Linuxの学習メモ

教材：Udemy

#目次

セクション1 : Linuxのインストールとコマンド操作

- Linuxとは
- 学習用マシン Centos7001
  192.168.21.3
  root/rootroot

◎ショートカットキーを覚える

```
Ctrl A 行の先頭に移動する
Ctrl E 行の最後に移動する
Ctrl F カーソルを右に移動する
Ctrl B カーソルを左に移動する
Ctrl D カーソルの部分を削除する
Ctrl H カーソルの左の文字を削除する
Ctrl U カーソルの行頭までの文字を削除する
Ctrl K カーソルから行末までの文字を削除する
Ctrl W カーソル位置の単語を削除する
Ctrl Y 直前に削除した文字を貼り付けする
Ctrl L Clear
Ctrl S 画面をロックする
Ctrl Q 画面のロックを解除する
```

◎コマンドを覚える

```
【 man 】
      (f(d) : 1画面進む b(u) : 1画面戻る n : 下方向に検索 N : 上方向に検索)

【 date 】
        date '+%Y/%m/%d %H:%M:%S'
        2023/03/01 01:29:17
        date -d ‘1 day’ : 1日後
        date -d ‘-3 days’ : 3日前

【 ls / touch / mkdir / pwd / cd 】

【 vi 】
      A : カーソル位置の行の最後から入力できる
      O : 前に一行あけて入力を開始する
```

```
        o : 後ろに一行あけて入力を開始する
        d数値d : 数値分行を削除する
        yy : 行をコピーする
        p : コピーした行を張り付ける
        y数値y : 数値分行コピー

【 less 】
【 rm / mv / cp 】
        rm -rfi test
        rm: 通常ファイル `test' を削除しますか? Y
```

```
[root@localhost work]# cp -p cp_dir/ cp_dir_bk
cp: ディレクトリ `cp_dir/' を省略しています
[root@localhost work]# cp -pr cp_dir/ cp_dir_bk
[root@localhost work]# ls
cp_dir  cp_dir_bk
```

セクション2 : シェルスクリプトについて

◎サポートしているshの種類の確認方法

```
cat /etc/shells
/bin/bash
```

◎ファイルの作成方法

```
    * ファイル名の末尾に.shをつけ
    * 一行目にシェルの場所「#! /bin/bash」を指定する
    * コメントには#をつける
    * 処理を終了させる場合には、「exit 0」（0が正常終了。それ以外は異常終了）
    * 実行はパーミッションを755に変更しておく
```

◎スクリプト内での変数の定義

```
    var1=’変数’  変数の宣言
    var2=\`command`  コマンドの実行結果を変数に格納する
    var3=\$(command)  コマンドの実行結果を変数に格納する
```

◎配列の作成

```
fruits=('banana' 'apple' 'grape')
echo "\${fruits[@]}"  配列の中の値をすべて表示(banana apple grape)
echo "\${fruits[0]}"  インデックス0の要素を表示(1番目の要素(banana))
echo "\${!fruits[@]}"  インデックスを表示(012)
echo "\${#fruits[@]}"  配列の要素の数を表示(3)
fruits[3]='lemon'  配列のインデックス3にlemonを追加
unset fruits[2]  配列のインデックス2を削除
```

◎引数

```
echo $1 $2 $3 #第一引数、第二引数、第三引数を表示
echo $0 シェルスクリプトのファイル名
echo $@  全引数を表示
echo $#  引数の数を表示
  \$?  直前のコマンドの戻り値(成功の場合0,失敗の場合は0以外が入ってくる)
```

◎標準入出力、ファイル出力

```
* 標準入力：ターミナルからの値の読み取り。
* 標準出力；ターミナルでの値の表示。
 read var  標準入力
 echo var1 = $var  入力結果表示
 read var1 var2 var3  入力結果から3個の変数を設定
 echo var1 = $var1, var2 = $var2, var3 = $var3

 read -p 'var1:' var1  文字付の標準入力
 read -sp 'password:' password  シークレットモードでの標準入力
 read -a names  配列入力  （  echo ${names[@]}  で表示）
```

◎ファイル出力

```
>：上書き
>>：追記
```

◎if文

```
・書き方
 ①  if test 条件文;
 ②  if [ 条件文 ];
 ③  if [[ 条件文 ]];
```

```
if test 条件文;
then
    プログラム  #ifの条件文を満たすときに実行される

elif test 条件文;   #ifの条件文を満たしていない場合に実行される
then
    プログラム  #elifの条件文を満たす場合に処理が実行される
else
    プログラム  #ifもelifも満たさない場合に実行される
fi
```

```
* 文字列記号
=
!=
* 数値比較
-eq(=)
-ne(!=)
-lt(<)
-le(≦)
-gt(>)
-ge(≧)

【if文のand or】
* and
①   if [ 条件文1 ] && [ 条件文2 ];
②   if [ 条件文1 -a 条件文2 ];
③   if [[ 条件文1 && 条件文2];
④   if test 条件文1 && test 条件文2;
* or
①   if [ 条件文1 ] || [ 条件文2 ];
②   if [ 条件文1 -o 条件文2 ];
③   if [[ 条件文1 || 条件文2];
④   if test 条件文1 || test 条件文2;
* NOT
①   if !test 条件文
②   if [ ! 条件文 ]
③   if [[ 条件文 ]]
* コマンドの連結
cmd1 && cmd2  #cmd1が正しく実行されたらcmd2を実行
cmd1 || cmd2    #cmd1がエラーならcmd2を実行
cmd1 && cmd2 || cmd3    #cmd1が正しく実行されたらcmd2が実行されて、cmd1がエラーならcmd3を実行
```

◎【if文 ファイルの存在チェック】

```
if [ -e ファイル名 ];    #ファイルもしくはディレクトリの存在確認
        then
        else
fi
```

```
if [ -f ファイル名 ]  #ファイルが存在し、ディレクトリではない
if [ -d ファイル名 ]  #ディレクトリが存在するか
if [ -s ファイル名 ]  #ディレクトリ or 中身のあるファイルか
if [ -w ファイル名 ]  #書き込み権限があるか
if [ -xファイル名 ]  #実行権限があるか
if [ -fileA -nt fileB ]  #fileAがfileBより新しいか
if [ -fileA -ot fileB ]  #fileAがfileBより古いか
```

◎例題 exam1

問題文
・引数を2つ取ります
・1つめの引数が性別(masn,woman)、2つめの引数が年齢(age)
・第一引数がmanの場合、Manと表示、womanの場合Womanと表示
・第二引数が20未満の場合Child、20以上60未満の場合Adult、60以上の場合はElderlyと表示します
・標準出力はコロン（:）でつないで表示してください。例えばMan:Adult、Woman:Childと表示します
・以下の場合はexit 1で終了してください
  ・引数が二つでない場合
  ・一つ目の引数がman woman以外の場合「
  ・二つ目の引数が0以上でない場合

```
./exam1.sh man 20 => Man:Adult
./exam1.sh woman 70 => Womana:Elderly
./exam1.sh man 15 => Man:Child
```

◎例題2 exam2

問題文
・まず、現在いるディレクトリのファイル一覧を表示します
・2回、値を標準入力で受け付けて、変数に設定します
・1つめの入力がファイル名とし、2つめの入力は任意の値とします
・ファイルが存在する場合は、そのファイルに2つめの値を追記します
・ファイルが存在しない場合は、「ファイルが存在しません」と表示してください

◎シェルスクリプト基本 数値計算

```
* シェルスクリプトでの四則演算
 echo 1+1 # 1+1と表示されて、計算結果(2)が表示されない
 echo $(( 1 + 1 )) # 2と表示される +和 -差 *積 /商 %剰余
 echo $(expr 1 + 1) # 2と表示される +和 -差 \*積 /商 %剰余
* 変数を使った四則演算
 num1=1
 num2=2
 echo $(( num1 + num2 ))  +和 -差 *積 /商 %剰余
```

```
echo $(expr $num1 + $ num2 )  +和 -差 *積 /商 %剰余
echo $(expr $num1 \* $num2)
```

◎シェルスクリプト基本 数値計算2

```
* bcコマンドのインストール(yum install bc)
  bcコマンドは、任意の制度(小数点以下の桁数を指定)の数値を扱い、四則演算・平方根・
  三角関数など様々な数学関数を計算できるコマンドです。
  計算内容を文字列をして | を使って。bcコマンドに渡すと計算されます。

echo "20.5+5" | bc  #bcコマンドで文字列20.5+5を計算する
echo "20.5*5" | bc  #bcコマンドで文字列20.5*5を計算する
echo "scale=20;sqrt(20.5/5)" | bc -l
   # -lを利用すると標準数学ライブラリを読み込んでより複雑な数学関数を実行します
echo "$num1+$num2" | bc  #変数を使用しても、bcコマンドを利用できます
```

◎case

```
* ある変数を評価して、値に応じて処理をかえる

case $var in   #$varはチェックする変数
 パターン1 )
  プログラム1 ;;
 パターン2 )
  プログラム2 ;;
 * )
  プログラム3 ;;
esac

var=$1
case $var in
  'blue' )
     echo 'GO';;
  'red' )
     echo 'STOP';;
  'yellow' )
     echo 'WAIT';;
  *)
     echo 'Wrong Color';;
esac
```

◎while break continue

・while文はループ文の一種でwhileの条件を満たしている限り、中の処理を実行し続けます

```
while [ 文字列の比較 ] (( 数値の比較)) #nが10以下の場合にwhileの中の処理を実行
```

```
do
    プログラム1
    sleep 1 #1秒処理が止まる
done

# ファイルの読み込み(変数pの中にファイルの中身が1行ずつ読み込まれます)
while read p
do
    プログラム
done <aa.txt
cat aa.txt | while read p
do
 statement
done
```

・unitil分はループ文の一種でuntilの条件を満たすまで中の処理を実行し続けます

```
until [ 文字列の比較 ] (( 数値の比較 ))
do
    statement
done
```

break: while文、until文の中にbreakを入れるとそれが実行された時点でループの外に出ます

continue: while文、until文の中にcontinueを入れるとそこでループの移行の処理が実行されず、次のループに移ります。

```
read -p '文字列を入力してください：' var

while [ $var != 'exit' ];
do
   echo '入力した値は' $var
   echo 'exitで処理を終了'
   read -p '文字列を入力してください：' var
done
```

```
n=0

while (( $n < 10 ));
do
   echo $n
   sleep 1
   n=$(( n + 1 ))
done
exit 0
```

◎for break continue

- for文はループ文の一種で条件にある値を変数格納してループ内の処理を実行し続けます。

```
for var in 12345  #変数varに12345を代入していきます
```

```
for i in `seq 1 10`  # seq1 10 : 1~10までの整数
for i in `seq 1 2 10` #seq 1 2 10: 1 3 5 7 9
```

```
for command in ls pwd date; # 変数commandにls pwd dateを代入していきます
do
   echo $command
   $command
done
```

◎例題3 exam3

```
Fizz Buzz問題
・1-100までループする
・数値が3の倍数の場合は「数値:Fizz」と表示
・数値が5の倍数の場合は「数値:Buzz」と表示
・数値が15の倍数の場合は「数値:FizzBuzz」と表示
・上の条件に満たさない場合は、数値のみ表示
・for while until それぞれ使用して作成する
```

◎例題4 exam4

```
Fileの中身を計算する問題
以下のような数値を1列に並べたファイルを作成する
1
2
...

・ユーザから標準入力（ファイル名）を受け、ファイルの存在を確認する
・さらに標準入力を受け取り、その標準入力がsumの場合はファイルの合計を計算
・avgの場合は平均値うを計算
・minの場合は最小値
・maxの場合は最大値を計算
・exitで処理を終了する
（ただし、ファイルの中には1～190までの数値が入っており、必ず1つ以上の数値が入っていることは前提とする）
```

◎select

- selectを利用す r と複数の文字の中から、どれを選ぶのか数値で選択することができる。ユーザに数値を選択させて、それに応じて処理をさせたい場合に用いる。

```
select var in apple banana lemon  #ユーザが入力して、1)apple 2)banana 3)lemonの中から選択する (変数varに1の場合はapple/1,2,3以外の入力の場合空白が入る)

do
  echo $var
  breake #breakでselectの外に出る
done
```

◎function

- 関数を定義(function 関数名())

```
function hello(){
  プログラム
  echo$1
}
```

```
hello #関数の呼び出し(引数なし)
hello 'hello' #第一引数を'hello'で関数helloの呼び出し
```

```
#関数の中に同名の変数を変更すると関数の外でも変更されてしまいます。
function hello(){
        name=$1
        echo $name
}

name=Tom
echo $name
hello Mike
echo $name #nameの値がhello関数の中で変更されます(Mikeになる)

function hello(){
        local name=$1  #変数宣言の前にlocalを付けると変数を関数の中だけ専用利用できます。
}
```

```
function set_name(){
        local name=$1   #関数内だけはlocal nameが$1
        echo function: $name
}
name='Taro'
```

```
echo before: $name
set_name 'Hanako'
echo after: $name


#before: Taro
#function: Hanako
#after: Taro
```

◎readonly

```
var=31
readonly var #readonlyを設定
var=41 #値を変更するとエラーになる


function hello(){
        echo "hello world"
}
readonly -f hello #関数をreadonlyに設定
```

◎例題5 exam5

問題文
演習問題のsum,avg,min,maxを関数化しましょう

以下のような数値を1列に並べたファイルを作成する
1
2
・・・

・ユーザから標準入力（ファイル名）を受け、ファイルの存在を確認する
・さらに標準入力を受け取り、その標準入力がsumの場合はファイルの合計を計算
・avgの場合は平均値うを計算
・minの場合は最小値
・maxの場合は最大値を計算
・exitで処理を終了する
（ただし、ファイルの中には1～190までの数値が入っており、必ず1つ以上の数値が入っていることは前提とする）

◎例題6 exam6

問題文
ファイル操作するシェルの作成

list,delete,rename,show,exitの中から処理を選択（select）

・listの場合、ファイル一覧表示
・deleteの場合、削除するファイルを入力=>削除
・renameの場合、=>名前変更するファイルを入力=>変更する名前を入力=>ファイル名変更
・showの場合、表示するファイルを入力=>ファイルの中身表示
・exitの場合処理を終了する

◎pid , trap

例えば、以下の場合にシェルスクリプト内でプロセスIDを用いることがあります。

- 特定のプロセスを落とす
- プロセスIDをファイルに記憶する

```
pid is $$ #自分のプロセスID

trapコマンドでプログラム終了時の挙動を設定します

trap "echo exit command is detected" 0  #プログラム終了時に実行されるコマンド
trap "echo Exit" 15  #kill -15でのプロセス終了時に実行されるコマンド
trap "rm -f $file && echo file deleted" 0 2  #終了時、割り込み時に実行される

1：再起動
```

2：割り込み（ctrl+c）
9：強制停止（trapは設定できない）
15：プロセスの終了

```
function calcurate_sum(){
  sum=0
  while read p;
  do
    sum=$(( sum + p ))
  done < $1
  echo SUM: $sum
  exit 0
}
function calcurate_avg(){
  sum=0
  count=0
  while read p;
  do
    sum=$(( sum + p ))
    count=$(( count + 1 ))
  done < $fh
  echo AVG: $(( sum / count ))
  exit 0
}
function calcurate_min(){
  min=101
  while read p;
  do
    if [ $min -gt $p ]; then
      min=$p
    fi
  done < $fh
  echo MIN: $min
  exit 0
}
function calcurate_max(){
  max=0
  while read p;
  do
    if [ $max -lt $p ]; then
    max=$p
    fi
  done < $fh
  echo MAX: $max
  exit 0
}

read -p '入れて' fh

if [ -f $fh ]; then
  read -p 'sum avg min max exit' command
```

```
  if [ $command = 'sum' ]; then
    calcurate_sum $fh
  elif [ $command = 'avg' ]; then
    calcurate_avg $fh
  elif [ $command = 'min' ]; then
    calcurate_min $fh
  elif [ $command = 'max' ]; then
    calcurate_max $fh
  elif [ $command = 'exit' ]; then
    exit 0
  else
    echo 'そんなコマンドはない'
    exit 1
  fi
else
    echo 'ファイルがない'
fi
```

```
select command in "list" "delete" "rename" "show" "exit"
do
        echo $commnad in
        "list")
                ls;;
        "delete")
                read -p "delete file name" file_name
                if [ -f $faile_name ];
                then
                        rm $file_name
                fi;;
        "rename")
                ls
                read -p "rename file name" file_name
                read -p "new name" new_file_name
                if [ -f $file_name];
                then
                        mv $file_name $new_file_name
                fi;;
        "show")
                ls
                read -p "look" file_name
                cat $file_name;;
        "exit")
                bream;;
        esac
done
```

◎例題7 exam7

```

```

問題文
1-1000まで数字を1秒おきにファイル（ファイル名:output_プロセスID.txt）に記入するシェルを作成する。
start,stopを用いて実行、停止する。

1. 引数がstartの場合、ジョブを実行する。ただし、すでにジョブが動いている場合は'already running'とメッセージを送って終了する
2. 引数がstopの場合、ジョブを終了(kill -15)。ただし停止中の場合は'Not running'と表示する
3. 引数がstatusの場合、ジョブが停止中が起動状態を確認。プロセスIDとともに状態を表示する

◎debug

・デバッグモードで実行する方法

```
bash -x ./hello.sh
#!/bin/bash -x
```

・プログラム内で動的にデバッグにするか変更する方法

```
set -x  #デバッグモード
set +x  #もとに戻す
```