

シェルスクリプト

26, Feb. 2016

ゲノムインフォマティクストレーニングコース 準備編
基礎生物学研究所 情報管理解析室
西出 浩世 hiroyo@nibb.ac.jp

作業場所

- 以降の作業は、以下のディレクトリで行います。

```
~/data/5_script/
```

cd コマンドを用いてディレクトリを移動し、

pwd コマンドを利用して、カレントディレクトリが上記になっていることを確認してください。

シェルスクリプトとは？

- あらかじめ実行したいコマンドを記述しておいたテキストファイル
- 実行権を持たせることによってシェルから実行できる。
- 何度も繰り返し実行するコマンドを記述しておくことによって、作業を簡略化できる。
- 作業の記録にもなる。

シェル

- ユーザーに対してカーネルへの操作機能を提供するためのソフトウェア
- シェルには bash, tcsh など様々な種類がある。
- OSXでのシェルはデフォルトで bash
- 現在のシェルの確認方法

```
$ echo ${SHELL}
```

シェルスクリプト

- 一連のコマンドを記述したファイル
 - スクリプトファイル、バッチファイル
- 実行
 - ファイルを単体で実行するには実行権が必要

```
$ chmod +x testpg
```
 - 実行はパスを省略せずに入力

```
$ ./testpg
```
 - コマンドパスの通ったディレクトリに保存すれば、コマンド名のみで実行できる。

シバン (shebang)

- シェルスクリプト先頭の行にある記述

```
#!/bin/sh
```

- 「#!」 をシバンと呼び、スクリプトを実行するプログラムを指定する

```
Perlスクリプトの一行目に書いてあるシバンの例  
#!/usr/bin/perl
```

- bashはshと同じものと考えて慣例的に#!/bin/shと書くことが多い (#!/bin/bash でなく)
- スクリプトファイルの拡張子も慣例的に「.sh」とすることが多い

コメント

- スクリプト内で、行頭に # をつけると、それ以降はコメントとして扱われる。

```
# this is comment.
```

- コメント内に有効なコマンドを書いても、実行時には何もしない。
- 前述のシバンは特別に、コメントにはならない。

シェルスクリプト例

- 実際にシェルスクリプトを見てみよう
(bowtie2用index作成, bowtie2の実行)

```
$ less ex1.sh
```

```
#!/bin/sh
bowtie2-build -f ecoli_genome.fa ecoli_genome
bowtie2 -x ecoli_genome -U ecoli.1.fastq
        -S ecoli.1.sam
```

```
$ ./ex1.sh
```

と入力し、このシェルスクリプトを実行し、`$ ls` で
ecoli.1.sam等 ができていることを確認

変数

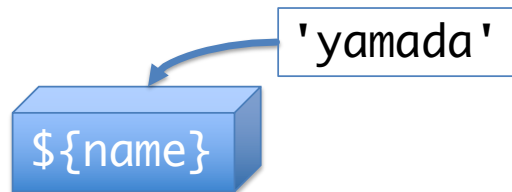
- シェルでは「変数」が使用できる。
- 任意でつけた名前の「変数」に、プログラムの実行状況によって、文字列や数値などさまざまなデータを記憶しておく。

```
name=yamada
```

変数 name に 'yamada' という文字列（値）を記憶させる。
変数、イコール、入れる値の間にスペースは入れない。

```
echo ${name}
```

変数の中の値を呼び出すには、\$を使う。
\$のあとに続く文字列が変数であることを示すため、
{ } を使用して区別するとよい。



クォーテーションの違い

- シングルクォーテーション [']
中身は単に文字列として扱われる。
スペースを含む文字列を使用する時に有効。
- ダブルクォーテーション ["]
中身に変数がある場合、シェルはその中の値を採用する。

```
name='yamada'
```

```
echo '${name}'
```

`${name}` と表示される。

```
echo "${name}"
```

yamada と表示される。

シェルスクリプトの編集



- テキストエディタ「mi」を使う
 - ファイル→開く... ex1.sh を開く
- スクリプトの作成・編集にWord等を使わないこと
 - 書式情報などがあるファイルでは正常に動かない
 - ワードプロ：フォントや文字サイズ、レイアウトまで加工（= process）保存や印刷も
 - エディタ：文字（テキスト）のみのファイルを使って、文章だけを作成・編集

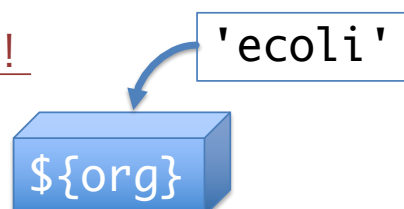
変数の使用

- ex1.shを変数を使用するスクリプトに書き換え

```
#!/bin/sh
org='ecoli'
bowtie2 -x ${org}_genome -U ${org}.1.fastq
        -S ${org}.1.sam
```

ex2.sh

- 書き換えたら「ファイル→別名で保存」
- 名前を「ex2.sh」として保存
- bowtie2のコマンドは一行で書く！



編集したシェルスクリプトを実行

- ターミナルに戻る
- ex2.sh に実行権を付与してから実行

```
$ chmod +x ex2.sh
```

```
$ ./ex2.sh
```

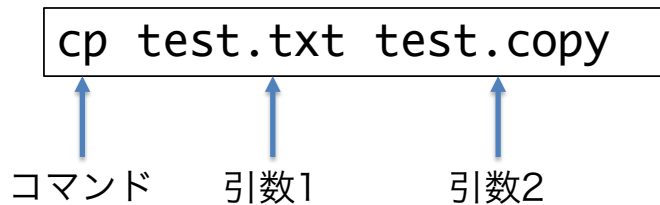
- アプリケーション間を移動する際は常にディレクトリに注意すること
 - 保存したディレクトリと別の場所を見ている意味がない

変数と引数

- 先ほどの例では、変数に入れる値としてecoliという文字列を使用した。
- しかし、別の文字列を使用したい場合、毎回スクリプトを直さないといけない。
- そのような手間を省くために「引数」を使用することもできる。

引数

- 引数とは、コマンド実行時にコマンドラインから渡される値である。



- シェルスクリプトの中で、受け取った引数を利用できる。n番目の引数は `${n}` に設定される

<code>\${1}</code>	1 番目の引数	<code>\${2}</code>	2 番目の引数	...	<code>\${9}</code>	9 番目の引数
<code>\${0}</code>	コマンド自身	<code>\${*}</code>	引数全て			(等々)

引数の使用

- 指定するファイル（生物種）名を引数として受け取るスクリプト

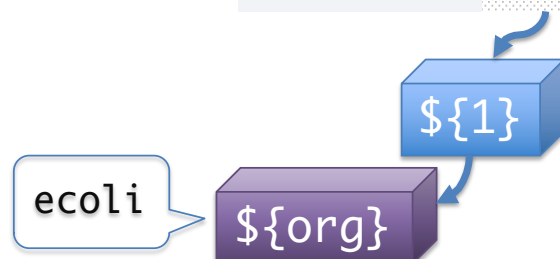
```
$ less ex3.sh
```

```
#!/bin/sh  
org=${1}  
bowtie2 -x ${org}_genome -U ${org}.1.fastq  
      -S ${org}.1.sam
```

```
$ ./ex3.sh ecoli
```

↑
1 番目の引数

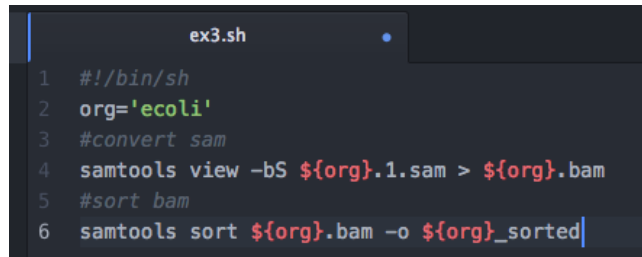
```
$ ./ex3.sh ecoli
```



スクリプト作成に便利なエディタ

Atom

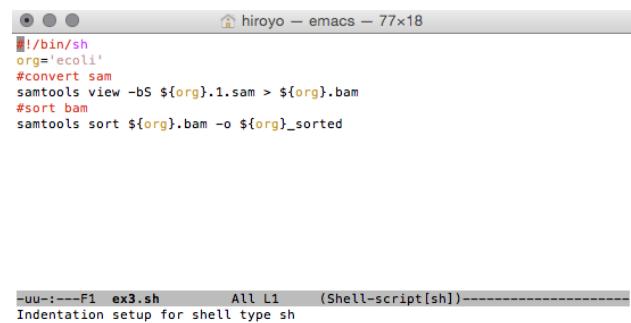
- GUIで使う
 - Atom
 - <https://atom.io/>
 - vi
 - <http://www.mimikaki.net/>



```
ex3.sh
1 #!/bin/sh
2 org='ecoli'
3 #convert sam
4 samtools view -bS ${org}.1.sam > ${org}.bam
5 #sort bam
6 samtools sort ${org}.bam -o ${org}_sorted
```

emacs

- コマンドラインで使う
 - emacs
 - vi



```
#!/bin/sh
org='ecoli'
#convert sam
samtools view -bS ${org}.1.sam > ${org}.bam
#sort bam
samtools sort ${org}.bam -o ${org}_sorted
```

シェルスクリプト応用編： 複数ファイルをまとめて処理する

- 条件違いの同じフォーマットのデータが大量にあるとき
- 一件ずつコマンドを実行するのは大変
 - ヒューマンエラーの元にもなる
- 複数ファイルに対し繰り返し処理をしてくれるシェルスクリプトの書き方

for 文

- ex4.sh の内容を確認
- 実行してみる

```
$ less ex4.sh
```

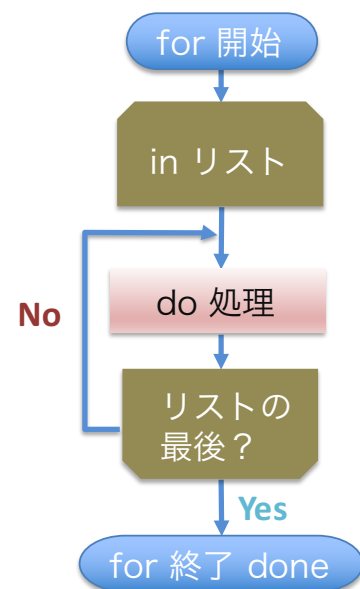
```
#!/bin/sh
for file in ./*.fastq
do
    echo ${file}
done
```

```
$ ./ex4.sh
```

繰り返し処理：for文

```
for 変数名 in 文字列などのリスト
do
    処理
done
```

- 文字列やファイルのリストに対し、順番に「do」と「done」の間に書かれた処理を実行する
- リストはスペース区切りの文字列挙、配列、数字など
- for 後の変数にリストの値が順番に1つずつ代入され、その後に do – done 間の処理が行われる
- リストの値全てが代入され終わったらfor文も終了



for文に使えるリストの例

```
for file in ./*  
do...  
done
```

カレントディレクトリ内にある全てのファイル名をワイルドカード「*」を使ってリスト（変数 `${file}` にファイル名が1つずつ代入される）

```
for i in 1 2 3 4 5 6 7  
do...  
done
```

1～7までの整数をリスト（変数 `${i}` に1～7が順に代入される）

```
for i in {1..10}  
do...  
done
```

1～10までの整数をリスト（変数 `${i}` に1～10が順に代入される）

for文の作成

- mi 「ファイル → 開く... → ex3.sh」
- 以下のスクリプトに改変し、ex5.sh として保存
- 実行権を与えてから、引数「ecoli」を与え実行してみる

```
#!/bin/sh  
org=${1}  
for i in 1 2 3  
do  
    bowtie2 -x ${org}_genome -U ${org}.${i}.fastq  
            -S ${org}.${i}.sam  
done
```

```
$ chmod +x ex5.sh  
$ ./ex5.sh ecoli  
$ ls
```

if 文

- ex6.sh の中を確認し、実行
- bowtie2が実行できたかの簡単なチェック

```
$ less ex6.sh
```

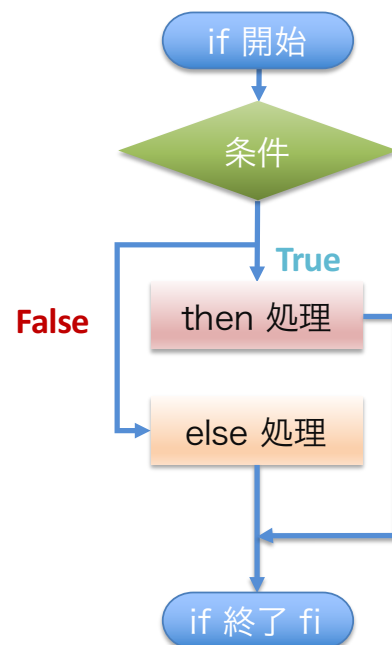
```
#!/bin/sh
if [ -f ecoli.3.sam ]
then
    echo 'ok'
else
    echo 'not ok'
fi
```

```
$ ./ex6.sh
```

条件分岐：if 文

- [] 内の条件が真か偽か？で処理を変える

```
if [ 条件 ]
then
    条件が真だった場合の処理
else
    偽だった場合の処理
fi
```

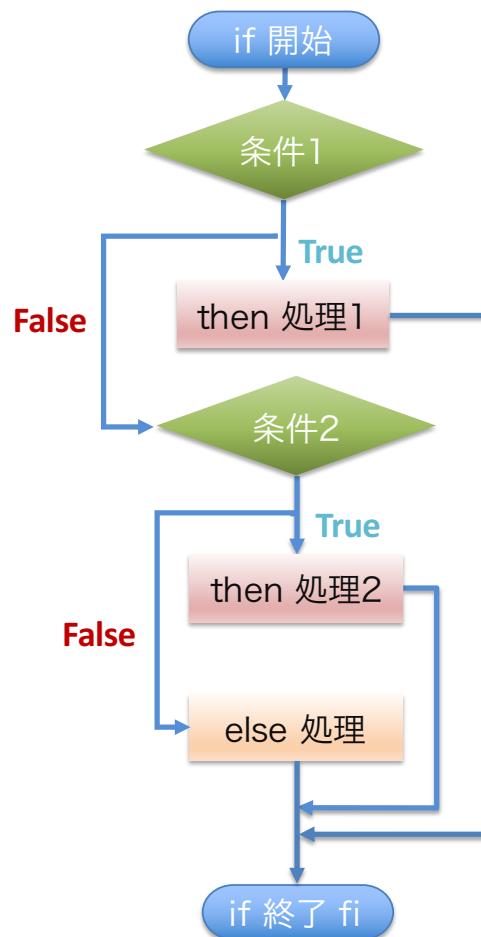


- if → 条件 → then or else → fi
- 「if」は必ず「fi」で終わらねばならない
- 条件を複数設定したい場合は「elif」を使う

条件分岐：if 文

- 複数の条件を設定
： elif

```
if [ 条件1 ]
then
    処理
elif [ 条件2 ]
then
    処理
elif [ 条件3 ]
then
    処理
else
    全て偽な場合の処理
fi
```



if：条件判断の演算子

- 条件判断の [] は、test コマンドの代替表現
- 下記の演算子一覧は man test で見ることができる

数値比較	
数1 -eq 数2	両辺が等しいと真
数1 -ne 数2	両辺が等しくないで真
数1 -gt 数2	数1 > 数2 の場合に真
数1 -lt 数2	数1 < 数2 の場合に真
数1 -ge 数2	数1 >= 数2 の場合に真
数1 -le 数2	数1 <= 数2 の場合に真

論理結合	
! 条件	条件が偽であれば真
条件1 -a 条件2	条件1, 2 共真であれば真
条件1 -o 条件2	1, 2 どちらかが真であれば真

文字列比較	
-n 文字列	文字列の長さが0でなければ真
! 文字列	文字列の長さが0なら真
文字列1 = 文字列2	両文字列が同じなら真
文字列1 != 文字列2	両文字列が同じでなければ真

ファイルチェック	
-d ファイル名	ディレクトリなら真
-f ファイル名	通常ファイルなら真
-e ファイル名	ファイルが存在すれば真
-L ファイル名	シンボリックリンクなら真
-r ファイル名	読み取り可能ファイルなら真
-w ファイル名	書き込み可能ファイルなら真
-x ファイル名	実行可能ファイルなら真
-s ファイル名	サイズが0より大きければ真
ファイル名1 -nt ファイル名2	1が2より新しければ真
ファイル名1 -ot ファイル名2	1が2より古ければ真