NIBB GITC「BLAST 自由自在」
September 14-15, 2017@Okazaki

BLAST結果のテキスト処理

内山郁夫

BLAST tabular 形式 (`-outfmt 6`)

query (*S. cerevisiae*)	subject (*S. pombe*)	%identity	align-len	mismatch	gap_open	q_start	q_end	s_start	s_end	evalue	bit-score
sce:NP_001018030.1	spo:NP_596641.1	29.758	289	178	9	4	288	110	377	1.16e-16	79.0
sce:NP_001018030.1	spo:NP_587715.1	25.478	314	204	9	6	317	25	310	2.10e-11	62.0
sce:NP_001018030.1	spo:NP_595332.1	21.127	213	113	7	7	195	23	204	8.02e-04	38.5
sce:NP_001027534.1	spo:NP_594157.1	41.667	72	42	0	1	72	1	72	1.19e-20	75.1
sce:NP_001032579.1	spo:NP_595676.1	36.508	63	37	2	11	73	10	69	1.59e-05	36.6
sce:NP_001106949.1	spo:NP_001018267.3	41.463	82	46	2	1	80	1	82	3.44e-14	59.7
sce:NP_009305.1	spo:NP_039499.1	60.748	535	202	4	2	534	8	536	0.0	650
sce:NP_009306.1	spo:NP_039499.1	62.229	323	122	0	2	324	8	330	7.29e-141	418
sce:NP_009307.2	spo:NP_039499.1	58.871	248	102	0	2	249	8	255	1.29e-93	294
sce:NP_009307.2	spo:NP_039501.1	25.468	267	167	9	260	505	23	278	7.25e-11	61.2
sce:NP_009308.2	spo:NP_039499.1	58.750	80	33	0	2	81	8	87	1.32e-26	109
sce:NP_009308.2	spo:NP_039501.1	27.143	280	150	9	105	383	82	308	4.53e-17	79.3
sce:NP_009308.2	spo:XP_004001693.1	30.556	108	51	5	162	261	5	96	1.82e-04	38.1
sce:NP_009309.1	spo:NP_039503.1	36.269	579	319	19	294	848	246	798	9.46e-85	286
sce:NP_009309.1	spo:NP_039499.1	58.209	67	28	0	2	68	8	74	1.66e-19	90.5
sce:NP_009310.1	spo:NP_039503.1	34.934	687	399	20	153	827	149	799	3.02e-96	316
sce:NP_009310.1	spo:NP_039499.1	59.649	57	23	0	2	58	8	64	8.38e-16	79.0
sce:NP_009311.2	spo:NP_039501.1	28.674	279	165	9	60	319	51	314	4.37e-15	72.8
sce:NP_009311.2	spo:NP_112417.1	33.962	106	63	4	200	302	110	211	8.22e-07	48.1
sce:NP_009312.1	spo:NP_039506.1	52.083	48	23	0	1	48	1	48	6.05e-10	46.2
sce:NP_009313.1	spo:NP_039504.1	44.788	259	133	3	8	259	2	257	4.93e-67	206
sce:NP_009315.1	spo:NP_039502.1	58.886	377	155	0	1	377	1	377	2.62e-157	446
sce:NP_009316.1	spo:NP_039502.1	61.694	248	95	0	1	248	1	248	2.21e-109	332
sce:NP_009316.1	spo:NP_039501.1	25.581	215	113	10	386	571	63	259	6.09e-07	49.3
sce:NP_009316.1	spo:NP_112417.1	25.974	154	78	5	406	523	109	262	2.16e-04	41.6
sce:NP_009317.1	spo:NP_039502.1	60.920	174	68	0	1	174	1	174	2.09e-71	230
sce:NP_009317.1	spo:NP_112417.1	28.794	257	142	14	270	502	113	352	1.70e-11	63.5
sce:NP_009318.1	spo:NP_039502.1	60.140	143	57	0	1	143	1	143	3.42e-55	185
sce:NP_009319.1	spo:NP_039507.1	62.500	72	27	0	3	74	1	72	8.91e-30	98.2
sce:NP_009326.1	spo:NP_039508.1	53.527	241	110	1	13	251	7	247	1.59e-94	275

BLAST tabular 形式のカスタム出力

- 各カラムに出力するデータは、BLAST実行時の `-outfmt 6`に続けて、さらにオプションを追加することにより変更できる。

  例） tabular形式のデフォルトの出力(std)の後に、クエリの長さ(qlen)、サブジェクトの長さ(slen)、サブジェクトのタイトル(stitle)を加える。

  ```
  -outfmt "6 std qlen slen stitle"
  ```

UNIXコマンドを使ったテキスト処理

- `grep` 正規表現パターンによる文字列検索
- `sed` 文字列置換等によるファイルの変換
- `sort` ファイルのソート
- `awk` 様々なテキストファイル処理

grep: 正規表現による文字列検索

`grep 'パターン' [ファイル名 ...]`

- ファイル 中で パターン を含む行を出力する

例1） sce-spo.blastから、spo:NP_039502.1を含む行を検索

```
% grep 'spo:NP_039502.1' sce-spo.blast
```

例2）FASTA形式のファイル sce から > で始まるタイトル行を検索

```
% grep '^>' sce
```

注意）正規表現にはシェルのメタキャラクタが含まれるので、そのままコマンドラインで指定すると思わぬエラーになることが多い。そこで、パターンは '' で囲むようにする。

sed: 文字の置換など

`sed 's/置換対象パターン/置換文字列/g' [ファイル名]`

- ファイル中の、指定した正規表現パターンに合致するすべての文字列を、指定した置換文字列で置き換える。置換文字列が空の場合は文字列の削除になる。
  - 例）file中の文字列abcをすべてxyzに置き換える。
    ```
    % sed 's/abc/xyz/g' file
    ```

- 最後のgをつけない場合は、各行で最初にマッチしたパターンのみが置換される。
  - 例）各行で最初に出現した : をスペースに置き換える
    ```
    % sed 's/:/ /' file
    ```

sedコマンドにも一般にシェルのメタキャラクタが含まれうるので、パターンの指定は常に '' で囲むようにする。

FASTA形式ファイルの処理例

spo というFASTAファイルからタイトル行を抜き出し①、行頭の'>'を除去して②、最初のスペースをタブに換える③。

```
grep '^>' spo | sed 's/^>//' | sed 's/ /<tab>/' > spo.tit
```

①　②　③

(<tab> は Control-v のあと tabキーをおす)

```
spo:NP_001018179.1 hydroxymethylbilane synthase (predicted)
MPSCTSFPIGTRKSKLAVIQSEIIREELEKHYPHLEFPIISRDTIGDEILSKALFEFKRQLAKSLWTRELEALLVTNQCR
ILVHSLKDLPSEMPDGMVIACIPKRSCPLDAIVFKAGSHYKTVADLPPGSVVGTSSIRRRALLARNFPHLRFVDIRGNVG
TRLAKLDAPDSQFDCLVLAAAGLFRLGLKDRIAQMLTAPFVYYAVGQGALAVEVRADDKEMIEMLKPLQHQETLYACLAE
RALMKRLQGGCAIPIGVQTDVLAISNSSYRISLLGTVLSADGLRAAFGNAEAVVSSEEEAEELGITVALALLKNGAGPIL
EEHQRSSDSEESLKNY
>spo:NP_001018181.1 poly(A) polymerase Cid14
MGKKSVSFNRNNYKKRKNERTEPLPRRIFKNDKPSKFKSKRKEKDKNSDAYDEMLLNNNFTLLDQEEPMVEIGSKKSRND
NDSEGIRDKGGVEISNKNDPYIQFGKADPLEPLEKPDLPEEAIKRGEPTILLGIPKREGRKTNPVHDKAVENNSDFIKFD
WNSDEDEDSVSNDKSKNNESLKKSSKNEIPGFMRQRGRFFHEANEKSDSNRKRKRQAYELDSQSCPWHRQYKVEREVSRI
FHQDILHFIDYITPTPEEHAVRKTLVSRINQAVLQKWPDVSLYVFGSFETKLYLPTSDLDLVIISPEHHYRGTKKDMFVL
```

↓

```
spo:NP_001018179.1    hydroxymethylbilane synthase (predicted)
spo:NP_001018181.1    poly(A) polymerase Cid14
```

sort: 行の並べかえ

sort [-t 区切り文字]　[-k キーフィールド] [ファイル名…]

- ファイルを行単位で並べかえる。
- ソートのキーとするフィールドの指定は

  -k FLD1,FLD2[option]

  ただし、FLD1は開始フィールド、FLD2は終了フィールド。
  フィールド指定の後に以下のoption を指定可能。
  - n 数値として比較
  - g 浮動小数点文字列 (例: 1e-10)も含めて数値として比較
  - r 大きい順にソート

BLAST結果の並べかえ

sce-spo.blastをspoのエントリー名順に並べ、かつ同じエントリー名の行はE-valueの小さい順に並べ替える

```
% sort -k 2,2 -k 11,11g sce-spo.blast
```

第1フィールド　　第2フィールド

```
sce:NP_010076.1 spo:NP_001018179.1   40.373   322  181  4    5    320  6    322  5.62e-78  239
sce:NP_014100.2 spo:NP_001018181.1   34.541   414  239  9    89   476  138  545  6.47e-72  243
sce:NP_014526.1 spo:NP_001018181.1   41.590   327  185  5    139  463  202  524  7.66e-82  268
sce:NP_013466.1 spo:NP_001018187.2   26.087   138  75   4    538  654  445  576  1.01e-10  62.8
sce:NP_010407.1 spo:NP_001018187.2   22.965   344  171  13   120  395  362  679  1.05e-13  73.2
sce:NP_013943.1 spo:NP_001018187.2   28.814   177  113  4    542  711  517  687  1.05e-19  91.7
sce:NP_010795.3 spo:NP_001018187.2   25.000   344  154  12   25   291  368  684  1.06e-16  82.8
sce:NP_116620.1 spo:NP_001018187.2   23.529   170  101  6    794  938  362  527  1.06e-04  44.7
sce:NP_010267.3 spo:NP_001018187.2   35.593   59   36   2    279  337  540  596  1.07e-04  42.4
sce:NP_015436.1 spo:NP_001018187.2   25.468   267  157  9    169  426  362  595  1.10e-11  65.1
sce:NP_009833.1 spo:NP_001018187.2   32.231   121  54   5    119  220  466  577  1.16e-07  52.0
sce:NP_011704.3 spo:NP_001018187.2   21.053   266  180  10   695  947  352  600  1.20e-04  43.5
sce:NP_013822.1 spo:NP_001018187.2   25.561   223  133  7    342  538  360  575  1.23e-13  72.0
sce:NP_010778.3 spo:NP_001018187.2   20.896   402  196  15   84   391  309  682  1.24e-14  75.5
sce:NP_009907.2 spo:NP_001018187.2   23.123   333  168  11   27   287  368  684  1.26e-15  79.
```

awk: テーブル形式データの処理

- awkはテキストファイル処理に適した多機能なコマンド。基本形は

  awk　'コマンド文'　ファイル

- コマンド文の一般形式は

  パターン { アクション }

  パターンに指定した条件に合致した行について、アクションで指定した操作を行う。パターンを省略するとすべての行が対象になる。
- タブ区切りテキストなどテーブル形式のファイルでは、$1, $2, … によって各フィールド（カラム）の値を参照できる。$0は行全体、$NFは最後のカラムを表す。

awk のコマンド文の中には、一般にシェルのメタキャラクタが含まれうるので、常に ' ' で囲むようにする

awkによるBLAST tabular形式データの処理

- テーブルカラムの抽出
  - クエリ配列、サブジェクト配列、E-value（1, 2, 11カラム目）のみを出力

```
awk '{print $1,$2,$11}' sce-spo.blast
```

※パターンが指定されていないので、すべての行が出力される。

- 条件を指定したフィルタリング
  - E-value(11カラム目)が0.001以下の行を出力

```
awk '$11<=0.001{print}' sce-spo.blast
```

※出力フィールドが指定されていないので、行全体が出力される。

- その混合
  - E-valueが0.001以下の行の、クエリ配列、サブジェクト配列、E-valueを出力

```
awk '$11<=0.001{print $1,$2,$11}' sce-spo.blast
```

BLAST tabular 形式からトップヒットを抜き出す

- awkを用いて、BLAST tabular ファイルからクエリ配列ごとに、トップヒット（E-valueが最小）を出力する

方針） BLASTの出力は、クエリ配列ごとにE-valueの小さい順にソートされているので、トップヒットをとるには各クエリ配列の最初に出現する行のみを抜き出せばよい。

sce:NP_009307.2	spo:NP_039499.1	58.871	248	102	0	2	249	8	255	1.29e-93	294
sce:NP_009307.2	spo:NP_039501.1	25.468	267	167	9	260	505	23	278	7.25e-11	61.2
sce:NP_009308.2	spo:NP_039499.1	58.750	80	33	0	2	81	8	87	1.32e-26	109
sce:NP_009308.2	spo:NP_039501.1	27.143	280	150	9	105	383	82	308	4.53e-17	79.3
sce:NP_009308.2	spo:XP_004001693.1	30.556	108	51	5	162	261	5	96	1.82e-04	38.1
sce:NP_009309.1	spo:NP_039503.1	36.269	579	319	19	294	848	246	798	9.46e-85	286
sce:NP_009309.1	spo:NP_039499.1	58.209	67	28	0	2	68	8	74	1.66e-19	90.5
sce:NP_009310.1	spo:NP_039503.1	34.934	687	399	20	153	827	149	799	3.02e-96	316
sce:NP_009310.1	spo:NP_039499.1	59.649	57	23	0	2	58	8	64	8.38e-16	79.0

BLAST tabular 形式からトップヒットを抜き出す

方針） 各クエリ配列の最初に出現する行のみを取り出す

```
awk ‘prev!=$1{print; prev=$1}' sce-spo.blast
```

変数prevに一つ前の行のクエリ配列名（1カラム目）が入っている。
クエリ名が変わるごとに、その行を出力して、prevにそのクエリ名を代入する。

変数prevの値
(1つ前の行の$1)

```
(空)
sce:NP_009307.2
sce:NP_009307.2
sce:NP_009308.2
sce:NP_009308.2
sce:NP_009308.2
sce:NP_009309.1
sce:NP_009309.1
sce:NP_009310.1
```

$1 の値

```
sce:NP_009307.2   spo:NP_039499.1    58.871   ....  1.29e-93  294
sce:NP_009307.2   spo:NP_039501.1    25.468   ....  7.25e-11  61.2
sce:NP_009308.2   spo:NP_039499.1    58.750   ....  1.32e-26  109
sce:NP_009308.2   spo:NP_039501.1    27.143   ....  4.53e-17  79.3
sce:NP_009308.2   spo:XP_004001693.1 30.556   ....  1.82e-04  38.1
sce:NP_009309.1   spo:NP_039503.1    36.269   ....  9.46e-85  286
sce:NP_009309.1   spo:NP_039499.1    58.209   ....  1.66e-19  90.5
sce:NP_009310.1   spo:NP_039503.1    34.934   ....  3.02e-96  316
sce:NP_009310.1   spo:NP_039499.1    59.649   ....  8.38e-16  79.0
```

prev != $1

スクリプトによるテーブルデータの処理

Evalueが閾値以下の時、クエリ、サブジェクト、E-valueを出力するスクリプト

#! (スクリプトを実行するインタプリタのパス) （シバン：省略可）

(変数の設定など)

ファイルを開く (ファイルオブジェクト／記述子の取得)

ファイルを一行ずつ読み込み、ファイルの終端に達するまで以下を繰り返す
ここから

　行末の改行文字を取り除く
　行を空白文字(タブ)で区切って、各カラムの値を配列(リスト)に格納

　(このデータを用いて様々な処理を行う)
　Evalueが閾値以下の時、クエリ、サブジェクト、E-valueを出力

ここまで

Awk ではこの部分のみを記述

```
awk '$11 <= 0.001 {print $1, $2, $11}' sce-spo.blast
```

Perlの場合

```
#!/usr/bin/perl

$infile = $ARGV[0];      #引数からファイル名を取得
$eval_cutoff = 0.001;    #E-value閾値の設定

#ファイルを読込モードで開く。Fはファイル記述子。失敗するとエラーを表示して終了。
open(F, $infile) || die "Can't open file\n";

while ($line = <F>) {      #ファイルFから一行読み込み$lineに格納
                           #終端に達すると$lineが空となりループを抜ける
    chomp $line;           #行末の改行文字を取り除く
    @col = split(/\t/, $line);#タブ文字で分割

   #E-valueが閾値以下の時出力。配列の添字は0から始まる。
    if ($col[10] <= $eval_cutoff) {
        print join("\t", $col[0], $col[1], $col[10]),"\n";
    }
}
```

Rubyの場合

```
#!/usr/bin/ruby

infile = ARGV[0]
eval_cutoff = 0.001

#ファイルを読込モードで開く。fはファイルオブジェクト。失敗すると例外が発生。
f = File.open(infile)

while line = f.gets      #ファイルfから一行読み込みlineに格納
                         #終端に達するとlineが空となりループを抜ける
    line.chomp!          #行末の改行文字を取り除く
    col = line.split("\t")     #タブ文字で分割

   #E-valueが閾値以下の時出力。配列の添字は0から始まる。
    if col[10].to_f() <= eval_cutoff
        puts [col[0], col[1], col[10]].join("\t")
    end
end
```

Pythonの場合

```
#!/usr/bin/python

import sys                      #sysモジュールを読み込む

infile = sys.argv[1]            #引数からファイル名を取得
eval_cutoff = 0.001                 #E-value閾値の設定

#ファイルを読込モードで開く。fはファイルオブジェクト。失敗すると例外が発生。
f = open(infile)

for line in f:    #ループにおいて、fはファイルを一行ずつ返すイテレータとしてはたらく
    line = line.rstrip()        #行末の改行文字を取り除く
    col = line.split('\t')      #タブ文字で分割

   #E-valueが閾値以下の時出力。配列の添字は0から始まる。
    if float(col[10]) <= eval_cutoff:
        print '\t'.join( [col[0], col[1], col[10]] )

#pythonでは「字下げ」でブロックを認識するので、ブロックの終端を示すマークはない
```

スクリプトの実行

- インタプリタから実行 (シバンは不要)

```
% perl filter_blast.pl sce-spo.blast
```

- 直接コマンドとして実行(先頭行にシバンが必要)

あらかじめスクリプトファイルに実行権を与える必要がある

```
% chmod +x filter_blast.pl
% ./filter_blast.pl sce-spo.blast
```

より複雑なデータ処理

- 以下の処理を行うスクリプトを作成しよう。ただしファイル名は引数から取得するようにすること。

> sce-spo.blast の結果から、出芽酵母(sce)の各クエリに対するトップヒットで、かつE-value が 10^{-5}以下の場合のみを抽出せよ。

```
sce:NP_009307.2   spo:NP_039499.1    58.871   248   102   0   2    249   8    255   1.29e-93   294
sce:NP_009307.2   spo:NP_039501.1    25.468   267   167   9   260  505   23   278   7.25e-11   61.2
sce:NP_009308.2   spo:NP_039499.1    58.750   80    33    0   2    81    8    87    1.32e-26   109
sce:NP_009308.2   spo:NP_039501.1    27.143   280   150   9   105  383   82   308   4.53e-17   79.3
sce:NP_009308.2   spo:XP_004001693.1 30.556   108   51    5   162  261   5    96    1.82e-04   38.1
sce:NP_009309.1   spo:NP_039503.1    36.269   579   319   19  294  848   246  798   9.46e-85   286
sce:NP_009309.1   spo:NP_039499.1    58.209   67    28    0   2    68    8    74    1.66e-19   90.5
sce:NP_009310.1   spo:NP_039503.1    34.934   687   399   20  153  827   149  799   3.02e-96   316
sce:NP_009310.1   spo:NP_039499.1    59.649   57    23    0   2    58    8    64    8.38e-16   79.0
```

Perlによるスクリプト例

各クエリに対するトップヒットで、かつE-value が 10^{-5}以下の場合のみを抽出する。

(get_tophit.pl)

```
#!/usr/bin/perl
$eval_cutoff = 1e-5;          # BLAST E-valueの閾値
$file = $ARGV[0];

open(F, $file) || die "Can't open file\n"; # 引数で指定したファイルを開く
# ファイルFから1行ずつ読み込んで処理する
while ($line = <F>) {
     chomp $line;                        # 行末の改行を取り除く
     @col = split(/\t/, $line);          # タブで分割して、各カラムを配列 @col に格納

     # 配列を分かりやすい名前の変数に再格納する
     $query = $col[0];         # クエリ配列は第1カラム（配列の添字は0から始まる）
     $evalue = $col[10];       # E-valueは第11カラム

     # クエリ配列が直前のクエリ配列から変わっていればトップヒット
     if ($prev_query ne $query) {
          # E-value が閾値以下であれば、行全体を出力する。
          if ($evalue <= $eval_cutoff) {
               print $line,"\n";
          }
     }
     $prev_query = $query;     # 次行の処理のため、直前のクエリを記憶しておく
}
```

```
% ./get_tophit.pl sce-spo.blast > sce-spo.blasttop
```

2つのテーブルの結合（join）

- 2つのテーブルにおいて、それぞれに指定したカラムに同じ値を持つ行を結合して、一つのテーブルにまとめる操作。

例題）分裂酵母の各配列のタイトル行を集めた spo.tit を用いて、 sce-spo.blasttop の各行に、ヒットした分裂酵母の配列(2カラム目）のタイトル行を付加せよ。

sce-spo.blasttop

sce:NP_009305.1	spo:NP_039499.1	60.748
sce:NP_009306.1	spo:NP_039499.1	62.229
sce:NP_009307.2	spo:NP_039499.1	58.871
sce:NP_009308.2	spo:NP_039499.1	58.750
sce:NP_009309.1	spo:NP_039503.1	36.269

spo.tit

spo:NP_039499.1	cytochrome c oxidase 1 (mitoch
spo:NP_039501.1	hypothetical protein ScpofMp03
spo:NP_039502.1	cytochrome b (mitochondrion)
spo:NP_039503.1	hypothetical protein ScpofMp05
spo:NP_039504.1	ATPase6 (mitochondrion)

sce:NP_009305.1	spo:NP_039499.1	60.748	cytochrome c oxidase 1 (mitochondrion)
sce:NP_009306.1	spo:NP_039499.1	62.229	cytochrome c oxidase 1 (mitochondrion)
sce:NP_009307.2	spo:NP_039499.1	58.871	cytochrome c oxidase 1 (mitochondrion)
sce:NP_009308.2	spo:NP_039499.1	58.750	cytochrome c oxidase 1 (mitochondrion)
sce:NP_009309.1	spo:NP_039503.1	36.269	hypothetical protein ScpofMp05 (mitoch

2つのテーブルの結合（join）

- (方針） ハッシュを使って2つのテーブルを結合する

ハッシュ（連想配列、辞書）
文字列（キー）を使って各要素（値）を取り出すことができる配列

Perl での記法: $hash{'key'} = $value
Ruby, Pythonでの記法: hash['key'] = value

spo.tit から名前とタイトルとを対応付けるハッシュを作成

キー	値
spo:NP_039499.1	cytochrome c oxidase 1 (mitoch..
spo:NP_039501.1	hypothetical protein ScpofMp03
spo:NP_039502.1	cytochrome b (mitochondrion)
spo:NP_039503.1	hypothetical protein ScpofMp05

ハッシュを使ってタイトルを取得し付加

sce-spo.blasttop

sce:NP_009305.1	spo:NP_039499.1	60.748
sce:NP_009306.1	spo:NP_039499.1	62.229
sce:NP_009307.2	spo:NP_039499.1	58.871
sce:NP_009308.2	spo:NP_039499.1	58.750
sce:NP_009309.1	spo:NP_039503.1	36.269

cytochrome c oxidase 1 (mitoch..

Perlスクリプトによるjoin

BLAST結果にアノテーションを付加する。BLAST結果の2カラム目とタイトルを集めたファイルの1カラム目を使ってjoinする。その際、配列名とタイトルとを対応づけたハッシュ表を使う。

(add_title.pl)

```
#!/usr/bin/perl

$blast_file = $ARGV[0];                  # BLAST結果ファイル
$tit_file = $ARGV[1];                    # タイトル行を集めたファイル
$coln = ($ARGV[2] ? $ARGV[2] : 1);       # BLAST結果でjoinの対象とするカラム(1から数える)。デフォルトは1。
# まず、$tit_file を使ってハッシュ表を作成する
open(F1, $tit_file);           # $tit_file を読み込み用に開く
while($l = <F1>) {
     chomp $l;                                  # 最後の改行を取り除く
     ($name, $title) = split(/\t/,$l);          # タブで区切って、各カラムを$name, $titleという変数に格納
     $Title{$name} = $title;                    # 名前からタイトルを引くためのハッシュ%Titleを作成
}
close(F1);                #複数のファイルを扱うときは、一つのファイル処理が終わるごとに明示的にクローズした方がよい

# ハッシュを使って、$blast_file にタイトルを付け加える
open(F2, $blast_file);   # $blast_file をファイルを読み込み用に開く
while($l = <F2>) {
     chomp $l;                  # 最後の改行を取り除く
     @col = split(/\t/,$l);     # タブで区切って、各カラムを配列 @col に格納
     $name = $col[$coln-1];         # $coln番目のカラムを $name として取り出す
     $title = $Title{$name};        # 先に作ったハッシュを使って、タイトルを取り出す
     print join("\t", @col, $title),"\n";    # 各カラムの値(@col)の後ろに$titleを加えて、タブ区切りで出力
}
```

```
% ./add_title.pl sce-spo.blasttop spo.tit 2 >sce-spo.blasttop.addtit
```

モジュールの利用

- モジュール：特定の処理を行うためのプログラムのまとまりで、スクリプト言語の機能を拡張するもの
- バイオインフォマティクス関連の様々なデータを扱うモジュールとして、**BioPerl**, **BioRuby**, **BioPython**などが公開されている。
- 多くのモジュールは「オブジェクト指向」によって設計されており、効果的に利用できる

Perlにおいて、オブジェクト指向のモジュールを利用する際の基本形

```
$オブジェクト変数 = クラス名->new(引数,...);      # オブジェクトの作成
$オブジェクト変数->メソッド(引数,...);                # メソッドの利用
```

例）FASTQ形式の核酸配列を読み込んで、タンパク質に翻訳してFASTA形式で表示する (BioPerl のBio::SeqIOモジュールを利用)

(translate.pl)

```
#!/usr/bin/perl
use Bio::SeqIO;     # Bio::SeqIO モジュールの利用

$infile = $ARGV[0];       # 入力配列ファイル(FASTA形式)は引数で指定
$outfile = ">&STDOT"      # 出力もFASTA形式で標準出力(&STDOUT)へ

# FASTA形式の配列ファイル($infile)を読み込み用に開く(入力ファイルオブジェクト$seqinの作成)
$seqin = Bio::SeqIO->new(-file=>$infile, -format=>'fasta');
# FASTA形式の書き込み用の配列ファイル($outfile)を開く(出力ファイルオブジェクト$seqoutの作成)
$seqout = Bio::SeqIO->new(-file=>">$outfile", -format=>'fasta');

# 入力ファイルから配列を一つずつとってくる (メソッドnext_seq; 配列オブジェクト$seq_objが返る)
while ($seq_obj = $seqin->next_seq) {
     $protseq = $seq_obj->translate;    # 配列を翻訳する(メソッドtranslate)
     $seqout->write_seq($protseq);      # 出力ファイル(FASTA形式)に書き込む(メソッドwrite_seq)
}
```