したものと同じargvです。このスクリプトが引数無しで呼び出されると$#argvは0です。この時はdefaultを使うと言っています。次の$1は第1引数です。$argv[1]と同じ意味です。$は変数参照でした。argvという変数を参照します。[1]はその変数の中の1番目のワード(文字列)を表します。この例では引数としてファイル名を想定しています。ところがifの括弧の中は論理式を想定しています。単に$1を書いただけではうまく行きません。そこで-eという演算子を用います。その後の名前のファイルが存在すれば真です。この種の演算子はよく使います。まとめておきましょう。

-e filename	ファイルが存在すれば真
-z filename	ファイルの寸法が0(空)なら真
-d filename	ファイルがディレクトリであれば真
-f filename	ファイルが通常のファイルであれば真
-o filename	ユーザがそのファイルの持ち主であれば真
-r filename	ユーザがそのファイルを読むことを許されていれば真
-w filename	ユーザがそのファイルに書き込みを許されていれば真
-x filename	ユーザがそのファイルの実行を許されていれば真

条件分岐には他にswitchがあります。これもC言語のswitch文と同じ使い方をしま

す。

```
# switchを使う
while( 1 )
    echo -n "command: "
    set cmd = $<
    switch( $cmd )
    case quit:
          echo "exit from procedure"
          break;
    case help:
          echo "how to use this procedure..."
          breaksw
    default:
          echo "command $cmd not found"
          breaksw
    endsw
end
```

この例ではswitchがwhileループの中で使われています。まずwhile(1)は常に真ですから無限ループになります。つぎのecho -nは文字列command:を改行せずに表示します。コマンドプロンプトとして使っています。次のsetではcmdと言う変数に$<という特殊な変数の持つ値を代入します。$<とは標準入力から一行取り込むというものでスクリプトの中でユーザのキーボード入力を受け付けるために用意されています。さて、やっとswitchまできました。endswまでの間が対象になります。cmdの値がcaseの次のラベルと比較され一致すればそこから実行します。どれにも一致しないときはdefault:から実行します。quitと一致したときはechoした後breakしています。これはwhileの無限ループから抜け出すことを意味します。switchの範囲に関する命令ではありません。一方helpのときはbreakswしています。これはswitchの範囲から抜け出てendswの次に飛ぶことを意味します。

繰り返しには他にforeachという命令もあります。

```
# foreachを使う
foreach input ( test.dat run003.dat run023.dat )
    echo "Try to analyze dataset ${input}"
    myana $input
```

```
end
```

この例では変数inputにtest.dat、run003.dat、run023.datという文字列が順に代入されendまでが繰り返されます。""の中でも$は変数参照として認識されます。foreachの括弧の中にワイルドカードを使うこともできます。上の例でforeachの()の部分を(*.dat)とすると、現在のディレクトリの中にあるすべての.datファイルについて繰り返されます。

ここまでの例でおおかたの仕事はこなすことができると思います。あと必要と思われるのはsleepでしょう。プロセスの実行を指定した秒数だけ休みます。10秒休むには

```
sleep 10
```

です。状態表示を繰り返したいときや会議の時間の予告など使い道は色々あります。次をやってみましょう。

```
% ( sleep 300; echo "会議だよ!" ) &
[1] 21044
%
```

()の中は一つのプロセスとして扱われます。;は命令を順番に実行することを指定します。まずsleepが実行されそれが完了するとechoが実行されます。パイプ|とは動作が異なります。最後の&は()内のプロセスをバックグラウンドで実行せよという意味です。5分経つと端末に「会議だよ！」と表示してくれます。自分が実行させたバックグラウンドジョブが完了するのを待つwaitも覚えておいてください。

6.2.2.シェルスクリプトの使い方

さてここまででシェルスクリプトの書き方はわかりました。それを使うにはどうしたらいいのでしょうか。例えばmyprocというファイルをviで書いたとします。それを実行するには単にmyprocでよいでしょうか。できればそうしたいものです。やってみましょう。

```
% vi myproc
% myproc
myproc: Permission denied.
%
```

どうしたことでしょう。自分のファイルなのに許可がないとは。じつはこれは実行許可が設定されていないということです。UNIXのファイルにはどういう人にそのファイルをアクセスさせるかを設定するビットが用意されています。ls -lで見ることができます。

```
% ls -l myproc
-rw-r--r--  1 iwashi      293 Sep 21 15:16 myproc
%
```

この最初の記号-rw-r--r--がそれを表しています。最初の-は普通のファイルを表しています。そこがdならばディレクトリ、lならばシンボリックリンクを表します。そこから後の9つは、所有者、グループ、その他の人の読みだし、書き込み、実行の許可を表しています。最初の3つ、rw-は所有者はrつまり読みだしとw書き込みは設定されていますが、最後の実行xは設定されていません。同様にグループ、その他ともr--です。読みだしのみが許されています。実行可能なスクリプトですからxも付けたいと思います。chmodコマンドを使います。

```
% chmod 755 myproc
% ls -l myproc
-rwxr-xr-x  1 iwashi      293 Sep 21 15:16 myproc
%
```

chmodコマンドの最初の引数は上述の許可ビットを8進数で表現したものです。次のlsで見たとおりの数値になっています。その次の引数が許可すべきファイルです。これでmyprocは実行可能となりました。

```
% myproc
私のスクリプトを実行中です。
%
```

実は実行許可がなくても実行させる方法はあります。sourceを使います。

```
% chmod 644 myproc  # 実行許可を外しました。
% myproc
myproc: Permission denied.
```

```
% source myproc
私のスクリプトを実行中です。
%
```

実はこの説明はちょっと不正確です。じつはシェルスクリプトとして走らせるのとsourceコマンドを実行するのは意味が違っています。シェルスクリプトはシェルの子プロセスとして実行されるのに対して、sourceコマンドはシェルプロセス自身が実行します。そのため環境変数の定義などをする場合スクリプトとして実行させても子プロセスの中で定義されるために、親プロセスであるシェルには反映しません。sourceコマンドを使えば反映します。環境変数やアリアスなどを設定するコマンドを集めてファイルを作る場合はsourceコマンドで呼び出す必要があるというわけです。

便利なスクリプトができるといろんなところからそれを使いたくなります。スクリプトもコマンドと同じ検索方法で探されます。ですからpathに書かれているディレクトリのどこかにあれば呼び出せます。例えば~/binというディレクトリを作ってpathに加えておきたいものです。ログインするたびにちゃんとpathにこの名前が設定されるようにするにはどうしたらよいでしょうか。次に説明する.loginなどを使います。

6.3..loginと.cshrc

ユーザが直接呼び出さなくてもシステムやコマンドが呼び出すシェルスクリプトがいくつかあります。そのうちユーザに関係の深いものは.cshrcと.loginでしょう。どちらも自分のホームディレクトリに作ります。

.cshrcは、cshが起動されるたびに実行されます。cshrcのrcはrun commandで、最初に走らせる手続きを表します。なぜそう呼ばれているかは歴史をひもとかなければわかりません。UNIXが立ち上がるときに走る手続きは/etc/rcですから、由緒ある名前なのでしょう。.cshrcは起動毎に呼び出されますから、シェル変数の初期設定はここでやっておく必要があります。プロセスの起動の時環境変数は渡されますがシェル変数は渡されないからです。シェル変数のいくつかは環境変数からコピーして使われますから、環境変数に同じ名前のあるシェル変数の初期設定は必要ありません。一方.loginはログインシェルの中でのみ実行されます。ログインシェルとはユーザが自分で走らせたシェルでないシェルのことです。もともと端末はシステムが管理しているのでユーザがログインしようとするとシステムはそのユーザのためにシェルを起動してやらなければなりません。この時そのシェルプロセスの生みの親はシステムということになります。正確にはinitというプロセスがそれをやってくれます。そこで作られたシェルプロセスをログインシェルと言っています。ログインシェルでは親から引き継ぐ環境がな

いため自分で設定する必要があります。そこで.loginが登場します。いったんログインシェルが出来るとあとはそこから必要なプロセスがどんどん作られますが、その時はユーザ個人の環境が出来上がっていますからそれが継承されていきます。.loginは環境変数の設定や端末の設定など一度だけしておけばよいものを記述します。.logoutというファイルもログインシェル終了時に呼び出されます。