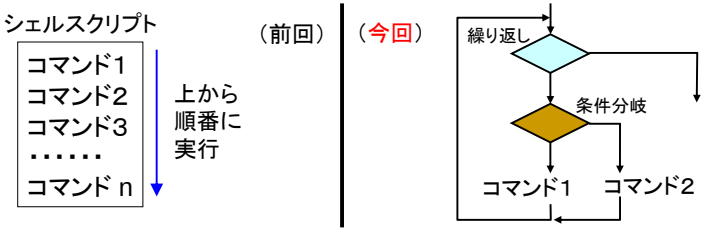
プログラミング言語 II
②フィルタの概念とシェルの基礎（その2）

阿萬裕久
aman@cs.ehime-u.ac.jp

シェルスクリプトをより便利に

- 前回はシェルスクリプトを使って，**複数のコマンドをまとめて実行**するという方法を学んだ



- 今回は，**繰り返し**や**条件分岐**といった，より進んだ使い方を学ぶ

繰り返し： for 文

（例）1, 2, 3, 4, a, b と順番に表示する

```
for x in 1 2 3 4 a b
do
  echo $x
done
```

変数 x へ
1，2，3，4，a, b
を**順番に代入**しながら
繰り返しを行う

do ～ done の間を繰り返し実行
（do, done ともに**前後に改行が必要**）

変数 x の値を echo コマンドで表示
参照時には $ マークを付けることに注意

for 文の構文規則

- for 文は以下のかたちで書かれる

```
for 変数名 in 値リスト
do
   繰り返し実行したいコマンド
   （複数行でもよい．
     変数は $変数名 で参照）
done
```

「値リスト」は
①**空白で区切って列挙**したもの：
1 2 3 4 a b
②**ワイルドカード**表現で書いたもの：
a*.c
どちらのタイプでもよい

「a」で始まり
「.c」で終わる
すべてのファイル

【例題１】

カレントディレクトリに置かれている**全ての C ソースファイル**（拡張子 .c）について，**そのコピー**を生成するシェルスクリプト makecopy.sh を作りなさい．ただし，コピーによって新たに作られるファイルの名称は，**元のファイル名の頭に "copy_" を**付けたものとする．

- 使用するコマンド
  - **cp**

【例題１】（答え）

```
for name in *.c
do
  cp $name copy_$name
done
```

（カレントディレクトリにある）**すべての C ソースファイルの名前**がリストになる

元の C ソースファイル名

新しい C ソースファイル名：**頭に copy_ をくっつけている**

※変数名は自由に決めてよい
（name でなくてもよい）

【例題２】

自分の**ホームディレクトリ以下に存在する全ての C ソースファイル**（拡張子 .c）に対し，その**名前の一覧を表示**するシェルスクリプト clist.sh を作りなさい．ただし，各ソースファイル名の**拡張子（末尾の .c）は取り除いて**出力すること．

- 使用するコマンド
  - **find**

【例題２】（答え）

```
list=$(find ~ -name "*.c")
for name in $list
do
  echo ${name%.c}
done
```

以下のスライドで順番に解説していく

- find コマンド
  ファイルを**サブディレクトリ以下まで含めて全て探索**する
- 変数の加工
  変数の値の一部を**パターンマッチで切り取る**

ホームディレクトリ以下に存在する全ての C ソースファイル

- ホームディレクトリは ~（チルダ）で表される
- ところが ls ~/*.c では不十分
  → サブディレクトリ以下が対象外であるため

（例）

◎ ~/foo.c
◎ ~/bar.c
× ~/aaa/hoge.c

「ls ~/*.c」で OKだが
こちらは対象外

サブディレクトリまで深掘りして探すには find コマンドが有功

find コマンド

- **サブディレクトリも含めてファイルの探索**を実行するコマンド
  - 通常，**ls コマンド**では，指定された**ディレクトリの直下のみ**が表示される （-R オプションを付ければ ls コマンドを再帰的に実行してくれるが，リスト生成には向かない）
  - まずは試しに `find ~` を実行してみるとよい
    → 一行に一件ずつ，ファイル名をフルパス表示
  - 特定のファイルに限定する場合は –name オプション：

```
find ~ -name "*.c"
```


コマンドの実行結果を変数へ代入

- シェルの中では，コマンドの実行結果をまるごと一つの変数に代入できる：$(と) で囲む

```
変数=$(コマンド)
```

※ = の前後に**空白は入れない**こと！

- したがって，ホームディレクトリ（～）以下にある全ての ＊.c ファイルをリストとして持たせるには

```
list=$(find ~ -name "*.c")
```


いったん答えを再確認すると

```
list=$(find ~ -name "*.c")
for name in $list
do
  echo ${name%.c}
done
```

① find コマンドで *.c ファイルだけをリストアップ

② それらを for 文で繰り返しながら，変数 name へ代入

③ echo $name ならば，そのまま表示となるが，少し加工が入っている（この後説明）

続いて，変数の値の加工が必要

（もう一度問題文を見ると）

自分のホームディレクトリ以下に存在する全ての C ソースファイル（拡張子 .c）に対し，その名前の一覧を表示するシェルスクリプト clist.sh を作りなさい．ただし，各ソースファイル名の**拡張子（末尾の .c）は取り除いて**出力すること．

- つまり，ファイル名が xxxxx.c の時，末尾の .c を削らなければならない
- よって，変数 name を**そのまま echo しても駄目**

変数の値の加工

※最長マッチングを行う場合はそれぞれ ## と %% を用いる

- 変数 name に対し，次の加工が可能：
  - $ {name#パターン}
    → name の**先頭から**「**パターン**」を削除
  - $ {name%パターン}
    → name の**末尾から**「**パターン**」を削除

（例）

name="/usr/bin/emacs"

echo $name ⟶ /usr/bin/emacs

echo $ {name#/*/} ⟶ /usr/bin/emacs

echo $ {name%e*} ⟶ /usr/bin/emacs

今回は，末尾の .c を削除したいので

```
list=$(find ~ -name "*.c")
for name in $list
do
  echo ${name%.c}
done
```

このように % を使用し，
パターンとして .c を与えている

【例題３】

コマンドライン引数として指定されたファイルについて，**それが存在すれば**その**ファイルの行数**を表示し，**存在しない場合は「見つかりません」と表示**するシェルスクリプト linecounter.sh を作りなさい．
ただし，行数のカウントには wc コマンドを利用すること．

- 使用するコマンド
  - **wc**

【例題３】(答え)

```
if [ -f $1 ]
then
   wc -l $1 | cut -d " " -f 1
else
   echo 存在しません
fi
```

- wc コマンド
  ファイルの行数等を表示する

以下のスライドで順番に解説していく

まずはファイル行数の出力について

- **wc** コマンドは，ファイルの**行数**，**単語数**，**バイト数**を数えてくれる：
  （例）「wc foo.txt」を実行

```
3  20  85  foo.txt
```

行数　単語数　バイト数

行数だけが欲しい場合は -l（エル）オプションを付ける：

```
$ wc -l foo.txt
3  foo.txt
```

あとは，この結果から一つ目の 3 だけを切り取ればよいので，切り取りには cut コマンドを使う

ゆえに，ファイルの存在判定以外は

```
if [ -f $1 ]
then
   wc -l $1 | cut -d " " -f 1
else
   echo 存在しません
fi
```

【存在しない場合】
「存在しません」と
echo で表示

【存在する場合】
wc ＋ cut で行数を表示

条件分岐：if 文

- if 文は以下のかたちで書かれる

```
if 条件
then
   条件成立時に実行したいコマンド
else
   条件不成立時に実行したいコマンド
fi
```

条件部分

- ファイルが存在するかどうか判定

[-f ファイル名]

  - 全体を [と] で囲む（前後に空白が必要）
  - -f がファイルの**存在チェック**を意味する
    正確には，ファイルが存在し，なおかつ，通常のファイルである（ディレクトリではない）ことを確認する

ファイルの有無で条件分岐

```
if [ -f $1 ]
then
    wc -l $1 | cut -d " " -f 1
else
    echo 存在しません
fi
```

コマンドライン引数で指定されたファイル（**$1**）が存在するかどうか

条件部分（その他）

- ファイル属性に関するその他の確認
  - A はディレクトリ？ -d A
  - A は B より新しい？ A -nt B

※その他，講義の Web サイトを参照

- 条件の組み合わせ
  - AND（&&）やOR（||）も可能

```
if [ 条件 ] && [ 条件 ]
then
  . . . . .
```


【例題４】

データがカンマで区切られた形式（CSV形式）のファイルがあるとき，そのファイルをタブ区切り形式に変換して出力しなさい．

- 使用するコマンド
  - sed

この例題で求められていること

- カンマ区切り形式のファイル

sample.csv

```
1,2,3
4,5,6
7,8,9
```

[あるコマンド] sample.csv

カンマをタブ文字に置き換えて出力

1	2	3
4	5	6
7	8	9

【例題4】(答え)

```
sed 's/,/¥t/g' < sample.csv
```

- sed コマンド
  ファイルの内容を一行ずつ加工して出力する

sed: 入力内容を加工して出力する

```
sed 's/,/¥t/g' < sample.csv
```

- 文字列の置換を意味する
- 変更前の文字列
- 変更後の文字列
- 置換を何度も繰り返すことを意味する(gを書かないと一行で一回だけ置換を行う)

(注意)シェルスクリプト内に書く場合

- sed コマンドをシェルスクリプトの中に書く場合は
  シングルクォーテーション(')を付けない

消すこと

```
sed 's/,/¥t/g' < sample.csv
```

（参考）sed, awk

- 従来から高度なフィルタを実現するために
  - sed
  - awk

  が活用されてきている
- これらを使うと，先の例のような置換だけでなく，さまざまな加工や計算も可能である
- ただ，本講義では同等以上のことが実現可能な Perl 言語について説明する（次回以降）ため sed, awk については割愛する

まとめ

- シェルスクリプトでの**繰り返し（for）**と**条件分岐（if）**の構文を紹介した
- **コマンドの実行結果を変数に代入**したり，**変数の値の加工**を行ったりといった使い方も紹介した
- 工夫次第で**シェルスクリプトは強力なツール**にもなる（例：多数のファイルをまとめて加工する等）
- 参考程度ではあるが，sed によるフィルタの実現についても紹介した