

2006/10/28



2006年10月28日開催

オープンソースカンファレンス 2006 Tokyo/Fall

Linux ハンズオンセミナー

『シェル環境のカスタマイズと
シェルスクリプト作成の基礎知識を学習しよう』

日本電子専門学校

杉松秀利

申 載雄

木下稔雅

sugimatsu@mail.pcmarks.jp

sin@jec.ac.jp

kinosita@jec.ac.jp

Copyright(c) : Japan Electronics College All Rights Reserved.

シェル環境のカスタマイズとシェルスクリプトの作成

本日のハンズオンセミナーでは、シェルを使用する環境をカスタマイズする方法と、シェルスクリプトを作成するために基礎知識について学習します。

I シェル環境の設定の仕組み

セミナーの前半は、シェル環境のカスタマイズを行うために基礎知識について学習します。キーワードは、シェル変数、環境変数、そして初期化スクリプトです。

1. シェルについて

(1) シェルとは何か？

- シェル(shell)とは、コマンドラインで私たちがキーボードから入力した文字列を解釈して、その文字列に対応する処理を実行するようにカーネルに伝えてくれるソフトウェアのことです。
- 本来は、貝殻という意味です。
- ユーザから見たときに、カーネルを包み込んでいる貝殻のような存在であることから、シェルという名称が付けられたとのことです。

(2) もしシェルがなかったら

- Linux のユーザが何らかの操作をしたいときには、どのような操作をしたいのかをカーネルに伝え、その指示を受けてカーネルが実行してくれます。
- もし、シェルがなかったとしたら、プログラムの中からシステムコールという機能呼び出す方法によって、ユーザの意思をカーネルに伝えなければなりません。
- これでは、プログラミングができない人は、Linux を操作できないということになってしまいます。
- そこで、コマンド名やプログラム名を入力するだけで、システムコールを呼び出してくれる、シェルというソフトウェアが開発されたのです。
- まさに、シェルは Linux ユーザがカーネルと対話するための仲介役というわけです。

(3) Linux のシェル

- これまでに、UNIX 用に何種類ものシェルが開発されてきました。
- UNIX の標準仕様を定める POSIX 規格では、B シェル(Bourne Shell: ボーン・シェル)を、標準のシェルとして採用しています。
- Linux では、B シェルを機能拡張した bash(Bourne-Again SHell)というシェルが、標準的なシェルとして使用されています。

- このため、みなさんは使用するシェルを自由に選ぶことができますが、まず bash の使い方をマスターしてから、他のシェルについて学習した方がよいでしょう。

2. シェル環境の設定する変数

シェルを使用する環境は、シェル変数と環境変数という 2 種類の変数を使って設定されています。

(1) 変数とは

- 変数は、シェルやその他のプログラムが実行時に利用するものです。
- 一般的にはユーザーの名前や、ターミナルの設定、言語環境の設定、そして個別のプログラムのための設定など、さまざまな情報が数値や文字列で、あらかじめ設定されています。
- Linux では、シェル変数と環境変数という 2 種類の変数が使用されています。

(2) シェル変数とは

- 現在使用しているシェル内でのみ有効な変数のことを、シェル変数と言います。
- シェル(bash)が、シェル自身の動作を定義するために使用する変数です。
- シェルスクリプトや、コマンドの入力の際に、値を一時的に格納しておき、後で再利用できるようにするための変数でもあります。
- 環境変数とは異なり、シェル以外のプログラムからは参照できません。
- また、現在のシェルの中から、新しいプログラム(子プロセス)を起動したときには、現在適用されているシェル変数は、引き継がれません。

(3) シェル変数を操作するには

シェル変数を操作する場合には、次のようなコマンド等を使います。

| コマンド等 | 機能 | 使用例 |
|-------|------------------------|---------------|
| set | 現在設定されているシェル変数の一覧を表示する | set |
| = | シェル変数を設定(変更)する | シェル変数名=値 |
| echo | シェル変数の値を表示する | echo \$シェル変数名 |
| unset | シェル変数の設定を解除する | unset シェル変数名 |

◇シェル変数の一覧を表示する操作例

```
$ set
```

◇シェル変数を設定する操作例

```
$ PARA1=256
```

注: =の前後には、スペースを挿入してはいけない。

◇スペースを含む値をシェル変数に設定する操作例

```
$ TESTCOM="ls /etc/sysconfig"
```

◇シェル変数の値を表示する操作例

```
$ echo $TESTCOM
```

注: シェル変数の値を参照する場合には、変数名の前に、必ずドル記号(\$)を付ける。

◇シェル変数の設定を解除する操作例

```
$ unset TESTCOM
```

【設定上の注意事項】

- 変数名は、自由に指定できます。
- ただし、1文字目は必ずアルファベット、またはアンダーバー(_)を使用してください。
- すでに定義されている変数名と重複しないように注意してください。
- シェル変数の値としては、数値も文字列も設定できます。
- 空白を文字列を設定する場合には、文字列をクォーテーション(')、またはダブルクォーテーション(")で囲んで指定してください。

【使用上の注意事項】

- シェル変数に格納されている値を参照する場合には、シェル変数名の前に、ドル記号(\$)を付けます。
- シェルは、ドル記号(\$)に続く文字列が正しいシェル変数名ならば、その変数に格納されている値と置き換えます。
- 何も格納されていないシェル変数を参照した場合には、初期値のヌルで置き換えられます。

(4) 環境変数とは

- 環境変数とは、シェルによって起動されたすべてのプロセスで、有効に使用できる変数のことです。
- 環境変数は、最初にシェル変数として設定したのちに、export コマンドを使って設定します。
- つまり、export されたシェル変数が、環境変数になるわけです。
- シェル変数と環境変数では、基本的な機能は同じですが、適用範囲が異なっている、というわけです。
- 環境変数は、アプリケーションやコマンドなどの動作を、統一的に制御するために、ユーザーが設定します。

(5) 環境変数を操作するには

環境変数を操作する場合には、次のようなコマンド等を使います。

| コマンド | 機能 |
|------------------|----------------------------------|
| printenv または env | 現在設定されている環境変数を表示する (env は一覧表示のみ) |
| export | 環境変数を設定(変更)する |
| unset | 環境変数の設定を解除する |

◇環境変数の一覧を表示する操作例

```
$ printenv または $ env
```

◇特定の環境変数の値を表示する操作例 (環境変数 HOME の値を表示する場合)

```
$ printenv HOME
```

◇環境変数を設定する操作例（シェル変数として設定したのちに環境変数に設定する場合）

| | |
|-----------------|--------------------------|
| \$ PAGER=lv | ← シェル変数 PAGER を設定する |
| \$ export PAGER | ← シェル変数 PAGER を環境変数に設定する |

◇環境変数を設定する操作例（シェル変数と環境変数を同時に設定する場合）

| |
|--------------------|
| \$ export PAGER=lv |
|--------------------|

◇環境変数の設定を解除する

| |
|----------------|
| \$ unset PAGER |
|----------------|

【注意事項】

当然のことですが、unsetコマンドを使って環境変数の設定を解除した場合には、シェル変数の設定も解除されます。

3. bash の初期化スクリプト

- bash には、初期化スクリプトと呼ばれる複数の実行ファイルが用意されています。
- bash を使用するシェル環境は、この初期化スクリプトを実行する過程で、必要な環境変数を定義することによって設定されます。

(1) 初期化スクリプトの設定内容

bash の初期化スクリプトの実行順番と設定内容は、次のとおりです。

- ① ログイン画面でユーザ名とパスワードを入力して、ユーザがログインに成功すると、**bash** が起動します。
- ② **bash** は、まず最初に **/etc/profile** ファイル を読み込んで実行します。この初期化スクリプトには、システム全体に影響を与える環境変数の設定が記述されています。
- ③ **/etc/profile** ファイルによって、**/etc/profile.d/ディレクトリ** 以下にあるすべてのスクリプトファイルが実行されます。このファイルには、各種のツールに必要な環境変数の設定が記述されています。
- ④ 続いて、ログインユーザのホームディレクトリに保存されている **.bash_profile** ファイル が実行されます。このスクリプトを読み込み実行するのは、ログイン時の **bash** だけです。このため、ログイン時に各ユーザごとのシェル環境のカスタマイズを行いたい場合には、このファイルにシェル変数の設定内容を記述します。
- ⑤ **.bash_profile** ファイルによって、**.bashrc** ファイル が読み込まれ実行されます。このスクリプトは、ログイン時の **bash** だけでなく、ログイン後の **bash** の起動時にも読み込まれるファイルです。このファイルも、**.bash_profile** ファイルと同様に、ユーザのシェル環境のカスタマイズを行うために使用されます。ログイン後のシェル環境にも反映させたいシェル変数の設定内容は、このファイルに記述します。
- ⑥ 最後に、**./bashrc** ファイルによって、**/etc/bashrc** ファイル が読み込まれ実行されます。このファイルは、**/etc/profile** ファイルと同様に、システム全体に影響を与える環境変数の設定を記述するために使用されています。

上記の内容を整理すると、次のようになります。

◇bash の初期化スクリプト

| 順番 | ファイル名 | 実行時期 | 定義されている変数 | 適用範囲 |
|----|-----------------|-----------------|-----------|------|
| 1 | /etc/profile | ログイン時に実行される | 環境変数 | 全ユーザ |
| 2 | /etc/profile.d/ | ログイン時に実行される | 環境変数 | 全ユーザ |
| 3 | ~/.bash_profile | ログイン時に実行される | シェル変数 | 各ユーザ |
| 4 | ~/.bashrc | bash の起動時に実行される | シェル変数 | 各ユーザ |
| 5 | /etc/bashrc | bash の起動時に実行される | 環境変数 | 全ユーザ |

注：Debian の場合には、~/.bash_profile ファイルは、~/.profile ファイルになっています。

◇その他の初期化スクリプト

| ファイル名 | 内容 |
|----------------|---|
| ~/.bash_login | ~/.bash_profile ファイルがない場合に、ログイン時に実行されるスクリプト |
| ~/.bash_logout | ログアウト時に実行されるスクリプト |
| ~/.inputrc | シェル環境でのキー入力などに関する設定を行うスクリプト |

4. シェル環境のカスタマイズ方法

- 以上の説明によって、シェル環境が bash の初期化スクリプト内に記述されているシェル変数と環境変数の定義によって設定されることが、お分かり頂けたと思います。
- 私たちがシェル環境をカスタマイズする場合には、この仕組みを利用して、環境の変更を行うタイミングに応じて、それぞれの初期化スクリプト内に必要なシェル変数または環境変数の設定や、実行したい処理(コマンド)を記述すればよいのです。

(1) プロンプトの表示を変更する (コマンドラインで設定する場合)

- ログイン時に表示されるコマンドプロンプトの表示形式は、環境変数 PS1 によって設定されています。
- 環境変数 PS1 の設定には、次のような特殊な文字コードが使用されます。

◇コマンドプロンプトの表示設定に使用される文字コード

| 文字コード | 表示内容 |
|-------|-------------------------------------|
| ¥a | ASCII のベル文字 (07) を表示する (ビーブ音を鳴らす) |
| ¥d | 「曜日 月 日」の形式 (例: Fri Jan 5) で日付を表示する |
| ¥e | ASCII のエスケープ文字 (033) を表示する |
| ¥h | ホスト名のうち最初の「.」までの部分を表示する |
| ¥H | ホスト名を表示する |
| ¥n | 改行する |
| ¥r | 復帰する |
| ¥s | シェルの名前を表示する |
| ¥t | 現在の時刻を 24 時間の「HH:MM:SS」形式で表示する |
| ¥T | 現在の時刻を 12 時間の「HH:MM:SS」形式で表示する |
| ¥@ | 現在の時刻を 12 時間の「am/pm」形式で表示する |
| ¥u | 現在のユーザー名を表示する |

| | |
|------|---|
| ¥v | bash のバージョンを表示する |
| ¥V | bash のリリースを表示する |
| ¥w | 現在の作業ディレクトリを、ユーザーのホームディレクトリからの絶対パスで表示する |
| ¥W | 現在の作業ディレクトリを表示する |
| ¥! | このコマンドの履歴番号を表示する |
| ¥# | このコマンドのコマンド番号(現在のシェルのセッション中に実行されたコマンドのシーケンスにおける位置)を表示する |
| ¥\$ | 実効 UID が 0 の場合に#となり、それ以外の場合に\$となる |
| ¥nnn | 8 進数 nnn に対応する文字を表示する |
| ¥¥ | バックスラッシュを表示する |
| ¥[| 非表示文字のシーケンスを開始する。これを使って、プロンプト中に端末の制御シーケンスを埋め込むことができる |
| ¥] | 非表示文字のシーケンスを終了する |

注：Linux 上では、円マーク(¥)はバックスラッシュとして入力されます。

一般ユーザでログインしたときの標準的なコマンドプロンプトは、次のような形式で表示されますね。

```
[ユーザ名@ホスト名 カレントディレクトリ名] $ ■
```

そこで、コマンドラインで上記のような形式で表示されるように、カスタマイズしてみましょう。

◇プロンプトの表示を変更する操作 (シェル変数として変更する場合)

```
$ PS1="[¥u@ws0xx ¥w]¥¥$ "
```

上記のとおり入力して Enter キーを押すと、瞬時にプロンプトの表示が変更されることを確認してください。

また、新しい bash が起動した場合にも変更後の表示形式を維持するためには、次のように環境変数として設定します。

◇プロンプトの表示を変更する操作 (環境変数として変更する場合)

```
$ export PS1="[¥u@ws0xx ¥w]¥¥$ "
```

(2) プロンプトの表示を変更する (~/.bashrc ファイルで設定する場合)

- コマンドラインで変更した設定内容は、システムを停止または再起動した場合に消去されてしまいます。
- このため、設定内容を永続的に使用したい場合には、 ~/.bash_profile ファイルや ~/.bashrc ファイルに記述します。
- 環境変数 PS1 の設定変更は、 ~/.bashrc ファイルに記述することによって、ログイン後の bash の起動時にも反映されます。

◇プロンプトの表示を変更する操作 (~/.bashrc ファイルで設定する場合)

```
$ vi ~/.bashrc
....
export PS1="[¥u@ws0xx ¥w]¥¥$ " ← ファイルの末尾に追加する
```

注：Esc キーを押したのちに、:wqと入力してEnterキーを押すと、上書き保存してviが終了します。

(3) bash 起動時の日時を表示する (~/.bashrc ファイルで設定)

- 次に、新しい bash が起動するごとに、その時点での日時(含む曜日)を表示するように、シェル環境をカスタマイズしてみましょう。
- 設定操作は、簡単です。vi で ~/.bashrc ファイルを開き、ファイルの末尾に date という記述を追加するだけです。
- date は、日時の情報を取得するコマンドです。
- ~/.bashrc は、スクリプトファイルであるため、ファイルの末尾に date と追記するだけで、bash が起動するごとに、date コマンドが自動的に実行されるようになるのです。

◇bash 起動時の日時を表示する操作 (~/.bashrc ファイルで設定)

```
$ vi ~/.bashrc
.....
date                ← ファイルの末尾に追加する
```

注: Esc キーを押したのちに、:wqと入力して Enter キーを押すと、上書き保存して vi が終了します。

コンソール(ターミナルエミュレータ)を一旦終了させ、再度起動すると、コンソールの先頭行に日時情報が表示されることを確認してください。

II シェルスクリプトの作成手順

- シェルによって解釈/実行されるプログラムのことを、シェルスクリプトと言います。
- 複数のコマンドを組み合わせて実行させるようなシェルスクリプトを作成することによって、自分独自のコマンドを作ることができます。
- シェルスクリプトを利用することによって、コマンドラインでは実行しにくい複雑な処理の操作も、簡単に実行できるようになります。

1. 簡単なシェルスクリプトを作成してみよう

それでは、プログラムを実行すると、

```
Execute Shell Script
```

と表示する簡単なシェルスクリプトを作成しながら、シェルスクリプトを作成する手順を学習することにししましょう。

(1) スクリプトファイルを保存するディレクトリを作成/移動する

- まず最初に行う操作は、作成したシェルスクリプトファイルを、まとめて格納するディレクトリを作成し、そのディレクトリへ移動することです。
- ここでは、script という名前のディレクトリを、knoppix ユーザのホームディレクトリの下に作成する例で説明します。

◇新規に script ディレクトリを作成した後に移動する操作

```
$ mkdir script      ← script という名前のディレクトリを新規に作成する
$ cd script         ← 作成された script ディレクトリへ移動する
```

(2) シェルスクリプトを作成する

vi を使って、test1 という名前のシェルスクリプトを作成します。

```
$ vi test1
#!/bin/bash          ← この記述から始めるのが、シェルスクリプトの作法です。
                     シェルスクリプトを実行するためのシェルを指定しています。
echo "Execute Shell Script"
```

注： Esc キーを押したのちに、:wq と入力して Enter キーを押すと、保存して vi が終了します。

(3) スクリプトファイルに実行権限を設定する

作成したばかりのシェルスクリプトファイルは、他のファイルと同じパーミッションが設定されています。ls コマンドを使って、確認してみましょう。

◇作成したばかりのスクリプトファイルのパーミッションを確認する操作

```
$ ls -l test1
-rw-r--r-- 1 knoppix knoppix 41 2006-10-28 14:20 test1
```

- このままのパーミッションでは、実行権限がないため、実行ファイルとして使用できません。
- そこで、次の操作によって、スクリプトファイルに実行権限を設定します。

◇スクリプトファイルに実行権限を設定する操作（すべてのユーザによって実行可能な権限）

```
$ chmod +x test1
または
$ chmod 755 test1
```

なお、シェルスクリプトを作成したファイルの所有者のみに実行権限を設定する場合には、次のように操作します。

◇スクリプトファイルに実行権限を設定する操作（スクリプトファイル所有者のみ実行可能な権限）

```
$ chmod u+x test1
または
$ chmod 744 test1
```

(4) シェルスクリプトを実行する

とりあえず、次の操作を実行して、作成したシェルスクリプトを起動してみましょう。

◇とりあえずシェルスクリプト test1 を実行する操作

```
$ ./test1
Execute Shell Script
$
```

上記の操作では、シェルスクリプトのファイルを保存している script ディレクトリが、コマンドパスとも呼ばれる環境変数 PATH に追加設定されていないため、./というパスを指定して実行していることに注意してください。

(5) script ディレクトリを環境変数 PATH に追加する

- 現在のシェル環境でも、シェルスクリプトを実行するのに支障はありませんが、実行するたびに、シェルスクリプト名の前に./というパスの指定を記述することが必要です。
- そこで、環境変数 PATH に、/home/knoppix/script ディレクトリを追加することにします。

◇script ディレクトリを環境変数 PATH に追加する

```
$ export PATH=$PATH:$HOME/script
```

注：\$PATH には、変更前の環境変数 PATH の設定値が格納されています。

\$HOME には、ログインユーザである knoppix のホームディレクトリ/home/knoppix という値が格納されています。

これで、/home/knoppix/script ディレクトリも、コマンドパスに追加されたので、他のシェルコマンドと同様に、./の指定なしで実行ができるはずです。早速、確かめてみましょう。

◇環境変数 PATH の設定変更後のシェルスクリプトの実行操作

```
$ test1
Execute Shell Script
$
```

以上が、シェルスクリプトを使用する際の一連の操作手順です。

Ⅲ シェルスクリプトの要素

- シェルスクリプトはプログラムですので、他のプログラミング言語と同様に、複雑な処理を行うために利用できる要素が用意されています。
- 本セミナーの最後のテーマとして、シェルスクリプトの主要な要素を整理しながら、それらの要素を使ったスクリプトを作成してみることにしましょう。

1. シェルスクリプトの引数処理

- Linux では、コマンドラインからシェルスクリプトに引数を渡すことができます。
- シェルスクリプトを実行する際に、ファイル名やその他の文字列などを、引数として渡すことができれば、シェルスクリプトを活用する範囲は、大幅に広がります。

(1) 引数処理用の変数

シェルには、シェルスクリプトで引数を処理するための変数が、組み込み変数として用意されています。

主な引数用の組み込み変数は、次のとおりです。

◇主要な引数用の組み込み変数

| 変数名 | 内容 |
|-----------|--|
| \$@ | スペースで区切られたすべての引数を格納する変数です。 |
| *\$ | 変数 IFS に指定された文字で区切られたすべての引数を格納する変数です。 変数 IFS が未定義の場合には、スペースで区切られたすべての引数を格納します。 |
| \$# | シェルスクリプトを実行する際に指定された引数の数を格納する変数です。 |
| \$0 | コマンド名(シェルスクリプト名)を、フルパス付きで格納する変数です。 |
| \$1 ~ \$9 | 1 番目から 9 番目に指定された引数を格納する変数です。たとえば、1 番目に指定された引数は変数\$1 に、5 番目に指定された引数は変数\$5 に格納されます。 |
| \$(10) ~ | 10 番目以降に指定された引数を格納する変数です。たとえば、12 番目に指定された引数は、変数\$(12)に格納されます。 |

注:

- 一般的には、引数の区切り文字としてはスペース(空白)が使用されるため、変数 IFS を使って、特別な区切り文字を指定することは、ほとんどありません。
- このため、変数\$@と*\$は、同じ機能の変数として使用することができるのですが、ほとんどのシェルスクリプトでは、変数\$@が使用されています。

(2) 最もよく使用される引数処理用変数

シェルスクリプトを作成するとき、最もよく使用される引数処理用の変数は、次の2種類です。

| | |
|---------------|--------------------------------|
| \$@ | ← スペースで区切られたすべての引数を格納する |
| \$1 \$2 . . . | ← 1 番目の引数、2 番目の引数、n 番目の引数を格納する |

◇主要な引数処理用の変数を使用するシェルスクリプト

ファイル名: varlist

```
#!/bin/bash
echo Shell Script Name : $0
echo Parameters Num : $#
echo No1 parameter : $1
echo No2 parameter : $2
echo No3 parameter : $3
echo No4 parameter : $4
echo No5 parameter : $5
echo All parameter : $@
```

◇スクリプトファイルに実行権限を設定する操作 (すべてのユーザによって実行可能な権限)

| | | |
|---------------------|-----|----------------------|
| \$ chmod +x varlist | または | \$ chmod 755 varlist |
|---------------------|-----|----------------------|

◇シェルスクリプト varlist の実行操作

| | |
|-------------------------------|--------------------------------|
| \$ varlist 11 12 13 | ← 11、12、13 という 3 個の引数を指定して実行する |
| Shell Script Name : ./varlist | |
| Parameters Num : 3 | |
| No1 parameter : 11 | |

```
No2 parameter : 12
No3 parameter : 13
No4 parameter :
No5 parameter :
All Parameter : 11 12 13
```

2. if 文の使い方 -- フロー制御①

- シェルスクリプトを作成するときには、処理の流れを制御(フロー制御)するために、いくつかの制御文を使うことができます。
- これらの制御文のなかで、最も一般的に使用されるものが if 文です。
- if 文を使用すると、指定した条件の判断結果によって、異なるステートメントを実行させることができるようになります。
- このようなフロー制御のことを、条件分岐と言います。

(2) if 文の書式

if 文を定義するときの書式は、次の3種類に分けられます。

◇書式 1: 指定した条件に合った場合に、指定した実行分を実行する構文

```
if 条件文
then
    実行文
fi
```

【書式 1 の別の書き方】

書式 1 は、セミコロン(;)を使って、次のように 1 行で記述することもできます。

```
if 条件文; then 実行文; fi
```

◇書式 2: 条件文に合った場合には実行文 1 を、合わなかった場合には条件文 2 を実行する構文

```
if 条件文
then
    実行文 1
else
    ← else 句と言います
    実行文 2
fi
```

【書式 2 の別の書き方】

書式 2 は、セミコロン(;)を使って、次のように 1 行で記述することもできます。

```
if 条件文; then 実行文 1; else 実行文 2; fi
```

◇書式 3: 条件文 1 に合った場合には実行文 1 を、条件文 2 に合った場合には実行文 1 を、いず

れの条件文にも合わなかった場合には実行文 3 を実行する構文

```
if 条件文 1
then
    実行文 1
elif 条件文 2      ← elif 句と言います
then
    実行文 2
else                ← else 句
    実行文 3
fi
```

注： elif 句は、いくつでも連続して定義することができます。

【書式 3 の別の書き方】

書式 3 は、セミコロン(;)を使って、次のように 1 行で記述することもできます。

```
if 条件文 1; then 実行文 1; elif 条件文 2; then 実行文 2; else 実行文 3; fi
```

3. 条件文の機能

- 条件文は、シェルによって判断され、条件に合っている場合には真(true)、合っていない場合には偽(false)という値が返されます。
- たとえば、書式 2 で定義されえた if 文の場合であれば、条件判断の結果として、真(true)という値が返された場合には、then 句で指定した実行文 1 が実行され、偽(false)という値が返された場合には、else 句で指定した実行文 2 が実行されるという仕組みになっています。

(1) 条件文の指定内容

- 条件文は、シェルコマンドなどの通常のコマンド、または test コマンドを使って定義します。
- シェルコマンドなどの通常のコマンドを使用した場合には、そのコマンドが正常に実行されたときに、真(true)という値が返され、then 句で指定した実行文が実行されます。
- test コマンドの使用方法については、後で詳しく説明します。

◇シェルコマンドなどの通常のコマンドにより条件文を指定するシェルスクリプト ファイル名 : test2

```
#!/bin/bash

if find $1      ← 引数に指定したファイルが存在する場合には真という値が返される
then
    vi $1      ← 真が返された場合には、そのファイルを vi で開く
else          ← 偽が返された場合には、
    echo "error : This File Not Found."      エラーメッセージを表示する
fi
```

◇スクリプトファイルに実行権限を設定する操作

```
$ chmod +x test2   または   $ chmod 755 test2
```

◇シェルスクリプト test2 の実行操作

```
$ test2 test1           ← 存在するファイル名を指定して実行してみる
.....
$ test2 dummy           ← 存在しないファイル名を指定して実行してみる
```

(2) test コマンドを使った条件判断

- 一般的には、条件文は test コマンドを使って定義されます。
- test コマンドの判定オプションを使えば、さまざまな状況に対応する条件文を定義することができます。

シェルスクリプトの真・偽の値

シェルスクリプトでは、実際には真 (true) と偽 (false) の値として、次のような値が返されるようになっています。

| 真・偽 | 値 |
|-----|---|
| 真 | 0 |
| 偽 | 1 |

- 上記の値は、C 言語や Java などの一般的なプログラミングシステムで使用される真・偽の値と異なる、ということに注意しておいてください。
- 一般的なプログラミングシステムでは、真が1、偽が0という値で返されます。

(3) test コマンドの書式

- test コマンドを記述する場合には、次のいずれかの書式が使用できます。
- 一般的には、書式②の書式が使用されています。

書式①

```
if -test 条件
```

書式②

```
if [ 条件 ]
```

↑ ↑
スペースを1個入れる

書式②は、書式①を簡略化した記述方法として考案された書式ですが、現在では、書式②の方が、一般的な記述方法になっています。

(4) test コマンドの条件判定オプション

test コマンドで指定できる条件判定オプションは、次のとおりです。

◇ファイル処理に関する判定オプション

| オプションの種類 | 判定内容 |
|----------|---|
| -d ファイル名 | 引数に指定したファイルが、ディレクトリである場合に真が返される |
| -e ファイル名 | 引数に指定したファイルが、存在する場合に真が返される |
| -f ファイル名 | 引数に指定したファイルが、通常のファイルである(ディレクトリなどでない)場合に真が返される |
| -L ファイル名 | 引数に指定したファイルが、シンボリックリンクである場合に真が返される |
| -r ファイル名 | 引数に指定したファイルが、読み取り可能である場合に真が返される |
| -s ファイル名 | 引数に指定したファイルのサイズが、0でない場合に真が返される |
| -w ファイル名 | 引数に指定したファイルが、書き込み可能である場合に真が返される |
| -x ファイル名 | 引数に指定したファイルが、実行可能である場合に真が返される |

◇文字列処理に関する判定オプション

| オプションの種類 | 判定内容 |
|----------------|---|
| -z 文字列 | 引数に指定した文字列が、NULL 文字列(長さ 0 の文字列)であれば真が返される |
| -n 文字列 | 引数に指定した文字列が、NULL 文字列でなければ真が返される |
| 文字列 1 = 文字列 2 | 指定した2つの文字列が、等しい場合に真が返される |
| 文字列 1 != 文字列 2 | 指定した2つの文字列が、等しくない場合に真が返される |

◇数値処理に関する判定オプション

| オプションの種類 | 判定内容 |
|-----------------|---------------------------------|
| 整数値 1 -eq 整数値 2 | 指定した2つの整数値が、等しい場合に真が返される |
| 整数値 1 -ne 整数値 2 | 指定した2つの整数値が、等しくない場合に真が返される |
| 整数値 1 -lt 整数値 2 | 整数値 1が整数値 2よりも、小さい場合に真が返される |
| 整数値 1 -le 整数値 2 | 整数値 1が整数値 2よりも小さいか、等しい場合に真が返される |
| 整数値 1 -gt 整数値 2 | 整数値 1が整数値 2よりも、大きい場合に真が返される |
| 整数値 1 -ge 整数値 2 | 整数値 1が整数値 2よりも大きいか、等しい場合に真が返される |

◇論理演算に関する判定オプション

| オプションの種類 | 判定内容 |
|--------------|-------------------------------|
| 1 条件 | 条件が偽(返される値が1)である場合に真が返される |
| 条件 1 -a 条件 2 | 条件 1と条件 2が、共に真である場合に真が返される |
| 条件 1 -o 条件 2 | 条件 1と条件 2のどちらかが、真である場合に真が返される |

◇引数に指定した文字列がディレクトリ名であるかどうかを判断するシェルスクリプト

```
#!/bin/bash

if [ -d $1 ]      ← 引数に指定した文字列が、ディレクトリ名であるかどうかを判定する
then             ← 真である(ディレクトリ名である)場合には、
    cd $1        ← cd コマンドを使って、そのディレクトリへ移動する
    ls -la       ← ls コマンドを使って、すべてのファイル一覧を表示する
else             ← 偽(ディレクトリ名ではない)場合には、
    vi $1        ← vi を使って、そのファイルを開く
fi
```

◇スクリプトファイルに実行権限を設定する操作

```
$ chmod +x check1   または   $ chmod 755 check1
```

◇4 個の引数が指定されているかどうかを判断するシェルスクリプト

ファイル名 : check2

```
#!/bin/bash

if [ $# -eq 4 ]; then      ← 指定した引数の数が4であるかどうかを判定する
    echo "-----result"  ← 真である(4である)場合には、
    echo $4 $3 $2 $1      ← 指定した引数を入力順とは逆の順番で表示する
else                       ← 偽である(4ではない)場合には、
    echo "-----error"
    echo "$0 : Parameters do not match." ← シェルスクリプト名(フルパス付き)と
                                エラーメッセージを表示する
fi
```

◇スクリプトファイルに実行権限を設定する操作

```
$ chmod +x check2   または   $ chmod 755 check2
```

4. for 文の使い方 -- フロー制御②

- 指定した処理を、繰り返し実行したい場合には、for 文を使用します。
- for 文も、if 文と同様に、シェルスクリプトのなかでは、頻繁に使用されます。

1. for 文の書式

for 文は、次のような書式を使って定義します。

◇for 文の書式

```
for 変数名 in 引数リスト
do
    実行コマンド
done
```

| | |
|--------|----------------------------|
| 変数名 | 実行コマンドの引数として使用する変数名を指定します。 |
| 引数リスト | シェルスクリプトの引数を格納する変数を指定します。 |
| 実行コマンド | 繰り返し実行するコマンドを指定します。 |

注：

「in 引数リスト」の指定を省略した場合には、「in "\$@"」が指定されたものとみなされて処理されま
す。つまり、シェルスクリプトを実行する際に指定されたすべての引数が、処理の対象となります。

◇for 文を使ったシェルスクリプト①

ファイル名 : lsfiles

| | |
|--------------------|---------------------------------|
| #!/bin/bash | |
| for fname in "\$@" | ← 指定されたすべての引数を対象とする |
| do | |
| ls -l \$fname | ← それぞれのファイルについて、ls -l コマンドを実行する |
| done | |

◇スクリプトファイルに実行権限を設定する操作

| |
|--|
| \$ chmod +x lsfiles または \$ chmod 755 lsfiles |
|--|

◇for 文を使ったシェルスクリプト②（サンプル）

| | |
|--------------------------|---------------------------------|
| #!/bin/bash | |
| for fname in \$1 \$2 \$3 | ← いくつ指定されても、最初の3個の引数だけを対象とする |
| do | |
| ls -l \$fname | ← それぞれのファイルについて、ls -l コマンドを実行する |
| done | |

5. while 文の使い方 -- フロー制御③

- 指定した条件を満たしている間、指定した処理を繰り返し実行したい場合には、while 文を使用します。
- C 言語などのプログラミング言語では、特定の条件が続く間、数値計算を繰り返し実行する場合などに、while 文がよく使用されます。
- しかし、計算処理を行うために、シェルスクリプトを使用することは、けっして多くありません。
- このため、while 文は、for 文や if 文ほど、頻繁に使用されることのないフロー制御文です。

(1) while 文の書式

while 文は、次のような書式を使って定義します。

◇while 文の書式

```
while 条件文
do
    実行コマンド
done
```

| | |
|--------|--|
| 条件文 | test コマンドや普通のコマンドを使って定義します。記述方法は、if 文で使用する条件文の書き方と、まったく同じです。 |
| 実行コマンド | 繰り返し実行するコマンドを指定します。 |

◇while 文を使ったシェルスクリプト

ファイル名 : shiftpara

```
#!/bin/bash
while [ "$1" != "" ]           ←1番目の引数が空でない間、以下のコマンドを繰り返し実行する
do
    echo "----- $1 -----"   ← 1番目の引数の値を表示する
    shift                       ← 引数を1つシフトする
done
```

重要

shift コマンドは、引数の位置を、ひとつずらす働きをします。上記のスクリプトでは、shift コマンドを実行するごとに、1番目の位置にあった引数は追い出され、2番目の引数が1番目の引数の位置に移動することになります。

while 文の使用方法については、後で説明する数値計算の際に、あらためて学習することになります。

6. case 文の使い方 -- フロー制御④

- 指定した文字列が、定義された複数のパターン(文字列)のどれと一致するかによって、実行するコマンドを制御するのが、case 文です。
- このような処理のことを、パターンマッチングと言います。
- コマンドを実行する条件が多い場合に、使用すると便利です。
- case 文も、if 文や for 文と同様に、シェルスクリプトのなかでは、頻繁に使用されます。

(1) case 文の書式

case 文は、次のような書式を使って定義します。

◇case 文の書式

| | |
|-------------|------------------|
| case 文字列 in | ← 指定した文字列の値が、 |
| パターン 1) | ← パターン1に一致するならば、 |
| 実行コマンド 1 | ← 実行コマンド1を実行する |

```

;;                                     ← 実行コマンドの定義の終了位置を指定する
パターン 2)
    実行コマンド 2-1                   ← 複数の実行コマンドを指定できる
    実行コマンド 2-2
    実行コマンド 2-3
;;
パターン n)
    実行コマンド n
;;
*)                                     ← どのパターンにも一致しない場合には、
    実行コマンド*                       ← *)で指定したコマンドが実行される
;;
esac                                    ← case 文の終了を定義する

```

◇各パターン定義を1行で記述する書式 --- スクリプトの記述がすっきりする

```

case 文字列 in
    パターン 1) 実行コマンド 1 ;;
    パターン n) 実行コマンド n ;;
    パターン*) 実行コマンド 8 ;;
esac

```

| | |
|---------|---|
| 文字列 | 評価の対象となる文字列を指定します。通常の場合には、変数が指定されます。 |
| パターン n) | 上記の文字列に一致するかどうか評価される文字列を、パターンとして指定します。たとえば、指定された文字列が、パターン2に一致した場合には、実行コマンド2が実行されます。 |
| *) | どのパターンにも一致しない場合に実行されるコマンドを指定します。 |

(2) 特殊なパターンの指定方法

◇複数のパターンを一度に指定できる

次の書式を使って、複数のパターンを一度に指定することができます。

```

パターン a | パターン b)           ← パターン a かパターン b に一致するならば

```

◇ワイルドカードを使用できる

パターンとして定義する文字列として、アスタリスク(*)、疑問符(?)、ピリオド(.)などを使用することができます。

```

*.txt                               ← 「.txt」で終わる文字列を指定する(ファイル名の評価に使用する)
test?                               ← 最後に1文字が何であってもよい場合に指定する

```

◇case 文を使ったシェルスクリプト①

ファイル名 : chkpara

```
#!/bin/bash
case $# in
    0) echo "Parameter in Nothing." ;;          ← 引数の数を格納する変数を文字列として指定して
    1) echo "Parameter Num = 1" ;;             ← その値が0ならば
    2) echo "Parameter Num = 2" ;;             ← その値が1ならば
    *) echo "Parameter Num >= 3" ;;           ← その値が2ならば
                                           ← どのパターンにも一致しない場合には
esac                                           ← case 文の終了
```

◇スクリプトファイルに実行権限を設定する操作

```
$ chmod +x chkpara   または   $ chmod 755 chkpara
```

◇case 文を使ったシェルスクリプト②（サンプル）

```
#!/bin/bash
case $1 in
    -v) vi $2 ;;          ← 1番目の引数を格納する変数を文字列として指定して
        ← -v ならば、2番目の引数で指定したファイルを vi で開く
    -c) cd $2
        pwd
        ls -la ;;        ← -c ならば、2番目の引数で指定したディレクトリへ移動して
        ← 複数のコマンドを実行する
    -m) mkdir $2 ;;      ← -m ならば、2番目の引数で指定したディレクトリを作成する
    -r) rm $2 ;;         ← -r ならば、2番目の引数で指定したファイルを削除する
    -*) echo "Parameter do not Matched!" ;;    ← どのパターンにも一致しない場合
esac                     ← case 文の終了
```

7. シェルスクリプト内の数値計算

- シェル変数の内容は、すべて文字列として設定されています。
- このため、変数に格納されている文字列を、数値として四則演算するときには、`expr` コマンドを使います。
- `expr` とは、`expressions` を略した名称であり、式という意味です。
- なお、シェルスクリプト内で行える四則演算は、整数計算に限られます。
- シェルスクリプトの数値計算は、もちろん単独で使用することもできますが、多くの場合、`while` 文と組み合わせて使用されます。

(1) 数値計算の書式

数値計算は、次のような書式を使って定義します。

◇コンソール上で使用する数値計算の書式

```
expr n1 + n2      ← 加算する
expr n1 - n2      ← 減算する
expr n1 ¥* n2     ← 乗算する
expr n1 / n2      ← 除算する
expr n1 % n2      ← 余りを求める
```

注：

乗算の書式は、特殊文字機能の起動を防止するために、`¥*`(正確には、バックスラッシュとアスタリスク)の組み合わせで指定します。`*`だけで指定すると、エラーになるので注意してください。

◇シェルスクリプト内で使用する数値計算の書式

| | |
|--------------------------------------|----------------|
| <code>n1='expr \$n2 + \$n3'</code> | ← 加算する場合の書式例です |
| または | |
| <code>n1=\$(expr \$n2 + \$n3)</code> | ← この書式も使用できます |

注:

`=`(代入演算子)の前後には、スペース(空白)を挿入すると、エラーになるので注意してください。

重要

- 条件文のなかで、変数に格納されている文字列を、数値として比較する場合には、すでに説明した `test` コマンドを使って定義します。
- 記述方法は、`if` 文で使用する条件文の書き方と、まったく同じです。

条件文の指定例

| | |
|-----------------------------|---|
| <code>If [n1 - n2]</code> | ← <code>n1</code> が <code>n2</code> よりも大きいという条件を指定しています |
|-----------------------------|---|

◇1 から 10 までの値の和と積を求める演算処理を行うシェルスクリプト ファイル名 : calc1

| | |
|---|---|
| <code>#!/bin/bash</code> | |
| <code>add=0</code> | ← 変数 <code>add</code> に 10 を代入する |
| <code>mul=1</code> | ← 変数 <code>mul</code> に 1 を代入する |
| <code>nn=1</code> | ← 変数 <code>nn</code> に 1 を代入する |
| <code>while [\$nn -le 10]</code> | ← 変数 <code>nn</code> の値が 10 以下の間は繰り返す |
| <code>do</code> | |
| <code>add=\$(expr \$add + \$nn)</code> | ← 変数 <code>add</code> と変数 <code>nn</code> の値を加算する |
| <code>mul=\$(expr \$mul ¥* \$nn)</code> | ← 変数 <code>mul</code> と変数 <code>nn</code> の値を乗算する |
| <code>nn=\$(expr \$nn +1)</code> | ← 変数 <code>nn</code> の値をプラス 1 する |
| <code>done</code> | |
| <code>echo "add(1-10) = \$add"</code> | ← 加算結果を表示する |
| <code>echo "mul(1-10) = \$mul"</code> | ← 乗算結果を表示する |

注: 上記の記述例のように、`expr` コマンドを組み合わせで繰り返し処理を行うのが、`while` 文の一般的な使用方法です。

◇スクリプトファイルに実行権限を設定する操作

| | | |
|--------------------------------|-----|---------------------------------|
| <code>\$ chmod +x calc1</code> | または | <code>\$ chmod 755 calc1</code> |
|--------------------------------|-----|---------------------------------|

以 上