知能情報工学演習 I

柳本 豪一

平成 17 年 5 月 25 日

ここでは、シェルについて説明します。シェルは、UNIX(OS) とユーザの間に存在する窓口的な役割 (インターフェイス) を提供するプログラムです。さまざまな設定が行え、適切に設定することで、さらに使いやすい環境を構築することができます。

1 シェルとは

シェルは、ユーザが入力したコマンドを管理し、ユーザに使いやすい環境を提供するプログラムです。一見 UNIX そのもののように思えるかもしれませんが、シェルと UNIX 本体とは異なるものです。例えば、UNIX を変更せずに、今使っているシェルを他のシェルに変更することができることからも、UNIX 自体とシェルは違うものだということが分かります。では、シェルの役割を理解するため、もう少し詳しく UNIX の構成について勉強していきましょう。

UNIX は、**カーネル**と呼ばれる基本機能を提供するプログラムと、多くのアプリケーションと呼ばれるプログラムから構成されています。今回説明するシェルもアプリケーションの一つです。カーネルは、コンピュータのハードウェアの基礎的な機能を扱えるようにするプログラムです。例えば、単純なハードディスクへの読み込みや書き込み、メモリの管理などがカーネルが担当する仕事です。カーネルは、ハードウェアと直接やりとりするプログラムであり、UNIX の中核となるプログラムです。しかし、カーネルをそのままユーザが使うとすると、非常に不便です。このため、ユーザが使いやすくなるように、さまざまなアプリケーションが開発されており、シェルは特にユーザが対話的にアプリケーションを実行できるようにするプログラムです。図 1 にシェルとカーネルの関係を表します。図に示すように、ユーザがプログラムを実行するときには、すべてシェルを介して操作します。このため、UNIX とユーザが一番接する機会が多いプログラムはシェルになります。

では、どのようなシェルが存在しているのでしょうか？現在よく使われているシェルは大きく 2 つに分けて、Bourne シェル系、C シェル系があります。

Bourne シェル系には、sh、bash、ksh があります。sh は、一番初期に作られたシンプルなシェルであり、どの UNIX システムにも必ずインストールされています。また、これから紹介するシェルの基礎となったシェルです。このため、UNIX の種類に関係なくインストールされており、シェルスクリプト (後で説明します) を作る場合には、sh をよく利用します。とくに、UNIX を起動する設定ファイルなどは sh を念頭において作られいます。bash や ksh は、sh を機能拡張したものであり、非常に強力なシェルとなっています。例えば、**ヒストリ機能**、**コマンド編集機能**、**補完機能**などが追加されています。しかし、bash や ksh はどの UNIX にもインストールされている訳ではなく、利用できるかできないかは、その UNIX を管理している人次第です。bash は、多くの Linux システムで**ログインシェル**として設定されている場合が多いです。ログインシェルとは、ユーザが

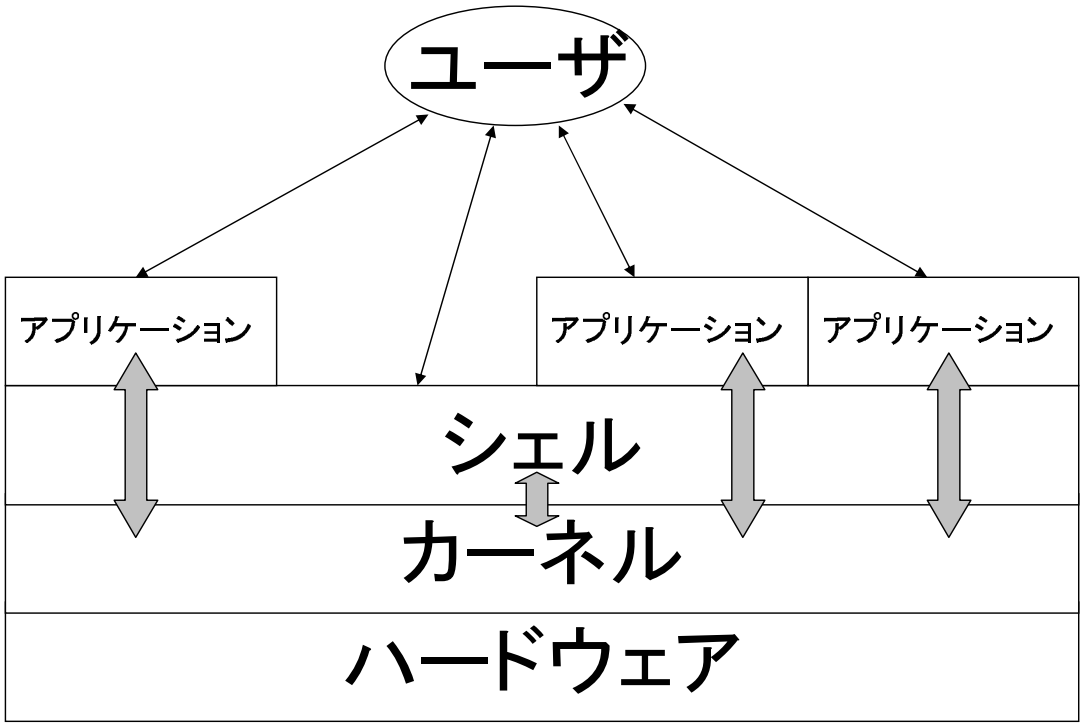

図 1: UNIX の構成

ログインしたときに起動されるシェルのことです。kterm を立ち上げたとき、実行されているシェルがログインシェルと考えて問題ありません。

C シェル系には、csh と tcsh があります。csh は、*C* **言語風の構文構造**、**ヒストリ機能**などが追加されています。sh に比べ、ユーザの操作が便利になっているので、多くのシステムで標準のログインシェルとして利用されています。しかし、最近では tcsh に置き換えられてきています。tcsh は、csh を拡張したものであり、強力な**補完機能**、**コマンド編集機能**が追加されています。サテライトホールのマシンでは、bash がログインシェルとなっています。

2 基本的な操作

ここでは、基本的な bash の使い方について説明していきます。

2.1 ログインシェルの確認

まずは自分が使っているシェルを確認しましょう。passwd ファイルで設定されている内容は、finger を用いて確認することができます。passwd ファイルには、自分のログイン名やログインシェルなどが登録されています。finger （ログイン名）と実行して表示されたメッセージ中に Shell と書かれている部分が自分のログインシェルとなります。おそらく、/bin/bash と表示されていると思います。これより、自分のログインシェルが bash であることが確認できると思います。

2.2 他のシェルの使い方

では、他のシェルを使うにはどうすれば良いのでしょうか？一時的にシェルを使う場合には、新しくシェルを起動します。今は bash が起動しているので、tcsh を起動する場合には tcsh と入力します。他にも sh、ksh、zsh が起動するので、いろいろ起動してください。終了は、C-d とすれば前のシェルに戻ります。

それでは次の説明に進むので、他のシェルを起動している人は、bash に戻ってから次に進んでください。

2.3 コマンド編集機能

では、コマンド編集機能について説明します。コマンド編集機能は、入力した文字を編集する機能です。コマンド編集機能は、XEmacs のキーバインドと同じなので、C-b で左にすすみ、C-f で右に進みます。C-p で前に入力したコマンド、C-n で次に入力したコマンドが表示されます。このように、保存されている過去に入力したコマンドのことを**ヒストリ**と呼びます。文字を削除する場合には、C-d を用います。

ここで、**ヒストリ**が出てきたので、少しだけ説明を加えておきます。ヒストリは、過去に入力したコマンドの履歴のことです。histroy と入力すると、現在保存されているヒストリの一覧が表示されます。ヒストリを利用してコマンドを実行する場合には、!(ヒストリ番号) で実行することができます。しかし、bash では、C-p により対話的にコマンドのヒストリをたどっていくことができるので、あまり使われなくなっています。

2.4 補完機能

bash の便利な機能として、補完機能があります。これは、実行可能なコマンド名や利用可能なファイル名を自動的に補完してくれます。補完機能は、入力中に TAB キーを押して実行します。候補が一つに絞れるときには、コマンド名やファイル名を表示しますが、複数の候補があるときには、途中までの補完となります。

補完機能は、非常に便利な機能なので、これからどんどん使っていってください。

2.5 メタキャラクタ

コマンドの引数として、実際のディレクトリやファイル名以外を用いることができます。例えば、すでに使っている「~」や「.」などがそれにあたります。このような文字を、**メタキャラクタ**と呼びます。grep で説明した、正規表現に似たようなものだと考えてください。しかし、同じ文字でも、メタキャラクタと正規表現では異なった意味で用いられるので少し注意が必要です。

すでに説明した「~」、「.」、「..」は説明から省きます。これらもシェルが提供するメタキャラクタの一つです。では、「*」、「?」、「[]」、「{}」について説明を行っていきます。

「*」は、正規表現で出てきた「*」に似ていますが、**任意の複数の文字**を表します。正規表現では、**直前の文字を *0* 回以上繰り返す**ことを表していたので、意味が異なっています。つまり、正規表現での「.*」に該当するものが、シェルの*となります。例えば、「ls n*」と入力すると、カレントディレクトリに含まれる n で始まるすべてのファイルが表示されます。もし、ディレクトリがマッチした場合には、ls の機能により、そのディレクトリの中身が表示されます。

「?」は、任意の 1 文字を表します。正規表現では、直前の文字の 0 回または 1 回の繰り返しを表すので、少し意味が異なります。正規表現での「.」に対応するメタキャラクタです。実際に使うときには、「ls ???????」と入力すると、7 文字のファイルが表示されます。

「[]」は、正規表現の「[]」と同じく、[] で囲まれた文字のうちの 1 文字を表します。例えば、「ls [le]*」と入力すると、l か e で始まるファイルが表示されます。正規表現と同じく、[] の中で「^」も使うことができます。「ls [^le]*」とすると、先ほど表示されたもの以外のファイルが表示されます。

「{}」は、カンマ (,) で区切られた文字列のどれか一つを表します。「[]」は [] で囲まれたどれか 1 文字ですが、「{}」は文字列を選択することができます。例えば、「ls *.{c,pl}」とすると、c または pl で終了するファイルが表示されます。

最後に、シェルでのメタキャラクタの扱い方について解説しておきます。シェルは、メタキャラクタを受け取ると、実際のファイル名に展開します。そして、その展開されたファイル名を引数としてコマンドを実行します。このため、「ls」と「ls *」と入力したときの動作が異なります。「ls」ではファイルとディレクトリが表示されますが、「ls *」ではディレクトリの中身が表示されてしまいます。メタキャラクタがどのように展開されたかを確認するためには、「echo」が利用できます。「echo *」と入力すると、メタキャラクタが展開された結果が表示されます。

それでは、メタキャラクタを利用してさまざまなファイルの指定の仕方を試してみてください。

2.6 プロセスとジョブ

UNIX では、すべてのプログラムは**プロセス**という形で管理されています。プロセスとは、UNIX が管理している処理の単位のことで、通常はプログラムと一対一対応しいると考えて問題はありません。

一方、シェルでは**ジョブ**という形で実行している処理を管理しています。プロセスは、UNIX 自体でのプログラムの管理方法で、ジョブはシェルでの管理方法という違いがあります。基本的には、プロセスとジョブは同じものを表していると考えて問題ありません。

それでは、ジョブの操作について説明をしていきます。ジョブの実行方法として、UNIX では 2 通りの方法が用意されています。それらの実行方法を、「**フォアグランド**」と「**バックグランド**」での実行と呼びます。フォアグランドでの実行は、今までの実行方法であり、ジョブを実行したシェルを占有して動作します。バックグランドでの実行は、ジョブを実行したシェルを占有せず、ユーザにそのシェルが利用できる形で動作します。このため、一つのシェルで、複数のジョブを実行することができます。バックグランドでジョブを実行するときは、コマンドの後ろに「&」を付けて実行します。それでは、フォアグランドとバックグランドでの実行の違いを見てみましょう。今までの方法で、kterm から XEmacs を立ち上げてください。XEmacs が立ち上がった後、XEmacs を立ち上げた kterm はコマンドを一切受け付けません。もしこの kterm を利用したい場合は、XEmacs を終了させなくてはなりません。では、起動している XEmacs を終了してください。kterm がコマンドを受け付けるようになったと思います。それでは、バックグランドで XEmacs を起動してみましょう。XEmacs を起動するときに、&を忘れないようにしてください。今度は、kterm は問題なく動作していると思います。

それでは、ジョブをフォアグランドかバックグランドで実行するには、起動時に指定するしかないのでしょうか。UNIX では、起動しているジョブを、フォアグランドからバックグランド、バックグランドからフォアグランドへ移すことができます。このコマンドが、「fg」と「bg」です。fg は、バックグランドで動作しているジョブをフォアグランドに移します。一方、bg はフォアグランドで動作しているジョブをバックグランドに移します。では、実際に利用してみましょう。フォア

グランドで XEmacs を起動してください。kterm は使えなくなっていると思います。この状況で、C-z と入力してください。C-z は、フォアグランドで実行しているジョブを一時停止するコマンドです。この状況で、kterm はコマンドを受け付けることができますが、XEmacs は、入力を受け付けない状況になっていると思います。ここで、XEmacs をバックグランドに移すために、bg と入力してください。XEmacs がバックグランドで実行されるようになり、kterm も XEmacs も使えるようになりました。ここで fg と入力すると、最初の状況に戻ります。

これまでで、ジョブをフォアグランドとバックグランドで実行する方法を知りました。これにより、1 つのシェルで複数のジョブを実行することが可能になりました。では、現在実行しているジョブをみるにはどうすればよいのでしょうか。現在シェルで実行しているジョブを見るには、jobs を用います。「jobs」と入力すると、現在実行しているジョブの一覧が表示されます。一方、プロセスは ps を用いて見ることができます。「ps」と入力すると、今 ps を実行したシェルで起動されたプロセスが表示されます。「aux」と言うオプションを付けて実行すると、現在の UNIX で動いているプロセスの一覧が表示されます。ジョブはシェルが管理しているのものなので、jobs を実行した以外のシェルのジョブの内容を見ることはできません。ここが、ps コマンドとの違いになります。それでは、ps と jobs を使って実行しているプロセスの一覧を表示してみましょう。

3 bash の設定

bash を設定することで、さらに使いやすい環境を構築することができます。ここでは、bash の設定について説明をします。

3.1 設定ファイル

UNIX の環境をカスタマイズするにために、**シェル変数**と**環境変数**と呼ばれる変数の値を設定します。シェル変数とは、シェルの動作を指定するための変数であり、シェルの動作を変更できます。環境変数とは、いろいろなアプリケーションから参照される変数のことで、アプリケーションの動作を指定するために設定を行います。このようなシェル変数や環境変数の値を、毎回ログインするとき自動的に設定するために、設定ファイルが用意されています。bash では、.bash_profile、.bashrc、.bash_logout が設定ファイルとして利用されます。.bash_profile はログインしたときに bash が読み込むファイルで、基本的な設定をするファイルとなります。.bashrc は、新しい bash が起動されるたびに読み込まれます。.bash_profile はログイン時に 1 回だけ読み込まれるという点で異なっています。利用者の設定は、基本的に.bashrc に書き込むことになります。.bash_logout は終了時に 1 度だけ読み込まれるファイルです。それでは、`~thy00057/.bashrc` を自分のホームディレクトリにコピーしましょう。そのあと、ホームディレクトリで `source .bashrc` を実行してください。source は変更した.bashrc を有効にするためのコマンドです。

以下で説明する内容を参考にして、設定ファイルを修正する場合には、必ずバックアップ (現在のファイルのコピー) を取るようにしましょう。.bashrc の設定が正しく行われていない場合、シェルが正常に動作しない場合があるので、必ず正常に動く.bashrc を残しておいてから、修正します。たとえば、修正前の.bashrc を.bashrc.org や.bashrc.old などの名前で保存するようにしましょう。また、これから説明するものは bash についてのシェル変数なので、bash 以外のシェルを利用する場合には異なるので、各自シェル変数を調べる必要があります。man bash でシェル変数を調べることができます。

3.2 シェル変数

それでは、シェル変数について説明を行っていきましょう。以下のものは代表的な組み込み変数なので、ほかにもいろいろあります。各自調べてみましょう。設定方法は、「シェル変数=値」です。

3.2.1 HISTSIZE

ヒストリに記憶されるコマンドの最大数を指定します。サテライトホールでは 1000 で設定されています。

3.2.2 PATH

シェルに入力されたプログラムを探索するディレクトリを指定します。UNIX では、PATH に設定されていないディレクトリは探索を行いません。そのため、よく使うプログラムは PATH が通っているディレクトリに置かなくてはいけません。現在設定されている PATH を表示するには、`printenv PATH` と入力します。

3.2.3 PS

シェルのプロンプトの設定を指定します。プロンプト設定でよく使う特殊文字列を説明します。

- `\W`
  カレントディレクトリの最終ディレクトリ名のみを表示する。
- `\w`
  カレントディレクトリを表示する。
- `\H`
  ホスト名を表示する。
- `\d`
  日付を表示する。
- `\s`
  シェルの名前を表示する。
- `\t`
  24 時間表示で時刻を表示する。
- `\T`
  12 時間表示で時刻を表示する。
- `\u`
  現在のユーザ名を表示する。
- `\!`
  現在のコマンドのヒストリ番号を表示する。

3.3 環境変数

代表的な環境変数を紹介します。環境変数とは他のプログラムからも参照できるようになっている変数の事です。通常のシェル変数は、シェルの設定のために利用され、他のプログラムからは参照する事ができません。設定方法は、「export 環境変数=値」です。ここで説明するもの以外にもたくさんあるので、各自調べてみましょう。

3.3.1 DISPLAY

X サーバが実際に描画するサーバ名を設定する環境変数です。他のマシンに ssh でリモートログインした場合、そのままでは画面の表示は、リモートログインしたマシンとなります。ここで、今自分が使っているマシンに画面を表示したいときに環境変数 DISPLAY を利用します。設定するときには、「export DISPLAY=マシン名:0.0」という形で設定します。

3.3.2 EDITOR

標準で利用するエディタを指定します。たとえば、less でファイルを閲覧しているときに、編集モードに入るときに起動されるエディタなどを指定します。

3.3.3 LANG

使用する言語を指定します。通常は、漢字コード EUC を利用するので、jp_JP.EUC と設定しておけば問題ありません。リモートログインしたときに、日本語が表示されない場合に設定を行えば解決することが多いです。

3.3.4 PAGER

標準で利用するページャを指定します。man などで、マニュアルを表示する際に利用しているものは、環境変数 PAGER で設定されたページャです。

3.4 エイリアス

bash では、コマンドに別名 (エイリアス) を付けることができます。エイリアスを使うと、よく使うコマンドとオプションを組み合わせて、自分専用のコマンドを作ることが可能になります。エイリアスは、「alias」を用いて設定します。使い方は、「alias 別名=' コマンド'」となります。

現在設定されているエイリアスを見るには、alias とそのまま入力します。画面上に、いくつか出てきたと思います。たとえば、rm と入力したとき、シェルは rm -i と解釈して動作しています。このため、rm がファイルを消してよいか確認するようになります。

それでは、エイリアスを用いて複雑なコマンドに簡単な別名を付けてみましょう。次回からも利用する場合には、エイリアスの設定を.bashrc に書き込めば有効となります。

4　シェルスクリプト

シェルスクリプトとは、一連の動作をファイルにまとめて一括に処理でるようにしたものです。基本はコマンドラインからの入力であり、入力した手順をそのままファイルに記述します。ここでは、csh を用いてシェルスクリプトを作成することにします。

4.1　シェルスクリプトのお約束

シェルスクリプトを作成するための、約束事をまとめます。ファイルの先頭は「#!/bin/csh」から始めます。これは、このファイルがシェルスクリプトであることを表すものなので、必ず忘れずに記述するようにしてください。

次に、ファイルの**実行許可**を与える必要があります。作成されたシェルスクリプトは、そのままでは実行されません。これは、新しく作成されたシェルスクリプトを UNIX が実行できないファイルだと思っているからです。このため、実行するために実行許可を与える必要があります。許可の設定は、「chmod」で行います。実行許可を与えるためには、「chmod +x ファイル名」とします。

ファイルの実行許可を知るには、「la -l」と入力し、ファイルの先頭に付いている「-rwxr-xr-x」という文字列で判断できます。r は読み込み可能を、w は書き込み可能を、x は実行可能を表します。chmod を実行する前に、ls -l で確認するとその違いが分かります。

chmod を使うと、ファイルの許可をさまざまに設定することができます。興味のある人は、man で chmod の動作を調べてみましょう。

4.2　文法

シェルスクリプトは、基本的に入力するコマンドをそのまま書けば作成できます。しかし、複雑な処理を行うため、いくつかの機能が追加されています。ここでは、シェルスクリプト特有の機能について説明します。

4.2.1　変数

変数名として利用できる文字は、アンダーバー (_) と英数字だけです。変数への値の代入は set を用いて行いますが、その値の中身を扱うときには、変数名のはじめに$を付ける必要があります。

それでは、次のシェルスクリプトを入力して、実行してみましょう。

```
#!/bin/csh

set value = hidekazu
echo value
echo $value
```

このシェルスクリプトを実行すると、表示される結果が異なることが分かります。変数に保存した値を扱いたいときには、変数の前に$を付けるようにしましょう。

シェルスクリプト内で計算を行う場合には、@を使います。例を次に示します。

```
#!/bin/csh
@ t = 1
echo $t
@ t = $t + 1
echo $t
```

このシェルスクリプトを実行すると、計算結果が表示されます。

4.2.2 引数

シェルスクリプトを実行するときに、ファイル名などの引数を付けて実行したい場合があります。シェルスクリプトでは、実行時に付けられた引数を、$(数字) で参照することができます。$0 はシェルスクリプトの名前、$1 は 1 つ目の引数となります。また、すべての引数を対象とするために、$*が用意されています。$*では、シェルスクリプト自体の名前は含まれていません。以下のシェルスクリプトを作成して、引数がどの変数に入っているかを確認しましょう。

```
#!/bin/csh

echo $0 $1
echo $*
```

このシェルスクリプトを実行すると、シェルスクリプトの名前と第 1 引数とすべての引数が表示されます。

4.2.3 制御構造

シェルスクリプトには、実行をコントロールするための制御構造を持っています。これらの制御文を使いこなすことによって、複雑なコマンドの実行を制御することができます。

foreach 書式は以下の通りです。

```
foreach 変数（リスト）
  リストの項目を一つずつ変数に代入して処理
end
```

while 書式は以下の通りです。

```
while（条件式）
  条件文と一致した時の処理
end
```

if　書式は以下の通りです。

```
if（条件文）then
  条件文と一致した場合の処理
else
  条件文と一致しなかった場合の処理
endif
```

シェルスクリプトではファイルに関する式が用意されています。代表的なものを以下にあげます。

- -r filename
  ファイルの読み込み許可がある場合に真
- -w filename
  ファイルの書き込み許可がある場合に真
- -x filename
  ファイルの実行許可がある場合に真
- -e filename
  ファイルが存在すれば真
- -f filename
  ファイルが通常のファイルなら真
- -d filename
  ファイルがディレクトリなら真

これらの条件式を組み合わせることで、シェルスクリプトからファイルの細かい操作が可能となります。

4.2.4　シェルスクリプトの例

それでは制御構造を利用して、実際のシェルスクリプトを作成してみましょう。まずは foreach から説明をしていきます。それでは、次のシェルスクリプトを入力してください。

```
#!/bin/csh

foreach i ($*)
  cp $i $i.bak
end
```

このシェルスクリプトは、引数に与えたファイル名に.bak という拡張子を付けて保存します。たとえば、「./シェルスクリプト名 *.c」というように使います。メタキャラクタの特徴から、「cp * *.bak」のような使い方ができないので、たくさんのファイルを変換するときには、このシェルスクリプトが役に立ちます。

次に while について説明をします。次のシェルスクリプトは、10 回文字を表示するスクリプトです。

```
#!/bin/sh
@ t = 0
while($t < 10)
  echo "OK($t)"
  @ t = $t + 1
end
```

これを実行すると、画面に 10 個 OK がでます。これは、実験のプログラムを自動で実行するときに役に立つスクリプトです。

次に if の説明に移ります。このシェルスクリプトは.bashrc というファイルがあると、Found とメッセージを返します. なければ、Not found と返します。

```
#!/bin/csh
if (-e .bashrc) then
  echo Found
else
  echo Not found
endif
```

このスクリプトは、ファイルが存在するかどうかで動作を変更することができるので、現在の UNIX の状態に応じて実行するコマンドを切り替えることができます。