# 2.UNIXと対話する。

UNIXの機械にログインすることができるようになりました。%というプロンプトが表示されてコマンドを受け付けてくれます。このときコマンド言語通訳(CLI)プログラムがあなたの入力した文字列を解読してくれます。UNIXではCLIプログラムをシェルと呼んでいます。UNIXシステムをコアだとしてそれを包んで人間とのインターフェースをしてくれるプログラムなのでそう呼ばれています。UNIXにはいくつかのシェルが用意されています。通常はcshあるいはそれと互換のtcshが使われます。%というプロンプトはcshが動いていますという表示でもあります。本書でもcshを使います。

ユーザからみると区別がつきにくいのですが、コマンドはシェルに対する命令と、シェルによって起動されるプログラムとに分けられます。シェルはまず自分に対する命令であるかどうかを判断し、そうでなければその名前のプログラムファイルを探しに行き見つければ実行します。その探し方については６章で説明しましょう。

シェルにはさまざまな機能が組み込まれています。それについては後でまたもどることにしてとにかくUNIXを使ってみましょう。

## 2.1.UNIXの命令を使う。

UNIXも他のOSのように階層化されたファイルシステムを持っていますが、シンプルです。ディスクごとにファイルシステムがあるわけではありません。すべてのディレクトリーは/の下に作られます。この/のことをルートと呼びます。ユーザがログインするとそのユーザのホームディレクトリーが現在の作業場所になります。現在の作業場所のことをCurrent Working Directory(CWD)と呼びます。CWDを知る命令はpwdです。

```
% pwd                  ←現在の作業場所を知る命令
/home/iwashi          ←iwashiさんはログインしたところなので
%                       自分のホームディレクトリーにいる。
```

/home/iwashiというのは/の下のhomeというサブディレクトリーのさらに下のiwashiというサブディレクトリーを表します。左端の/はルートを表します。ルートというのはUNIXのファイルシステムの頂点を表すもので、ディスクのほかあらゆる装置がその下で管理されます。それより右の/はディレクトリーのつながりを表す文字として使われます。VMSの[]のなかの.や、MSDOSの\と同じです。CWDは.で表されます。CWDの一つ上のディレクトリーを..で表します。ディレクトリーの指定を/では

じめたときは絶対位置の指定、.やほかの名前ではじめたときは相対位置の指定をしたことになります。ユーザのホームディレクトリーは~でも表されます。今の例では~は/home/iwashiと同じです。他のユーザのホームを指すときは例えば~sanmaと書きます。/home/sanmaの意味になります。(システムによっては/homeの代わりに/usr/usersなどが使われる場合もあります。)

そのディレクトリーの中にどの様なファイルがあるか知る命令はlsです。listの 2 文字をとったものです。

```
% ls                    ←そのディレクトリーの中を調べる。
%                       ←何もないのですぐ次のプロンプトがでる。
```

他のディレクトリーもlsで見ることができます。たとえばルートディレクトリーをみるには

```
% ls /                                          ←ルートの中を見る。
bin         etc         tmp         var        ←いろいろはいっている。
dev         sys         usr         vmunix
%
```

自分のホームには何もないので面白くありません。これから作業するのですから自分のホームの下に作業用のサブディレクトリーを切りましょう。mkdirを使います。makedirectoryの意味です。

```
% mkdir doc             ←自分のホームの下にdocというサブディレクトリー
% ls                    ←lsで確かめる。
doc                     ←docはファイルではなくてサブディレクトリー
% ls doc                ←docの中を見てみる。
%                       ←空であった。
```

CWDをdocにかえましょう。cdを使います。change directoryです。

```
% cd doc                ←ホームの下のdocに移ります。
% pwd                   ←確かめてみましょう。
/home/iwashi/doc        ←移っています。
% cd                    ←引き数無しにcdするとホームに戻ります。
```

```
% pwd
/home/iwashi
%
```

UNIXでは名前が.で始まるファイルやディレクトリは特別な意味を持っています。しかもlsでは表示されません。ls -aとやると.で始まるファイルも表示されます。試してみてください。

## 2.2.命令を探す。

UNIXらしい雰囲気を味わってもらえたと思います。あとはどのような命令が使えるかを知ればよいだけです。さきに述べたように非常に多くの命令が用意されているので、探し方を知っておく必要があります。マニュアルを見ればよいのですが手元にない場合も多いのでオンラインで知る方法を覚えておきましょう。

### 2.2.1.どのような命令があるかを知る。

さきに述べたように命令にはシェルに対するものとプログラム起動によるものがあります。シェルに対するものは、man命令を使って、(manualの意味です。)

```
% man csh                ←cshに関するマニュアルを表示する。
```

で表示される膨大なメッセージを根気良く読めばわかります。(これでは本書の入門の意味がないか？)　manはマニュアルページを表示させる命令です。manそのものについて見たいときは

```
% man man
```

で良いわけです。manを実行すると１画面分表示すると最下行に-more-と表示して止ります。ここでSPACEを押すと次のページを表示します。RETURNを押すと１行追加表示します。qを押すとmanから抜けます。

プログラム起動の場合はどこからファイルが探し出されるかを知っていればいいことになります。シェルがそれを調べるわけですから、シェルに記憶されているはずです。実際にはシェル変数というシェルが使うデータベースの一つpathという変数に記憶されています。シェル変数を調べてみましょう。setという命令を使います。これはシェ

ルに対する命令です。

```
% set
path        = /bin /usr/bin /etc
prompt      = %
...
%
```

おわかりのようにpathという変数にある/binというディレクトリーから命令を探します。見つからないときはつぎに/usr/binを探し、それでも見つからないときは/etcを探します。変数pathに含まれているディレクトリーをlsで調べれば使える命令がわかることになります。ただ、単にlsとやっただけでは先の例にもあるように命令として実行できるプログラムファイルかそのほかのファイルかそれともディレクトリーなのか区別がつきません。そこで、

```
% ls -F /bin
```

と、-Fフラグを付けてls命令を実行します。これはファイルの属性を名前の後の 1 文字で表すよう指定するものです。実行可能なファイルには*が、ディレクトリーには/が、それ以外には何もつきません。( @が付いたものが出てくることがありますが、そ

manページを表示させると項目に () で囲まれた1から8までの数字が表示されます。それぞれの数字は次のような分類(セクションと呼びます)を表しています。ですから (1) のついた項目はユーザ・コマンドであるということになります。なおそれぞれのセクションにintroというエントリーがあることがあります。man 数字 introとやってみてください。そのセクションがどのような情報を持っているか見ることができます。

| セクション | 内容 |
|---|---|
| 1 | ユーザ・コマンド |
| 2 | システムコール |
| 3 | ライブラリルーチン |
| 4 | デバイス用スペシャルファイル |
| 5 | 主なシステムファイル |
| 6 | ゲームやデモプログラム |
| 7 | パブリックファイル |
| 8 | システムの保守・管理用コマンド、デーモン |

れは後のお楽しみ。）これで実行可能なファイルすなわち命令を見つけることができます。次に知りたいのはその命令が何をしてくれるか、どのように使えば良いかです。それは先ほどのmanを使います。manがマニュアルページを表示してくれない実行可能ファイルがあるかもしれませんが、当面はそれは命令ではないと思っておいて下さい。

## 2.2.2.よく使うコマンド

必要なコマンドは自分で探しなさいというのはあまりに不親切なのでよく使うコマンドのリストを上げておきます。詳しくはmanで調べてください。具体的なコマンドについては今後の各章で取り上げていきますのでここで覚える必要はありません。なお本書では導入されたばかりの機械を想定して解説しています。使い込まれたシステムでは同じことをするのにはるかに便利なコマンドが用意されていることもあります。まわりの人から「こっちのコマンドの方が便利だよ」とか指摘されたときはそっちを試してみることをお勧めします。

| | |
|---|---|
| alias | コマンドの別名 |
| ar | アーカイバもしくはライブラリアン |
| bg | プロセスをバックグラウンドに |
| cat | ファイル内容表示 |
| cc | Cコンパイラ |
| cd | 作業ディレクトリ変更 |
| chmod | ファイル属性変更 |
| chown | ファイルオーナー変更 |
| compress | ファイル圧縮 |
| cp | コピー |
| dbx | ソースレベルデバッガー |
| df | ディスク使用状況 |
| du | ディレクトリが使っているディスク寸法 |
| echo | 引数を標準出力に表示 |
| emacs | フルスクリーンエディタ |
| ex | 行編集エディタ |
| exit | ログアウト |
| f77 | FORTRANコンパイラ |
| file | ファイル属性表示 |
| find | ファイルを探す |
| finger | ユーザの消息 |

| | |
|---|---|
| ftp | ネットワーク間でのファイル転送 |
| grep | 文字列検索 |
| head | ファイルの頭数行を表示 |
| history | ヒストリー |
| kill | プロセスを殺す |
| ld | リンカー |
| ln | シンボリックリンクを張る |
| lpq | 印刷待ち行列の状態表示 |
| lpr | 印刷 |
| lprm | 印刷待ち行列からジョブを取り除く |
| ls | ファイル表示 |
| mail | 電子メール |
| make | プログラム管理 |
| man | マニュアルページ表示 |
| mkdir | ディレクトリ作成 |
| more | ページ単位の表示 |
| mv | ファイルの移動もしくは改名 |
| passwd | パスワード変更 |
| printenv | 環境変数表示 |
| ps | プロセスステータス |
| pwd | 現在の作業ディレクトリ表示 |
| ranlib | ランダムライブラリー作成 |
| rm | ファイルの消去 |
| rmdir | ディレクトリ削除 |
| sed | ストリームエディタ |
| set | シェル変数設定・表示 |
| setenv | 環境変数設定 |
| stty | 端末機能設定 |
| tail | ファイルの末尾数行を表示 |
| talk | ユーザ同士の対話 |
| tar | テープアーカイバ |
| telnet | よその計算機へのログイン |
| touch | ファイル日付変更 |
| uncompress | 圧縮ファイル復元 |
| vi | スクリーンエディタ |
| whereis | コマンドやmanページのある場所を表示 |

| | |
|---|---|
| `which` | 呼び出されるコマンドの位置 |
| `who` | 今いるユーザ表示 |

# 2.3.シェルを使う。

UNIXを使いこなすにはシェルの機能を十分に理解する必要があります。非常に少ないタイピングでやりたいことが出来るようになると能率が格段によくなります。ここではシェルの基本的な使い方を説明します。第 6 章でさらに詳しい説明をします。

## 2.3.1.別名(アリアス)

実行可能なファイルを探すのに毎回`ls -F`とタイプするのは面倒だと思いますね。そこで

```
% alias ls 'ls -F'
```

とやります。以後`ls`とタイプすると別名の表が参照されて`ls -F`と置き換えられます。これで`ls`にいちいち`-F`を付ける必要がなくなりました。例えばVMSやDOSのユーザでついつい`ls`ではなくて`dir`とやってしまう人は

```
% alias dir ls
```

とやれば`dir`がコマンドとして使えるようになります。現在どの様な別名が定義されているかは、

```
% alias
ls          (ls -F)
dir         ls
%
```

で見ることができます。まちがえて定義したりして消したいときは、

```
% unalias dir
```

で別名表から削除できます。別名はログインしたときに自動的に登録しておくと便利で

す。VMSのlogin.comやDOSのautoexec.batのような機能も当然有ります。6.3.節を参照して下さい。

## 2.3.2.ヒストリー

場合によっては同じコマンド文字列を繰り返しタイプしなければならないことがあるかもしれません。以前にタイプした命令文字列を呼び出せれば便利です。これをヒストリー機能と呼んでいます。試しに次をやってみましょう。

```
% history
     1  pwd
     2  ls
     3  vi test.c
     4  cc -o test -I~/include test.c
     5  ./test
     6  history
%
```

%のプロンプトに対して入力実行された命令のことをイベントと呼んでいます。historyコマンドは過去に実行された一定の数のイベントを表示します。この例ではhistoryコマンドも含め6つのイベントが実行されています。例えばイベント番号4はcコンパイラを呼び出しています。命令自身の説明は後で出てきますのでここでは触れませんが、このように長い命令をタイプし直すのは面倒です。この例のようにイベント3でソースファイルを編集しイベント4でコンパイルして5で実行するというパターンはよく繰り返されます。そこでいま作ったtestを実行してバグを見つけました。test.cを編集し直します。その時は次のようにしてイベント3を呼び出します。

```
% !3
vi test.c
```

!のあとに番号を書くとその番号のイベントが呼び出されます。同様に!4でイベント4が呼び出されます。

```
% !4
cc -o test -I~/include test.c
```

ただしこれはイベント番号を覚えていなければならないのでそれほど使いよくありません。それよりも次のようにやったほうが簡単です。

```
% !v
vi test.c
% !c
cc -o test -I~/include test.c
%
```

!vは文字vで始まる最近のイベントを呼び出します。!viでもおなじです。!cは同様にcコンパイラ起動のイベントを呼び出します。直前のイベントは!!で呼び出せます。!-2は２つ前のイベントを呼び出します。!?str?は文字列strを含んだコマンドを持つイベントを探します。この場合strはコマンド名の途中にあらわれるものでもよいわけです。

　ヒストリーで呼び出せるコマンドの一部を修正して実行したいこともよくあります。タイプミスをしたり、一連のコマンドを順次実行するときなどです。この場合はイベント呼び出しの後に続けて修正を指示します。上の例では

```
% !v:s/test/demo/
vi demo.c
```

sは置き換えsubstituteを表します。/にはさまれた部分が置換されます。/を含んだ文字列を置換したいときはとにかくsの次の文字が分離文字として扱われますのでなんでも文字列に出てこない文字を使えばよいわけです。SUNでは直前のヒストリーの置換を^old^new^でできるようになっています。これは!!:s/old/new/と同じです。次の場合はどうでしょうか、

```
% !c:s/test/demo/
cc -o demo -I~/include test.c
```

最初のtestだけが置換されてしまいました。これでは困ります。すべてマッチする部分を置換したいときはgをつけて、

```
% !c:gs/test/demo/
cc -o demo -I~/include demo.c
```

本当に正しく置き換えられているのか確かめたいときは:の後にpを付けます。こうすると表示されるだけで実行はされません。ヒストリーには残るので確かめたうえで!!呼び出せます。

## 2.3.3.リダイレクトとパイプ

UNIXの命令が実行される場合３つの標準的な入出力装置が使われます。特に指定しなければもちろん入力はキーボードで、出力は標準出力とエラー出力のふたつがスクリーンに割り当てられます。場合によっては命令の出力をファイルに残したいことがあります。このときは出力をリダイレクトします。>を使って、

```
% ls /bin > filelist
%
```

この場合は標準出力はスクリーンではなくてfilelistという名前のファイルが作られそこに書き込まれます。さらにfilelistに追加して書き込みたいときは

```
% ls /usr/bin >> filelist
%
```

この場合filelistというファイルが存在すればそれに標準出力が追加されます。同様に入力もリダイレクトができます。この時の文字は<です。lprという命令を引き数無しで使うと標準入力をプリンターに出力してくれます。

```
% lpr < filelist
%
```

ただしこれは% lpr filelistとやるのと効果は同じです。以上の例は入出力のリダイレクトでしたが、ある命令の標準出力を別の命令の標準入力に直接つなぐことができます。文字|はその左側の命令の標準出力をその右側の命令の標準入力として使うことを指定します。これをパイプと呼びます。

```
% ls /bin /usr/bin | lpr
```

この場合中間的なファイルは作られずふたつの命令lsとlprが同時に起動されます。マルチタスクシステムですからこういうことができるわけです。(MSDOSのパイプ|は中

間ファイルをつくります。シングルタスクだからです。)
　標準入力からデータを読み込み加工して標準出力に書き出すだけのプログラムがたくさんあります。それらをフィルタープログラムと呼びます。例えば

```
% cat README | jistosj | lpr
```

　JISコードで書かれた日本語のファイルREADMEを、シフトJISコードしか受け付けないプリンターに打ち出すためにjistosjというフィルターを使っています。ここでcat命令を使いました。catはファイルの内容を標準出力に連鎖する(concatenate)命令で通常はファイルの中身をスクリーンに表示するVMSやDOSのTYPEと同じ意味で使われます。

## 2.3.4.プロセス

　パイプの所でマルチタスクについて少し触れました。UNIXの場合命令を実行するということは命令を実行するプログラムを一つのプロセスとしてシェルプログラムが生成起動するということです。通常は命令が実行されている間キーボードやスクリーンへの入出力は命令実行プログラムが占有して、シェルは休んでいます。キーボードを占有しているプロセスをフォアグラウンドプロセスと呼びます。例えばスクリーンのどこかに時刻を表示する命令clockがあったとします。このプロセスはキーボード入力を必要としませんが、走らせておくと便利です。このような時にはバックグラウンドで走らせることができます。( 実際にはSUNではclock、ほかのXウインドウシステムではxclockやoclockなど )

```
% clock &
%
```

　このプロセスはキーボードを占有しないバックグラウンドプロセスですからすぐに%が表示されますが、時計は表示されたまま時を刻んで行きます。命令の後に&をつけると、その命令をバックグラウンドとして走らせるようにとの意味です。うっかりしてこのclockを&無しで走らせてしまいました。このときclockはフォアグラウンドとなりキーボードを占有してしまいます。

```
% clock
```

こういうときはコントロールキーを押しながらyを押します。( 以下ctrl+yと書きま

す。 ) clockプロセスは中断され、

```
^Y
Stopped
%
```

プロンプト%が出てきました。clockがまだ生きていることを確認しましょう。

```
% jobs
[1]  + Stopped                 clock
%
```

このプロセスをバックグラウンドに切り替えます。bgコマンドを使います。clockプロセスには1というジョブ番号が割り振られています。コマンド文字列の中で%1はジョブ番号 1 番のプロセスを表します。

```
% bg %1
[1]    clock &
% jobs
[1]    Running                 clock
%
```

必要がなくなって止めたいときはkillします。

```
% kill %1
```

先程ctrl+yの例が出てきましたが、シェルにはいくつかのエスケープキーが定義されています。ある命令を実行中に

| | |
|---|---|
| ctrl+c | フォアグラウンドの強制終了。 |
| ctrl+d | 入力の終了。%に対しこのキーを押すとシェルが終わってしまう。 |
| ctrl+y | フォアグラウンドの強制中断。 |
| ctrl+z | 入力の中断。 |

によってシェルを呼び出すことができます。

jobsでは自分が現在ログインしている端末から実行したプロセスが表示されます。

実際にはそれ以外にも様々なプロセスが走っています。psはそれらを表示するコマンドです。process statusです。psではジョブ番号の替わりにpid(プロセスID)でプロセスが識別されています。何かの事情で走りっぱなしになっているプロセスができてもjobsで見ることができないことがあります。別のログインセッションで起動されたプロセスなどはそうなります。それらを止めたいときはpsで見てpidを確かめ、killでジョブ番号の替わりにpidそのものを(%なしで)引数にして殺します。

```
% ps x
  PID TT STAT  TIME COMMAND
19350 co IW    0:00 -csh (csh)
19469 co S     0:03 clock
19530 co R     0:00 ps
% kill 19469
%
```

psではオプションを表す-は省略できます。この例の中のxは-xと同じです。バックグラウンドで走っているプロセスも表示させるスイッチです。いらなくなったclockをkillします。

UNIXの場合でもプロセスの実行優先度は設定されます。ps lでプロセスの詳細を見てみましょう。

```
% ps l
      F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN    STAT TT  TIME COMMAND
20008201 100 21453 21449  1  15 0  88  328 kernelma S    p0  0:00 -bin/csh
20000001 100 21514 21453 20  30 0 184  416          R    p0  0:00 ps l
```

色々なデータが表示されていますが、優先度に関係するのはCP、PRIとNIです。UNIXは非常に平等なシステムなのですべてのプロセスが同じだけCPUにありつけるように優先度を計算します。CPは最近のCPU利用率でこれが多いと優先度を下げられます。PRIが実際の優先度を表します。これだけでは困ることもあります。UNIXの場合はバッチジョブの概念がありません。CPUは食うけれども時間がかかっても構わないプロセスはユーザが自分で優先度を下げる責任があります。そうでないとそのプロセスも一人前の顔をしてCPUを要求してきます。そうすると端末の応答が悪くなったりして人に迷惑をかけることになります。意識的に優先度を下げるのがniceです。NIがその値で、この数値は大きければ大きいほど優先度を下げます。自分自身のナイス値を増やすには

```
% nice +3
```

とやります。ナイス値は0から19までの値を取ることが出来ます。あるコマンドを特定のナイス値で実行するには

```
% nice +5 ps l
```

この場合ps lがナイス値5で実行されます。実際にやってみてどこが変わったか見つけてみましょう。自分ののナイス値より小さい値をバックグラウンドプロセスに設定することは出来ません。一旦増やしたナイス値を減らすことも出来ません。

## 2.3.5.その他

最後になってしまいましたが、シェルに命令を与えるときの引数の書き方にはルールがあります。空白で区切られた文字列一つ一つが引数として扱われます。それゆえ空白を含んだ文字列を引数として渡したいときは"か'で囲んで渡します。先程のaliasに例が出ています。cシェルでは`(バッククォート)は別の働きをします。`に囲まれた部分が先にシェルによって実行されその標準出力が文字列として採用されます。例えば、

```
% hostname
sakanaya
% set prompt = "`hostname`% "
sakanaya%
```

hostnameというのはその計算機の名前を表示する命令です。今の場合はsakanayaが標準出力に出ました。次にset promptはシェル変数promptを設定する命令です。""の中に``で囲まれた部分があります。まずこれが実行されます。sakanayaが出力でしたから``の部分がこれに置き換えられます。その結果set prompt = "sakanaya% "というようになります。prompt変数が変更されたので次からはsakanaya%というプロンプトが表示されるようになりました。

引数にはワイルドカードが使えます。ワイルドカードは現在のディレクトリの内容で展開されます。これはシェルが勝手に書き換えるので例えば*という文字をコマンドに与えたいときは'か"で囲む必要があります。ワイルドカードには(正規表現といいますが)長さ0以上の任意の文字列を表す*、任意の一文字を表す?のほか、[a,e,h]とやると[]の中のいずれか、[i-n]とやるとiからnまでのいずれかの1文字という意味になります。^は否定を表します。^aはaという文字以外という意味になります。

継続行は行末に\をタイプします。