6章 シェルおよびシェルスクリプト

6.1.1 変数の利用

- シェル変数
  - 設定　　：　変数名＝値
  - 一覧表示　：　set　|　declare　(すべてのシェル変数と環境変数を表示)
  - 削除　　：　unset
- 環境変数
  - 設定　　：　export 変数名　|　export 変数名＝値　|　declare -x 変数名
  - 一覧表示　：　env　|　printenv
  - 削除　　：　unset 変数名

6.1.2 エイリアスの設定

- alias = コマンドの別名

```
# エイリアスの定義・確認
alias [エイリアス名='実行内容']

    # ex (デフォルトで入ってる)
    alias ll='ls -l --color=auto'
```

aliasの特徴

- コマンドを実行する際、PATHが通っているディレクトリからコマンドの実行内容を参照するのが普通だが、aliasにコマンドを登録しておくと、そちらが優先される。
- aliasを利用したくない場合は以下

```
# \を前につけて実行
\ls
```

aliasの解除

```
unalias エイリアス名

    -a をつけるとすべて削除
```

6.1.3 コマンドの連続実行

```
command1; command2      #command1に続いてcommand2を常に実行
command1 && command2    #command1が成功したらcommand2を実行
commnad1 || command2    #command1が失敗したらcommand2を実行
```

6.1.4 bashシェルの環境設定

- 変数やエイリアスは、コマンドを使って設定しても、次回ログイン時には反映されない
- bashシェルの設定ファイル
    - 全員が参照
        - ログイン時に読み込み　　：　/etc/profile
        - bash起動時に読み込み　：
            - debian -> /etc/bash.bashrc
            - RedHat -> /etc/bashrc
        - ログアウト時に読み込み　：　なし
    - ユーザ側で個別設定
        - ログイン時に読み込み　　：
            - 1 . ~/.bash_profile
            - 2 . ~/.bash_login
            - 3 . ~/.profile
        - bash起動時に読み込み　　：　~/.bashrc
        - ログアウト時に読み込み　　：　~/.bash_logout

> bashのログイン時、環境設定ファイルを読み込む順番

```
「/etc/profile」2.「~/.bash_profile」3.「~/.bashrc」4.「/etc/bashrc」
```

設定ファイルについて

- /etc/profile
    - 変数
    - umask など
- ~/.bashrc
    - alias　など
- 設定ファイルをいじったあと、すぐ読み込みたければ「.」や「source」で実行

bash シェルのオプション設定

- bashシェルのオプション
    - emacs / vi
        - シェルのキーバインドをemacs/vi風に設定（どちらかのみ有効可能）
    - noclobber
        - 既存ファイルへの上書きリダイレクトを無効にする
    - noglob
        - ワイルドカードでのファイル指定を無効にする

- キーバインド
    - シェル上で実行する命令と割り当てられているキーとの組み合わせ
    - Ctrl+a などは、emacs風のキーバインドのおかげ

```
# bashシェルのオプション確認
set -o
set +o

# bashシェルのオプション設定
set +/- オプション名
```

6.2 シェルスクリプト

6.2.1 シェルスクリプトの実行

- ./スクリプト名 は読み取り権限と実行権限が必要
- そのほか は読み取り権が必要
- もちろんPATHを通せば、スクリプト名だけでスクリプトを実行できる

```
bash スクリプト名
source スクリプト名
. スクリプト
./ スクリプト
```

シェルスクリプトの冒頭

```
#!/bin/bash
```

-> **シバン**あるいは**シェバン(shebang)**と呼ぶ
-> スクリプトを解釈するインタープリタ(シェル)を指定

6.2.2 変数の利用

- 変数はシェル内でも定義できる
- 特殊変数もある

特殊変数

変数	表示するもの
$0	スクリプトファイル名
$1, $2...${10}...	引数($1は1つめの引数、$2は2つ目の引数)　10※以降は${10}

変数	表示するもの
$@	引数すべてをスペースで区切って出力
$*	引数すべてを$IFSで指定された値で区切って出力 (既定ではスペース)
$#	引数の数
$?	直前の処理の成否(成功は0、失敗は0以外)
$$	シェルのPID

6.2.3 制御構文の利用

- 制御構文

制御構文	意味
if	条件分岐
case	条件分岐 (多岐分岐)
for	繰り返し (値リスト)
while	繰り返し (条件指定)

- if文

```
if 条件式1
then
    処理1(=条件式1に合致した場合)
elif 条件式2
then
    処理2(=条件式2に合致した場合)
else
    処理3(=どの条件式にも合致しなかった場合)
fi
```

条件式（ファイルやディレクトリに関する条件式）

条件式	意味
-f　ファイル名	---> ファイルが存在
-d　ディレクトリ名	---> ディレクトリが存在
-e　ファイル / ディレクトリ名	---> ファイル / ディレクトリが存在
-r　ファイル / ディレクトリ名	---> ファイル / ディレクトリが読み取り可能

ファイルの更新日付による比較

条件式	意味

条件式	意味
ファイル1 -nt ファイル2	ファイル1の更新日付がファイル2より新しければ真(newer than)
ファイル1 -ot ファイル2	ファイル1の更新日付がファイル2より古ければ真(older than)

数値の条件式

条件式	意味	数式	数式の英語表記
値1 -eq 値2	値1と値2が等しい	num1=num2	equal
値1 -ne 値2	値1と値2が等しくない	num1≠num2	not equal
値1 -lt 値2	値1と値2より小さい	num1<num2	less than
値1 -le 値2	値1が値2以下	num1≦num2	less than or equal
値1 -gt 値2	値1と値2より大きい	num1>num2	greater than
値1 -ge 値2	値1が値2以上	num1≧num2	greater than or equal

その他論理式など

条件式	意味
! 条件式	条件を満たさない（NOT）
条件式1 -a 条件式2	条件1と条件2の両方を満たす（AND）
条件式1 -o 条件式2	条件1と条件2のどちらかを満たす（OR）
-n 文字列	文字列の長さが0ではない（文字列が存在する）

条件式の処理の結果

真偽	意味	終了ステータス
true	条件に合致している	0
false	条件に合致していない	0以外

文字列の条件式

条件式	意味
文字列1 = 文字列2	文字列1と文字列2が等しければ真
文字列1 != 文字列2	文字列1と文字列2が等しくなければ真

```
#!/bin/bash

if ls $1 2> /dev/null
then
```

```
    echo "success"
else
    echo "error"
    exit 2
fi
```

--- > 0の場合はthen以下を、0以外の場合はelse以下を実行する

```
#終了ステータスを定義
exit 終了ステータス
```

6.2.4 四則演算とcase文

- expr コマンド

```
expr 数値1 演算子 数値2

    足し算：+
    引き算：-
    掛け算：\*
    割り算：/
    剰　余：%
```

> ランダムな値が用意され、それを3で割った剰余によって運勢を占う

```
#!/bin/bash

NUM=$RANDOM
NUM=`expr $NUM % 3`
case $NUM in
    0) echo excellent! ;;
    1) echo good luck! ;;
    *) echo little luck! ;;
esac
```

- case 文

```
case 変数名 in
    値1) 値1だった場合の処理 ;;
        .
        .
    *) どの値にも合致しなかった場合 ;;
esac
```

6.2.5 繰り返し構文

- for 文

```
for 変数名 in 値1 値2･･･(値リスト)
do
    繰り返す処理
done
```

```
for fname in *.sh
do
    cp $fname ${fname}.bak
done
```

- while 文

```
while 条件式
do
    繰り返す処理
done
```

```
i=0
while [ $i -lt 5]
do
    echo $i
    i=`expr $i + 1`
done
```

6.2.6 関数の利用

- function 文

```
function 関数名(){
    処理
}
```

```
function convstr(){
    tr [:lower:] [:upper:]
}
case $1 in
    l) ls $2 | convstr;;
```

```
    c) cat $2 | convstr;;
    *) echo "usage: $0 l|c [filename]";;
esac
```

6.2.7 スクリプトのデバッグ

- bash実行時にデバッグ情報を出力するオプション
  - -v　：　実行するコマンドを出力
  - -x　：　実行するコマンドや変数への代入など、処理内容を出力
- シバン（#!/bin/bash）にオプションを加えるのも！それ以降はデバッグ情報を出力！
- setコマンドで囲むとその部分だけでバック

```
#!/bin/bash
set -x
i=o
set -x
 .
 .
 .
```

6.2.8 ntrapコマンドの利用

- trap
  - シグナルが送られてきた場合、指定した処理を実施

```
trap "処理" シグナル
```

```
trap "" 2 15
sleep 60
echo "wake up"
```

=== > Ctrl + C を押しても処理が中断されない

> 以下のtestコマンドを別の書式で記述したものはどれか。

```
test 条件式
  =
[ 条件式 ]
```

```
while test $NUM -le 3;
```

```
while [ $NUM -le 3 ];
```

> *の知られざる使い道 = その場でls

```
[root@localhost ~]# echo *
777 GCC_TEST anaconda-ks.cfg casetest,sh com dir1 file1 flaskAPP function.sh mm
nn.txt oneliner power test.git test.txt work
```