# C シェルスクリプトを使った一括処理

## 一括処理とは

複数の命令や手順（まずこれをやって次にこれをやって... という手順）を１つのファイル（”指示書”に喩えられる）に記述して、一回の指示で UNIX に処理をお願いすること。

処理しなければいけないデータファイルが膨大にあるとき、UNIX にリアルタイムに指示を出していたらたいへんである（１つ目のデータの処理を UNIX に指示して、終わるのをじっと待ち、終わったら２つ目のデータの処理を UNIX に指示して、また待ち、、、以降データの数だけ続く）。これに対して、必要な指示を全てあらかじめ”指示書”に記述しておけば、UNIX への指示は１回だけで済む。後は全データが処理されるのを他の仕事をしながら待てばいい。

ここで採り上げるシェルスクリプトも、上で”指示書”と喩えて言っているものの１種である。シェルスクリプトにはいくつか種類があるが、ここでは特に C シェルスクリプトを採り上げる。

## この文書の構成

基本的にチュートリアル形式で進める。後で実践的にゲノムアノテーションに取り組む際に役に立つような実用的な C シェルスクリプトの例も挙げることにする。C シェルスクリプトの記述方法の中でも特に役に立つ場面の多い foreach ループの使い方を集中して採り上げる。付録としてこの文書の最後に C シェルスクリプトの基本をまとめておく。

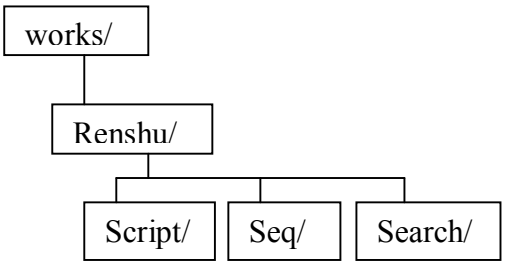
## 目次

## チュートリアル

### （0）準備

作業用のディレクトリを以下のように作る。

```
cd ~/works
mkdir Renshu
cd Renshu
mkdir Script Seq Search
cd Script
```




以下、~/works/Renshu/Script/の中で C シェルスクリプトを作成していく。

### （1）C シェルスクリプトの作成から実行までの流れ

① エディター（vi など）を使ってファイル（例：test.csh）を作成する。

```
vi test.csh
```

test.csh の中身の例：

```
1  #!/bin/csh -f
2
3  echo "Hello."
```

② ファイルの実行権を有効にする（実行可能にする）。

```
chmod u+x test.csh
```

③実行する。

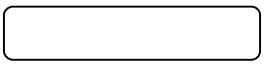
```
./test.csh
```

実行結果の例：

```
Hello.
```

---

［脚注］図の見方：


←ターミナル・ウィンドウに打ち込むコマンド


←ターミナル・ウィンドウに表示される文字

ファイルの行番号→ 1 2 3 
←ファイルの中身

## （2）foreach ループ　その１　（リストを明示する）

複数の人にハローと言うスクリプトを作る。
まず、ループを使わないと以下のようになる。

hello1.csh の中身：

```
#!/bin/csh -f

echo "Hello, Ai."
echo "Hello, Mai."
echo "Hello, Mami."
```

実行結果：

```
Hello, Ai.
Hello, Mai.
Hello, Mami.
```

同じことを今度は foreach ループを使ってやる。

hello2.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach name (Ai Mai Mami)
4     echo "Hello, $name."
5  end
```

変数 name に値（Ai, Mai, Mami の３つ）を１つずつ入れながら、foreach ~ end に挟まれた行（４行目）のコマンド（echo ~）を実行していく。

書式の注意として、foreach の横に置く変数名には$をつけてはいけない。３行目の name と４行目の$name は、$の有無で一見違って見えるが、C シェルスクリプトの書式の上では、同一の変数と見なされる（この辺り perl の書式とは異なる）。

**練習 1**：　ちゃん付けで挨拶してみよう。

hello2.csh の４行目の$name を${name}chan に置き換えればよい。この場合、どこまでが変数名なのかを示すために{}が必要である。

**実用的課題 1**：　データベース swissprot からコマンド fastacmd を使って、以下の *ID 名*それぞれの配列を取得し、それぞれ別々のファイルとして保存せよ。ファイル名は *ID 名*.fa とせよ（例：O94763.fa）。

O94763　　P10516　　P49758

get_seq1.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach id (O94763 P10516 P49758)
4     echo $id
5     fastacmd -s $id -d ~/works/DB/swissprot -o $id.fa
6  end
```

## （3）foreach ループ　その２　（ファイルからリストを読む）

引き続き複数の人にハローと言うスクリプトを作る。
今度はあらかじめハローと言いたい人の名前をファイルに保存しておく。

name.list の中身：

```
Daisuke
Takuya
Naoki
```

ファイル name.list から名前のリストを読み込んでひとりひとりにハローを言う。

hello3.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach name (`cat name.list`)
4     echo "Hello, $name."
5  end
```

前にやった hello2.csh と比較して、３行目の ( `cat name.list` ) のところが違うだけである。cat name.list は、ファイル name.list の中身を出力せよというコマンドである。コマンドを ` （バッククォートという記号）で囲んで書くと、コマンドの出力結果を取り込むことができる（foreach name ( ) の ( ) のところに取り込まれる　→　結果として foreach name (Daisuke Takuya Naoki)と書いた場合と同じになる）。

**実用的課題 2**：（２）の実用的課題 1 と同じ問題を、上で示した C シェルスクリプトの書き方でやってみよ。今回はファイルをディレクトリ../Seq/ の中へ作成せよ。

id.list の中身：

```
O94763
P10516
P49758
```

get_seq2.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach id (`cat id.list`)
4     echo $id
5     fastacmd -s $id -d ~/works/DB/swissprot -o ../Seq/$id.fa
6  end
```

## （4）foreach ループ　その３　（*を使ってリストを作る）

まず以下の UNIX のコマンドを実行してみよう。

```
ls ../Seq/*.fa
```

実行結果：

```
../Seq/O94763.fa ../Seq/P10516.fa
../Seq/P49758.fa
```

このように*の記号（ワイルドカードと言う）を使うと複数のファイルを同時に指定できる（*のところは何でもよく、とにかく最後の３文字が .fa であるファイルを、ディレクトリ../Seq/の中から探す（UNIX が探してくれる））。これを　foreach ループと組み合わせて使ってみよう。まずは単純にファイル名を表示してみる。

list.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach file (../Seq/*.fa)
4      echo $file
5  end
```

実行結果：

```
../Seq/O94763.fa
../Seq/P10516.fa
../Seq/P49758.fa
```

次に４行目だけ少し修正してみる。こうするとディレクトリ名が省略される。

list.csh の中身の一部：

```
4      echo ${file:t}
```

実行結果の一部：

```
O94763.fa
```

今度は以下のように修正してみる。拡張子（今の場合、.fa）も省略される。

list.csh の中身の一部：

```
4      echo ${file:t:r}
```

実行結果の一部：

```
O94763
```

最後に以下も試してみる。ディレクトリ名はそのままで拡張子だけ省かれる。

list.csh の中身の一部：

```
4      echo ${file:r}
```

実行結果の一部：

```
../Seq/O94763
```

**練習 2**：　ディレクトリ ../Seq/ 内のファイル（*.fa）それぞれの１行目だけを表示。

entry_name.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach file (../Seq/*.fa)
4      head -1 $file
5  end
```

おまけ：　ID の部分だけを抜き出したい場合は４行目を以下のように修正する。

entry_name.csh の中身の一部：

```
4    head -1 $file | awk '{print $1}' | sed 's/^>//'
```

**練習 3**：　ディレクトリ ../Seq/ 内のファイル（例：O94763.fa）それぞれについて、１行目だけを抜き出して新しいファイル（例：O94763.name）に保存せよ。保存先はカレントディレクトリ（./）とせよ。

entry_name2.csh の中身：

```
#!/bin/csh -f

foreach file (../Seq/*.fa)
   head -1 $file > ./${file:t:r}.name
end
```

**実用的課題 3**：　ディレクトリ ../Seq/ 内のファイル（例：O94763.fa）それぞれを１つずつクエリーとして、データベース pdbaa に対して blastp サーチを実行し、サーチ結果をクエリーごと別々のファイル（例：O94763.out）に保存せよ。保存先はディレクトリ ../Search/ とせよ。

do_blast.csh の中身：

```
1  #!/bin/csh -f
2
3  foreach query (../Seq/*.fa)
4     echo $query
5     blastall -p blastp -i $query -d ~/works/DB/pdbaa
                -o ../Search/${query:t:r}.out
6  end
```

（注意：紙面の都合上、５行目が２行に分かれているが、１行で書くこと）

バックグラウンドで実行するやり方の例：

```
./do_blast.csh >& do_blast.out &
```

## （５）その他　（変数と if の利用および foreach との組み合わせ）

### *（５−１）変数の利用（set の利用）*

必ずしも必要ないが変数を使うとスクリプトの記述量が減ったり、見やすくなったり、さらにそのおかげで改造しやすくなったりする。以下は変数を使ってみた例である。このスクリプトは、１つの配列（ID は P55160）を fastacmd で swissprot から取得して、つづけてそれをクエリーとして pdbaa に対して blast 検索を実行するものである。

example5_1.csh の中身：

```
#!/bin/csh -f

set swissprot = ~/works/DB/swissprot
set pdbaa     =  ~/works/DB/pdbaa

set id = P55160
set seq      =  ../Seq/$id.fa
set blastout =  ../Search/$id.out

echo $id
fastacmd -s $id -d $swissprot -o $seq
blastall -p blastp -i $seq -d $pdbaa -o $blastout
```

上でやったことを、複数の配列について実行したいときは、以下のようにする。

do_blast2.csh の中身：

```
#!/bin/csh -f

set swissprot = ~/works/DB/swissprot
set pdbaa     =  ~/works/DB/pdbaa

foreach id ( P55160 O15240 )
    set seq      =  ../Seq/$id.fa
    set blastout =  ../Search/$id.out

    echo $id
    fastacmd -s $id -d $swissprot -o $seq
    blastall -p blastp -i $seq -d $pdbaa -o $blastout

end
```

ここで、６行目を、チュートリアル（３）の実用的課題２で行ったのと同じように、あらかじめ ID のリストのファイル（id.list）を作成しておき、foreach id (`cat id.list`)と記述するやりかたもある。

### *（5－2）if の利用*

何度も同じ配列をクエリーとして blast を実行するのは無駄なので、すでに blast 検索済みの配列かどうかを確認して（blast の結果ファイルがあるかどうかを確認して）、検索済みなら何もせず、まだ未検索なら blast を実行するようにしたい。以下のようにする（これは example5_1.csh をコピーして緑色の２行を付け加えたものである）。

example5_2.csh の中身：

```
1  #!/bin/csh -f
2
3  set swissprot = ~/works/DB/swissprot
4  set pdbaa     =  ~/works/DB/pdbaa
5
6  set id = P55160
7  set seq      =  ../Seq/$id.fa
8  set blastout =  ../Search/$id.out
9  if ( ! -e $blastout ) then
10    echo $id
11    fastacmd -s $id -d $swissprot -o $seq
12    blastall -p blastp -i $seq -d $pdbaa -o $blastout
13 endif
```

すでに（５－１）で P55160 の blast を実行し終えているなら、この example5_2.csh を実行しても何も起こらずにすぐに終了する（プロンプトがもどってくる）であろう。その場合は、一度、blast の結果ファイル、../Search/P55160.out を削除して（rm ../Search/P55160.out を実行して）から example5_2.csh を実行し、挙動の違いを確かめてみよ。

9 行目以降を以下のようにしてもよい（論理的には同じこと）。

```
9  if ( -e $blastout ) then
10    echo "$id : already done"
11 else
12    echo $id
13    fastacmd -s $id -d $swissprot -o $seq
14    blastall -p blastp -i $seq -d $pdbaa -o $blastout
15 endif
```

上でやったことを、複数の配列について実行したいときは、以下のようにする。

do_blast3.csh の中身：

```
1  #!/bin/csh -f
2
3  set swissprot = ~/works/DB/swissprot
4  set pdbaa     =  ~/works/DB/pdbaa
5
6  foreach id ( P55160 O15240 )
7      set seq      =  ../Seq/$id.fa
8      set blastout =  ../Search/$id.out
9      if ( ! -e $blastout ) then
10         echo $id
11         fastacmd -s $id -d $swissprot -o $seq
12         blastall -p blastp -i $seq -d $pdbaa -o $blastout
13     endif
14 end
```

ここで、6行目を、チュートリアル（3）の実用的課題２で行ったのと同じように、あらかじめ ID のリストのファイル（id.list）を作成しておき、以下のように foreach id (`cat id.list`)と記述してもよい。

do_blast4.csh の中身：

```
1  #!/bin/csh -f
2
3  set swissprot = ~/works/DB/swissprot
4  set pdbaa     =  ~/works/DB/pdbaa
5
6  foreach id ( `cat  id.list` )
7      set seq      =  ../Seq/$id.fa
8      set blastout =  ../Search/$id.out
9      if ( ! -e $blastout ) then
10         echo $id
11         fastacmd -s $id -d $swissprot -o $seq
12         blastall -p blastp -i $seq -d $pdbaa -o $blastout
13     endif
14 end
```

**おまけ１：**　blast の結果ファイルからの情報抽出（grep と awk の活用）

i) 検索で釣れた配列のサマリー情報を抽出

例
```
grep "^pdb" ../Search/P49758.out
```

（この例では、pdbaa に対して blast 検索した結果のファイル P49758.out を例としているので、^pdb という記述で抽出可能となっているが、たとえば nr に対して blast 検索した場合は、^gi 等の記述にする必要がある。）

ii) さらに ID（第１列）だけを抽出

例
```
grep "^pdb" ../Search/P49758.out | awk '{print $1}'
```

iii) ID に加え、E 値（最右列）も抽出

例
```
grep "^pdb" ../Search/P49758.out | awk '{print $1,"¥t",$NF}'
```

iv) E 値 < 1e-4 を満たすものだけを抽出

例
```
grep "^pdb" ../Search/P49758.out | awk '$NF<1e-4 {print}'
```

例
```
grep "^pdb" ../Search/P49758.out | awk '$NF<1e-4 {print $1}'
```

**おまけ２：**　PDB のファイルをインターネット越しにダウンロード（wget の利用）

例：ID が 2A72 である PDB エントリーのファイルをダウンロードするには

```
wget ftp://ftp.rcsb.org/pub/pdb/data/
                              structures/all/pdb/pdb2a72.ent.Z
```
（注：紙面の都合で２行になっているが実際は１行で書くこと）

ダウンロードしたファイル pdb2a72.ent.Z は圧縮されている。解凍するには
```
gunzip pdb2a72.ent.Z
```

これで、PDB ファイル pdb2a72.ent が得られる。もしファイル名の拡張子として.pdb を使う習慣があるなら、さらに、

```
mv pdb2a72.ent.Z  2a72.pdb
```

とすれば、2a72.pdb というファイル名にすることもできる。

以下のシェルスクリプト getpdb を作って~/works/bin/の中に置いておけば、単純に getpdb 2a72 と入力するだけで、2a72.pdb を取得できるので便利であろう。

getpdb の中身：

```
1 #!/bin/csh -f
2 if ($#argv != 1) then
3     echo "Usage: $0 id"
4     exit
5 endif
6 set id = $1
7 wget ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb/pdb$id.ent.Z
8 gunzip pdb$id.ent.Z
9 mv pdb$id.ent $id.pdb
```

（ここで第２ー５行目の記述はなくてもいい。大事なのは第６ー９行目。）

上記ファイル getpdb を作成後、以下のようにしてインストールする。

```
chmod  u+x  getpdb
mv  getpdb  ~/works/bin
rehash
```

動くか試してみよう。たとえば以下のように実行し、

```
getpdb  1pga
```

以下のように rasmol で開けるか試してみよう。

```
rasmol  1pga
```

## 付録：C シェルスクリプトの基本のまとめ

（1）C シェルスクリプトの作成から実行までの流れ

①エディター（vi など）を使ってファイル（例：test.csh）を作成する。
②ファイルの実行権を有効にする（例：chmod u+x test.csh）。
③実行する（例：./test.csh）。

（2）変数

変数に文字列を代入するときの書式：　set *変数名* = *値*
例：　set file = "/data/file201.fa"
変数に数値を代入するときの書式　：　@ *変数名* = *値*
例：　@ i = 6
変数の値を参照するときの書式　　：　$*変数名*
${*変数名*}
例：　echo "$file is not found."
echo "The ${i}th sense"
特殊な参照のしかた　　例１：　echo ${file:r}
→ /data/file201
特殊な参照のしかた　　例２：　echo ${file:t}
→ file201.fa
特殊な参照のしかた　　例３：　echo ${file:t:r}
→ file201
計算結果を代入するときの書式　　：　@ *変数名* = *計算式*
例：　@ i = $i + 1
@ i ++

（3）foreach ループ

書式：　foreach *変数名* (*値１　値２　値３　…*)
*コマンド1*
・・・
end

例１：値のリストをそのまま明示して指定した例：

```
foreach file (aaa.fa abb.fa acc.fa)
   echo $file
end
```

例２：ワイルドカード（*）を使った例：

```
foreach file (*.fa)
   echo $file
end
```

例３：値のリストをファイルから読み込んだ例（（6）も参照せよ）：

```
foreach file (`cat filename.list`)
   echo $file
end
```

（4）while ループ

書式：
```
while (真偽判定式)
    コマンド1
    . . .
end
```

例：
```
@ i = 0
while ( $i < 10 )
    echo $i
    @ i++
end
```

（5）分岐（if 文）

書式：
```
if (真偽判定式) then
    コマンド1
    . . .
else
    コマンドI
    . . .
endif
```

例：
```
if ( $a == $b ) then
    echo "equal"
else
    echo "not equal"
endif
```

真偽判定式に使える比較演算子：

| 演算子 | 意味 | 例 1（文字列） | 例 2（数値） |
|---|---|---|---|
| == | 等しい | $string == "abcd" | $number == 12 |
| != | 等しくない | $string != "abcd" | $number != 12 |
| > | より大きい | (文字列には使えない) | $number > 12 |
| >= | 以上 | (文字列には使えない) | $number >= 12 |
| < | より小さい | (文字列には使えない) | $number < 12 |
| <= | 以下 | (文字列には使えない) | $number <= 12 |

真偽判定式に使えるファイル検査演算子：

| 演算子 | 意味 |
|---|---|
| -e | ファイルが存在するか |
| -z | ファイルの長さがゼロか |
| -d | ディレクトリであるか |
| -f | 普通のファイルであるか |

（前に!をつけると否定の意味になる）

-e の使用例：
```
if ( ! -e $file ) then
    echo "Hello" > $file
else
    echo "$file exists."
endif
```

（6）コマンド実行結果の取り込み

書式：　`コマンド`
（ここで使っている記号　`　はバッククォート。
引用符に使う記号　'　とは違うので注意すること）

例：　`set list = `cat filename.list``