

2016 GITC 準備編 UNIX/R の基礎 演習問題

復習問題1 UNIX 基本コマンド

1. ターミナルを起動して、~/data/1_unix ディレクトリに移動しカレントディレクトリを確認します。加えてカレントディレクトリ(~/data/1_unix)以下にある全てのファイルを確認しましょう。オプション"-R"を使うとディレクトリを辿りながら全てのファイルを表示します。
2. ~/data/1_unix/sprot ディレクトリ内の FASTA ファイル全てを~/unixtest/FASTA-EX ディレクトリにコピーしましょう。ディレクトリがない場合は新規に作成します。上記の操作を行った後に、~/unixtest/FASTA-EX ディレクトリ内のファイルリストを確認しましょう。
3. ~/unixtest/FASTA-EX ディレクトリに移動して、コピーした FASTA ファイル全てのタイトル行のみを抜き出して `less` コマンドで確認しましょう。ただし、ここではすべてのファイルに配列は1つずつ入っています。確認方法は、`grep` コマンドを使用します。オプションを使用しないと結果にファイル名が付加されてしまうので、ファイル名を表示しないオプションを `man` コマンドで調べましょう。
4. 同じように、FASTA ファイル全ての配列行のみを `less` コマンドで確認しましょう。こちらは、`tail` コマンドのみを使用します。
5. ~/unixtest/FASTA-EX にコピーした FASTA ファイルから HUMAN, MOUSE, DROME に関する(ファイル名に含まれる) Multi-FASTA 形式のファイルをそれぞれ `human.fasta`, `mouse.fasta`, `drome.fasta` として~/unixtest/の下に `cat` コマンドを使用して作成しましょう。
6. FASTA-EX ディレクトリを削除しますが、削除する前にこのディレクトリ全体の圧縮アーカイブを~/unixtest/FASTA-EX.tar.gz として `tar` コマンドで作成しましょう。作成できたら、圧縮アーカイブの中身を確認してから FASTA-EX ディレクトリ全体を削除します。
7. 圧縮アーカイブ FASTA-EX.tar.gz は自分自身のみ読み書きできるよう `chmod` コマンドを使用して(グループとその他は読み書きできないよう)アクセス権を変更します。
- *8. UNIX にはファイルの検索に用いる `find` という非常に便利なコマンドがある。`man` コマンドを使用して `find` コマンドの使用方法を確認し、この `find` コマンドを使用してホームディレクトリ配下のどこかにある "`SYMM_DROME.sprot`"を探しましょう。
- *9. 同じく、`find` コマンドでは探しだしたファイル(複数も可能)を対象にして(引数にして)コマンドを実行することができます(`-exec`を使用)。この機能を使って、ホームディレクトリ配下にある `Swiss-Prot` ファイル(拡張子 `.sprot`)を全て探し出して ~/SP/ ディレクトリに一度にコピーしましょう。

復習問題 2 R 入門

`cancer` データを使って、以下の問題を考えよう。

1.(a) 男性で喫煙歴がある人のデータを抜き出し、結果を `cancer.subset` という変数に代入せよ。
何人いるか。

(b) それらの人の `gene1` の発現データを取り出し、その平均値を計算せよ。

(c) 抽出した結果をタブ区切りテキストとして `cancer.subset.txt` というファイルに保存せよ。

2. (a) `gene1` と `gene2` の散布図を作成し、`gender` によって点を色分けせよ。

(b) 散布図に回帰直線を引け。この回帰直線へのあてはめは、有意水準を 0.01 として有意であると言えるか。

*3. テキストでは、`gene1` の発現量(`gene1`)が性別(`gender`)によって違いがあるという結論が t 検定で得られ、また癌のステージ(`stage`)によっても違いがあるという結論が分散分析から得られた。ただし、癌のステージの中で明らかな違いがあるのは `stage III` のみであった（これは `boxplot(gene1 ~ stage, cancer)` で確認できる）。

そこで、`gender` の効果を考慮しつつ `stage` の比較を行うため、Trellis Plot の技法を使ったプロットを作成してみよう。これを行う `lattice` package は R に標準で含まれているが、使う際にはライブラリのロードが必要である。

```
> library(lattice)
```

```
> bwplot(gene1 ~ stage | gender, cancer)
```

`bwplot` は `boxplot` と同様に箱ひげ図を作成するが、特定の因子によって条件付けしたプロットを作成できる。ここでは、`gender` によってまず被験者を `female` と `male` に分けて、そのそれぞれで箱ひげ図を作成している。この結果から `stage` の `gene1` の発現量への効果について、どのような結論が得られるか考察せよ。

*4.(a) `cancer` データから各患者の `gene1` から `gene6` の発現量を抜き出した部分データフレームを作成し、変数 `expr` に代入せよ。

(b) 各遺伝子間の発現量の相関（散布図を描いたときに傾きを持つ直線上に分布する傾向）の強さは相関係数によって表される。相関係数は -1 から 1 までの値を取り、0 が無相関を表す。相関係数が負の値のときは、傾きが負、すなわち一方が大きくなれば他方が小さくなる関係を表す。R では、行列の各カラム間の相関係数は、`cor` 関数によって一度に計算できる。これを用いて `expr` の各カラム間の相関係数を計算せよ。

(c) 相関係数の絶対値が 0.5 以上のときに強い相関があるとして、`gene1~gene6` を、発現の

相関の強さによっていくつかのグループに分けることができるかを検討せよ。ただし、絶対値をとるのは `abs` 関数で行える。

復習問題 3 UNIX によるテキストファイル処理

この演習では、`ecoli.htseq` ファイルを使用する。ファイルは下記のパスにある。

```
~/data/7_ex/ex2/
```

`ecoli.htseq` には、アノテーションテーブルを使って、各遺伝子にマッピングされたリード数をカウントしたものが書かれている。

1. `ecoli.htseq` に記載されていて、カウントされた値が 100 回以上である行を標準出力に表示せよ。
2. そのうち、必要なのは `b****` という文字列で始まる行である。それらだけを取り出して、`ecoli.temp` というファイルに保存せよ。
3. `ecoli.temp` を、カウントされた回数の多い順にソートし、`ecoli.htseq.sorted` というファイルに保存せよ。
4. 1.~3.の操作を一つのコマンドで出来るように、`htsort.sh` というシェルスクリプトを作成せよ。シェルスクリプトの実行後に中間ファイル(`ecoli.temp`)を消えることができていればなお良い。
- *5. 4.で作成したスクリプトは、`ecoli.htseq` という特定のファイルにのみ使用できる。上記のよう処理を他のファイルに対しても行えるように、引数を 1 つ取るように書き換えよ。例えば、`ecoli_other.htseq` というファイルに対し、

```
$ ./htsort.sh ecoli_other.htseq
```

というコマンド書式で同じことが行えるようにせよ。

復習問題 4 シェルスクリプト

~/data/5_script/more ディレクトリの中には、ecoli.4.fastq ～ ecoli.6.fastq の 3 つの fastq ファイルがあります。

1. これらを bowtie2 で ~/data/5_script/ecoli_genome.fa にマッピングし、~/data/5_script/more 内に sam ファイルとして保存するスクリプトを作成しましょう。講義内で作成した ex5.sh を改造しても良いです。fastq ファイル名の「ecoli」部分は引数から得ても、スクリプト内に直書きしてもどちらでもかまいません。ecoli_genome.fa (とそのインデックス) は、~/data/5_script/more 内ではなく、~/data/5_script 内にあることに注意してください。

ex5.shはデータセット内 data/5_script/cook にもあります

- *2. 1.で作成したスクリプトを以下のように書き換えました。1 のスクリプトを実行後、下記のスクリプトを~/data/5_script/more 内で実行すると標準出力には何かが出力されるか答えなさい。また、この改良は何を目的としたものかを 2 つ答えなさい。

```
#!/bin/sh
org=${1}
for i in 1 2 3 4 5 6
do
  if [ -e "${org}.${i}.sam" ]
  then
    echo "${org}.${i}.fastq has already been mapped"
  elif [ -e "${org}.${i}.fastq" ]
  then
    bowtie2 -x ../${org}_genome -U ${org}.${i}.fastq -S ${org}.${i}.sam
  else
    echo "${org}.${i}.fastq was not found"
  fi
done
```

*実践演習 1 RNA-Seq 解析結果の集計

大腸菌の RNA-Seq の例題をもう一度考える。~/data/7_ex/rnaseq に移動しよう。この下の **results** ディレクトリには、大腸菌ゲノムにマッピングした結果から **htseq-count** を使って各遺伝子領域に重なるリード数を集計した結果 (*.htseq) が格納されている。この結果を基に、UNIX コマンドを使って、簡単な解析結果の集計を行ってみよう。

なお、ここで使ったサンプルデータ (NCBI GEO データベースの GSE59468) には、大腸菌の 2 つの系統を、それぞれ 2 つの異なる条件で培養して、それぞれについて 3 つの反復をとった、計 12 個のサンプルのデータが含まれており、ファイルについての番号は、それぞれ以下の系統・条件の組合せに対応している (ただし、サイズを縮小するためにデータを大幅に間引いている)。

	strain	condition	replicate	SRA accession
1	MG1655	MPOS	1	SRR1515282
2	MG1655	MPOS	2	SRR1515283
3	MG1655	MPOS	3	SRR1515284
4	MG1655	Urine	1	SRR1515285
5	MG1655	Urine	2	SRR1515286
6	MG1655	Urine	3	SRR1515287
7	CFT_073	MPOS	1	SRR1515276
8	CFT_073	MPOS	2	SRR1515277
9	CFT_073	MPOS	3	SRR1515278
10	CFT_073	Urine	1	SRR1515279
11	CFT_073	Urine	2	SRR1515280
12	CFT_073	Urine	3	SRR1515281

以下で、コマンド先頭の **\$** は Unix のプロンプトを表すので、それ以降を改行なしで入力すること。

1) 各 *.htseq は遺伝子ごとのリード数が記載されている。この 12 個のファイルを、以下のよう にすべて横に並べて 1 つのファイルを作りたい。

```
(ecoli.1.htseq)  (ecoli.2.htseq)  (ecoli.3.htseq)  ..... (ecoli.12.htseq)
b0001  11      b0001  0      b0001  7
b0002  117     b0002  9      b0002  131
b0003  33      b0003  1      b0003  31
b0004  44      b0004  4      b0004  58
b0005  3       b0005  0      b0005  2
```

ファイルを行単位で結合する UNIX コマンドとして、**paste** がある。以下を実行してみよう。

```
$ paste results/ecoli.[1-3].htseq |less
```

`paste` は引数の順にファイルを結合するので、引数の順番に注意する必要がある。以下のワイルドカードを使った2つのコマンドの出力を比べてみよう。ただし、`echo` は受け取った引数をそのままの順で表示するコマンドである。

A) `echo results/ecoli.*.htseq`

B) `echo results/ecoli.?.htseq results/ecoli.1?.htseq`

ファイルが 1, 2, 3, ... の順に並ぶのはどちらか。なお、コマンド B) は、以下のようにより簡潔に書くこともできる（試してみよ）：`echo results/ecoli.{?,1?}.htseq`

この結果を用いて、`paste` コマンドによって `htseq` の出力ファイルを 1, 2, 3, ... の順に結合したファイルを作成し、結果をファイル `ecoli.paste_all` に格納せよ。

2) `ecoli.paste_all` は、b0001 などの遺伝子名が奇数列に繰り返し出現するため、冗長である。そこで最初の列だけ残して、あとの遺伝子名の列は除きたい。すなわち、1,2,4,6,...,22,24 列目のみを残したい。これは `awk` を使っても行えるが、`cut` の方がより簡潔に書ける。`cut` は `-f` オプションによって指定された列のみを出力する。たとえば、`cut -f 1,3,4 file` は、`file` からタブ区切りで 1,3,4 番目の列を抜き出して表示する。

また、`ecoli.paste_all` の最後の5行は遺伝子領域にマッチしなかったリード数が記載されているが、これらの行を除きたい。`grep` コマンドを使って除くことを考えよ。これらの処理を行った結果を、`ecoli.count_all` に格納せよ。

3) 発現解析でよく行われる標準化の方法に RPKM (Read Per Kilobase per Million mapped reads) がある。これを計算するには各遺伝子の長さの情報が必要である。遺伝子アノテーションファイル `ecoli.gtf` には各遺伝子 (exon) の開始位置と終了位置の情報が含まれている。`awk` を使ってこれから遺伝子の長さを計算しよう。3 列目が exon である行について、開始位置 (4 列目) と終了位置 (5 列目) を使って長さを計算し、10 列目にある遺伝子名、長さの順に出力する。

正しく実行すると、以下のような出力が得られるはずである。

```
"b0001"; 66
"b0002"; 2463
"b0003"; 933
"b0004"; 1287
"b0005"; 297
.....
```

ダブルクォート (") とセミコロン (;) が余分なので、取り除こう。これには `sed` コマンドが使える。`sed` はファイルの簡単な編集を行うプログラムで、`'s/置換する文字列/置換後の文字列/g'` によって、ファイル中の指定した文字列を一斉に置換することができる。置換後の文字列を空にすることで、特定の文字列を除去することもできる。また、置換する文字列には正規表現が使えるので、〔ダブルクォートまたはセミコロン〕という指定は正規表現を使って書ける。

そこで、以下のコマンドによって、`file` 中のすべてのダブルクォートとセミコロンを除くことができる。

```
$ sed 's/[";]//g' file
```

これによって、以下のような出力が得られるはずである。

```
b0001 66
b0002 2463
b0003 933
b0004 1287
b0005 297
....
```

これをファイル `ecoli.gene_length` に保存せよ。

4) `ecoli.count_all` と `ecoli.gene_length` を結合して一つのファイルにしよう。実は、`ecoli.count_all` は遺伝子名でソートされているが、`ecoli.gene_length` は遺伝子名でソートされていない (`ecoli.gtf` が位置によってソートされているため)。従って、そのまま `paste` すると正しく結合されない (確認せよ)。そこで、まず `ecoli.gene_length` を遺伝子名によってソートし、`ecoli.gene_length_sorted` というファイルに保存しよう。

`ecoli.count_all` と `ecoli.gene_length_sorted` を `paste` コマンドで結合することもできるが、また遺伝子名の行が余分にできてしまう。ここでは別のコマンド `join` を使った結合を試みよう。`join` は2つのファイルに共通の列 (ここでは1列目の遺伝子名) があり、かつファイルがいずれもそれらの列に関してソートされているとき、それらの列をキーとして2つのファイルを対応づけて結合するコマンドである。

```
$ join -1 1 -2 1 ecoli.gene_length_sorted ecoli.count_all
```

引数の `-1 1 -2 1` は1番目のファイルの1列目と2番目のファイルの1列目をキーとして対応づけることを意味しているが、これはデフォルトの動作なので、省略しても同じ結果になる。

`paste` はキー (ここでは遺伝子) の並びが2つのファイルで完全に一致していないと、行が合わなくなってしまうが、`join` はキーに重複や抜けがあっても2つのファイルの間でずれていても、同じキーを持つ行を正しく対応づけてくれる。非常に強力なコマンドである。

なお、`join` はタブ区切りのファイルをスペース区切りに変更してしまう。見やすさを維持するためにタブ区切りに直したい場合は、`sed 's/ / /g' file` によって、タブ区切りに変換できる。ただし、`sed` コマンドのスラッシュ (/) で区切られた文字列は、最初がスペースで2番目はタブである。コマンドラインでタブを入力するには `Control-v` を押してから `Tab` キーを押す。これによって以下のようなファイルができるはずである。

```
b0001    66    11    0    7    4    7   17    9    0    0    3    7    1
b0002   2463   117    9   131   51   20   161   20    1    0   32   34   2
b0003    933    33    1   31   10    4   30    9    1    2    7    9    4
.....
```

この結果を `ecoli.result_all` に保存しよう。

`ecoli.result_all` のようなタブ区切りテキストは、Excel などの表計算ソフトや R などの統計解析ソフトに読み込んで処理できる。実践演習 2 では、このデータから R を使って、**RNA-seq** データの標準化の方法としてよく用いられている **RPKM** 値を計算してみよう。

*実践演習 2 R を用いた RPKM 値の計算

ここでは R を使って、RNA-seq データの標準化の方法としてよく用いられている RPKM 値を計算してみよう。RPKM 値は、もとのリード数を、遺伝子の長さ 1000 bp あたり、各サンプルのリード数の和 1,000,000 read あたりになるように標準化する。

本実習では、実践演習 1 でディレクトリ `~/data/7_ex/rnaseq` の下に `ecoli.result_all` というファイルが作成されていることを前提としている。これを確認せよ。実践演習 1 をスキップする場合は、同じディレクトリの下に `answer` というディレクトリにある `eco.result_all` をコピーして使うこと。

以下でコマンド先頭の `>` は R のプロンプトを表すので、それ以降を改行なしで入力すること。

1) まず R の作業ディレクトリを `~/data/7_ex/rnaseq` に移動しよう。メニュー（その他→作業ディレクトリの変更）から移動しても、コンソールから `setwd("ディレクトリ名")` を打ち込んでよい。移動したら、`getwd()` で正しく移動できていることを確認しよう。

次に、`ecoli.result_all` からデータを読み込む。`read.table` 関数を使ってデータを読み込み、変数 `eco_rna` に代入しよう。このファイルは、セパレータはタブで、ヘッダはなしである。

1 列目に遺伝子名が入っているので、これを各行の「名前」として指定しよう。これには、`read.table` のオプションとして `row.names=1` を指定する。このとき、`row.names` で指定した列（1 列目）が行の名前として読み込まれる。

読み込んだら、`head` 関数で先頭数行を表示して、正しく読めているかを確認しよう。

```
      V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14
b0001  66 11 0  7 4  7 17 9  0  0  3  7  1
b0002 2463 117 9 131 51 20 161 20 1 0 32 34 2
b0003 933 33 1 31 10 4 30 9 1 2 7 9 4
....
```

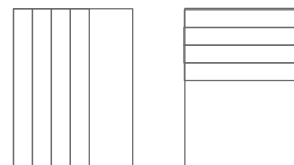
最初の行で `V1` がいないことに注意。1 列目はデータではなく、行の名前として読み込まれている。行に名前をつけると、`eco_rna["b0002",]` のように名前で行を抽出することが可能になる。

なお、1 行目はヘッダを表しており、最初のデータである `b0001` は 2 行目から始まる。「入力ファイルにヘッダあり」として読み込んでしまうと行がずれてしまうので注意しよう。また、今回はファイルにヘッダ行が含まれないので、各列にはデフォルトで R がつけた名前（`V+元`のファイルの列番号）がついている。列名を変更したい場合は `colnames` という関数で行えるが、ここでは省略する。

2) `eco_rna` は 1 列目に遺伝子の長さが入っており、2,3,4 列目、5,6,7 列目、8,9,10 列目、11,12,13

列目が、それぞれ4つの条件の組合せについての3回ずつの反復実験の結果に対応している。まず長さを標準化するため、2-13列のデータを取り出し、それらを1列目の遺伝子の長さで割った上で1000 bpあたりの値に変換しよう。Rでは、このように行列を列（縦）方向のベクトルの並びとみて（図1A）、それらと同じ長さのベクトルとの間で行うベクトル演算は、行列とベクトルの演算として簡潔に書ける。これを確認するため、以下を実行してみよう。

図1 行列のベクトルへの分割
A) 列方向に分割 B) 行方向に分割



```
> head(eco_rna / eco_rna[,1])
```

```
      V2      V3      V4      V5      V6      V7      V8
b0001  1 0.16666667 0.000000000 0.106060606 0.060606061 0.106060606 0.257575758
b0002  1 0.04750305 0.003654080 0.053187170 0.020706456 0.008120179 0.065367438
b0003  1 0.03536977 0.001071811 0.033226152 0.010718114 0.004287245 0.032154341
...
```

これが `eco_rna` の各値を同じ行の1列目の値（＝長さ）で割った値であることを確認しよう。1列目は自分自身で割ることになるのですべて値が1になっている。また、1行2列目は $11/66$ 、2行2列目は $117/2463$ の値などとなっている（確認せよ）。

これを用いて、`eco_rna` から2-13列のデータを取り出し、それらを1列目の遺伝子の長さで割った上で1000 bpあたりの値に標準化しよう。この結果を `eco_rna_tmp` という変数に代入しよう。結果を `head` で確認すると、以下の様になるはずである。

```
      V3      V4      V5      V6      V7      V8      V9
b0001 166.66667 0.000000 106.060606 60.606061 106.060606 257.575758 136.363636
b0002  47.50305  3.654080  53.187170  20.706456   8.120179  65.367438   8.120179
b0003  35.36977  1.071811  33.226152  10.718114   4.287245  32.154341   9.646302
```

3) RPKMを計算するには、`eco_rna_tmp` の値をさらにそれぞれのサンプルのリード数の和で割って標準化する必要がある。リード数の和を計算するため、`eco_rna` のデータに戻る。このデータの2-13列目に各サンプルのリード数が入っているので、それぞれの列を抽出したベクトルについて和をとるとよい。ベクトルの和をとるには `sum` 関数で行えるので、たとえば `eco_rna` の2列目の和は `sum(eco_rna[,2])` で計算することができる（試して見よ）。この計算を各列について繰り返し行えばよい。

このように、行列やデータフレームから各行または各列をベクトルとして抽出して、特定の関数を使った計算を繰り返すような処理を行う際は、`apply` 関数を使うことができる。`apply` は、`apply(x, MARGIN, FUN)` のように3つの引数をとる。`x` は入力データとなる行列（またはデータフレーム）、`MARGIN` は1または2をとり、1は行（横）方向（図1B）、2は列（縦）方向（図1A）のベクトルに分割した上で、行列全体にわたって処理を繰り返すことを指定する。`FUN` は適用する関数名である（`help(apply)` で確認せよ）。これを使って、`eco_rna` の2-13列目について、列方向にベクトルを抽出し、`sum` 関数を適用せよ。結果を `eco_rna_readsum` という変数に格納せよ。結果は要素数12のベクトルになるはずである。その最初の要素が `sum(eco_rna[,2])` と一致することを確認せよ。

4) これで RPKM を計算する準備が整った。あとは `eco_rna_tmp` の各行について、値を `eco_rna_readsum` の同じ位置の値で割って 1,000,000 倍すれば RPKM 値が計算できる。ただし、今回は列方向ではなく、行方向にベクトルを抽出して計算することになるので、これを行列とベクトルの割り算として `eco_rna_tmp / eco_rna_readsum` で計算することはできない。そこで再び `apply` 関数を使おう。

まずここで行う計算を確認しよう。たとえば 1 行目については以下のベクトルの割り算を行う。

```
> eco_rna_tmp[1,] / eco_rna_readsum
```

`apply` を使うには関数の形にする必要がある。そこで、`function` 関数を使って、2 つの引数間の割り算を行う簡単な関数 `div` を作ろう。

```
> div <- function(x,y){x/y}
```

うまく動作するかどうか確認しよう。

```
> div(eco_rna_tmp[1,], eco_rna_readsum)
```

これで `apply` には関数名として `div` を与えればよい。ただし、`sum` と違って `div` は引数を 2 つとる。1 つめの引数は `apply` 内部で与えられるが、2 番目の引数は外から与えなければならない。`apply` 関数では 4 番目以降の引数に、適用する関数の 2 番目以降の引数を与えることができるので、`eco_rna_readsum` を 4 番目の引数として指定する。これで行方向に関数を適用するように `apply` 関数を実行すればよい。`apply` を実行した後で全体を 1000000 倍する。この結果を `eco_rna_rpk0` という変数に代入せよ。

これを `head` で確認すると、結果が大量に表示されて流れてしまうはずである。実は結果は行と列が入れ替わってしまっており、1 行の長さが非常に長くなってしまっている。これは `dim` 関数で確認できる。`dim` 関数は、指定した行列やデータフレームの行数と列数を表示する。`dim(eco_rna_tmp)` と `dim(eco_rna_rpk0)` を比較してみよう。

そこで、`eco_rna_rpk0` の行と列を入れ替えよう。これは転置行列をとる (`transpose`) という操作であり、R では `t` 関数で行う (`help(t)` で確認せよ)。この結果を `eco_rna_rpkm` に代入しよう。`head` で結果を確認すると、以下の様になるはずである。

	V3	V4	V5	V6	V7	V8	V9
b0001	1622.0127	0.0000	1028.43656	1599.14670	3950.70424	1809.39031	1105.77962
b0002	462.3032	315.3604	515.73937	546.35889	302.47257	459.18611	65.84694
b0003	344.2213	92.5012	322.18362	282.80729	159.69774	225.87433	78.22235