新学術「ゲノム支援」
平成25年度 情報解析講習会
シェルスクリプト

東京工業大学 大学院情報理工学研究科
瀬々 潤 (sesejun@cs.titech.ac.jp)

シェルスクリプトとは？

- **単なる，シェルのコマンドの羅列．それらを順に実行．**
- つまり「いつもの」操作を順番に書いておくと，計算機が同じ操作を行ってくれる
- もう少し便利に使うために「変数」「コマンドライン引数」が使える
- 明日学ぶスクリプト言語の様な，for文やif文が使える．（但し，簡易的なものなので，より**複雑なことはスクリプト言語で行う**のが吉）
- もっと高度なこともできるので，必要ならWebで検索してみてください．

なぜシェルスクリプトを学ぶのか？

- NGS解析の多くの操作は，シェルの上のコマンドを何度も繋げて行う
- 「定形」の解析パターンがあり，毎回毎回，何度も同じコマンドを打ち込む
  - リードのマッピング（アラインメント），形式の変更，等々
- 定形のパターンは成らずしも完全に一致しておらず，少しずつ異なる
  - マッピングのパラメータの変更など
- 待ち時間があって，コンピュータをイチイチ監視するのが面倒
  - 生物学的実験同様，1回1回の実験が，それなりに時間がかかる場合がある.

- シェルスクリプトで解決しよう！

はじめの一歩

- 「習うより慣れろ」の精神で，早速スタートしてみます
  - ここでは，シェルの一つ bash を利用して話を進めます
- 利用するのはコマンド２つ.
  - date: 時刻を表示するコマンド
  - sleep: 指定した時間(秒)を待つコマンド.
- まず，シェルにこれらのコマンドを打ってみましょう.

```
1 $ date
2 $ sleep 5
3 $ date
```

  - 今の日時 (date), 5秒間の空白，今の日時，が出たでしょうか？

コマンドからスクリプトへ

- 1行に何個も書くのは大変なので，ファイルに書きます．次の内容のファイルを作成してください．ファイル名を “current.sh” としましょう．

current.sh

```
1 #!/bin/bash
2 date
3 sleep 5
4 date
```

- その上で，以下のコマンドをシェルで実行して下さい．

```
$ chmod u+x current.sh
$ ./current.sh
```

- 今の時刻，5秒の空白，今の時刻．が表示されましたか？
- “`permission denied: ./current.sh`” が出る場合にはchmodが正しく動いているか確認．
  - `ls -al current.sh` で，実行ビットが立っているか

解説

・current.sh の中身

```
1 #!/bin/bash
2 date
3 sleep 5
4 date
```

bashを使う宣言(おまじない)
date コマンドを実行
5秒待機
date コマンドを実行

・コマンドラインの実行

```
$ chmod u+x current.sh
$ ./current.sh
```

実行ビットを立てる（直接実行可能にする）
実行する.

```
$ bash current.sh
```

代わりに，左のようにしても実行できます.

・まとめ

・シェルスクリプトはコマンドを（単に）順に記述したもの

変数を使う

・プログラムの中で頻繁かつ共通して使う数字や文字を変数にする.

current2.sh

```
1 #!/bin/bash
2 OPTION=“-u”
3 date ${OPTION}
4 sleep 5
5 date ${OPTION}
```

```
1 #!/bin/bash
2 date -u
3 sleep 5
4 date -u
5
```

・解説
  ・変数の設定は，変数名=値
    ・=の前後にスペースを入れない事.
    ・perl，python等に比べて融通が利かない
  ・変数の呼び出しは${ + 変数名 + }
    ・変数名の範囲が計算機にとって明らかな場合は {} を省略できる.
      予期しない動作を防ぐためにも，付けておくと安全.
・date の “-u”オプションは，時刻をUTCで表示するオプション

便利な変数の使い方

- 予め定義されている変数群があり，便利に使えます
  - $n は n番目の引数の値を示しています
  - “echo” は，引数を表示するコマンドです.

echo1.sh

```
1 #!/bin/bash
2 echo mouse
```

echo2.sh

```
1 #!/bin/bash
2 echo $1
```

```
$ ./echo2.sh mouse
```

- echo2.sh の引数を変更して実行しよう.
- シェルから変数を引き渡す事もできます.

echo3.sh

```
1 #!/bin/bash
2 echo ${SPECIES}
```

```
$ SPECIES=mouse ./echo3.sh
```

変数を組合せて変数が作れる

- 変数を設定する時に変数が使える
- 変数以外の部分は，単なる文字列として認識される
  - 下の例の場合，${SPECIES}と${VER}の間の_は，変数ではないので，文字
  - ${SPECIES}がmouse, ${VER}がmm10なので，mouse + “_” + mm10 となって，FILEにmouse_mm10 が入る
- echo で，この中身を表示している.

echo5.sh

```
1 #!/bin/bash
2 SPECIES=mouse
3 VER=mm10
4 FILE=${SPECIES}_${VER}
5 echo ${FILE}
```

```
$ ./echo5.sh
```

コメント

- #以降は，コメントとみなされます
  - 要するに，無視されます.
  - あとで，Univa Grid Engineで実行する時には，特殊な用途に利用しています
  - 後で何を書いたか分からなくならないように，コメントを利用してメモを残しておくと良いです.

echo4.sh 改

```
#!/bin/bash
SPECIES=mouse                   # SPECIES NAME
VER=mm10                        # GENOME VERSION
FILE=${SPECIES}_${VER}
echo ${FILE}
```

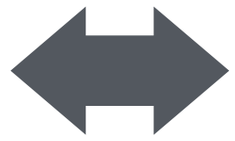
for文

- 繰り返して行いたい場合には，for文が便利
  - 例えば，全染色体に対して行いたい場合や，全ライブラリを走査する時

echo_chr1.sh

```
1 #!/bin/bash
2 for i in 1 2 X Y
3 do
4    echo ${i}
5 done
```

```
$ ./echo_chr1.sh
1
2
X
Y
```



```
#!/bin/bash
echo 1
echo 2
echo X
echo Y
```

- 2行目からがfor文です．for文は1行目の for ◯ in ～, 3行目のdo, 5行目の done で対を作っています.
- 2行目のin ～の～の中を順に変数◯に入れて，doからdoneの中を実行します
  - echo_chr.shでは，変数 i に1, 2, X, Y が順に入って実行されます.
  - echo $i の前は，”タブ” を入れてあります．入れなくても結果は代わりませんが，視認性を良くするために入れています.

2重のforループ

- 全染色体に対し，手持ちの全ライブラリを実行したい場合.
  - 全染色体 x 全ライブラリ，という2重ループが生じている.
- 人間の手で全てのパタンを記述するのは面倒なので，計算機に作ってもらう.

echo_chr2.sh

```
1 #!/bin/bash
2 for i in 1 2 X Y
3 do
4    for j in A B C
5    do
6       echo chr${i}_lib${j}
7    done
8 done
```

```
$ ./echo_chr2.sh
chr1_libA
chr1_libB
chr1_libC
chr2_libA
chr2_libB
```

配列 (1/2)

- for 文と併せて使うと便利な変数に配列があります
- 値の集合を表すことができます
- 配列の変数をaryとすると，aryのn番目の要素には${ary[n]}でアクセスができます．（注意：nは0から始まります）

echo_ary1.sh

```
1 #!/bin/bash
2 ary=(1 2 X Y)
3 echo ${ary[2]}
```

```
$ ./echo_ary1.sh
X
```

- 例えば，コマンドライン引数と組み合わせる事もできます

echo_ary2.sh

```
1 #!/bin/bash
2 ary=(1 2 X Y)
3 echo ${ary[$1]}
```

```
$ ./echo_ary2.sh 1
2
$ ./echo_ary2.sh 2
X
```

配列(2/2)

- 配列とfor文を組み合わせると，次のように，染色体番号とその長さを合わせる様な事ができます.

echo_ary3.sh

```
1 #!/bin/bash
2 chrs=(1 2 X Y)
3 scores=(50 35 25 25)
4 for i in 0 1 2 3
5 do
6    echo chr${chrs[${i}]}_len${scores[${i}]}
7 done
```

```
$ ./echo_ary3.sh
chr1_len50
chr2_len35
chrX_len25
chrY_len25
```

コマンドの結果を変数に入れる

- 「`」で囲うことで，コマンドの実行結果を変数に入れることができます
  - 日付を変数に入れる
  - ls の結果を入れる（配列になります）

echo_date.sh

```
1 #!/bin/bash
2 d=`date`
3 echo ${d}
```

```
$ ./echo_date.sh
```

echo_scripts.sh

```
1 #!/bin/bash
2 FILES=`ls *.sh`
3 for i in ${FILES}
4 do
5    echo ${i}
6 done
```

```
$ ./echo_scripts.sh
```

シェルスクリプトの結果は ファイルに書き出せますし, パイプにも渡せます

- つまり，新しいコマンドを自分で作った感じになります.
  - 沢山「おれおれ」スクリプトを作って，自分にとって便利な環境を構築しましょう

```
$ ./echo_ary3.sh
chr1_len50
chr2_len35
chrX_len25
chrY_len25
$ ./echo_ary3.sh > chrlist.txt
$ cat chrlist.txt
chr1_len50
chr2_len35
chrX_len25
chrY_len25
```

パイプの代わりにスクリプト

- パイプ (|) は，非常に便利なものですが，慣れないと使いにくいと思います.
- パイプは結局，コマンド > ファイル出力 → コマンド + 出力ファイル のつなぎなので，スクリプトで書くこともできます.

```
$ grep test.fastq | wc -header.txt
```

```
#!/bin/bash
grep test.fastq > header.txt
wc -l header.txt
```

if-then-else 文

- もし〜ならば (if), 〜する (then). さもなくば〜する(else) が使えます.
- ただ, 少し使いにくいので, 詳しい使い方は Webのドキュメントなどに譲ります.
- NGS解析でよく使うものとして「ディレクトリが無かったら作る」という例を挙げておきます.
  - ”logs”というディレクトリが無かったら, 作成します. あったら, ”logs dir exists” と表示します.

```
#!/bin/bash
if ! test -e "logs"
then
   echo mkdir logs
   mkdir logs
else
   echo logs dir exists
fi
```

まとめ

- シェルスクリプトは，コマンドの羅列
- 新しい自分専用コマンドを作ることができる.
- 変数とfor文を使って，本質的なQuestionに集中したNGS解析をしましょう.

参考

- UNIX & Linux コマンド・シェルスクリプト リファレンス
  - http://shellscript.sunone.me/