2. フィルタによるテキスト処理

2 回生前期の実習でプログラミング言語 Perl を学んだ[1]。しかし、簡単なテキスト処理は長いプログラムを作成するまでもなく、UNIX のコマンドを複数組み合わせるだけで実現できることが多い。

入力されたテキストを加工して出力するコマンドをフィルタと呼ぶことがある。UNIX にはフィルタが多数用意されている。今日の実習では、フィルタの便利な使いかたと、複数のフィルタの連携技を練習する。

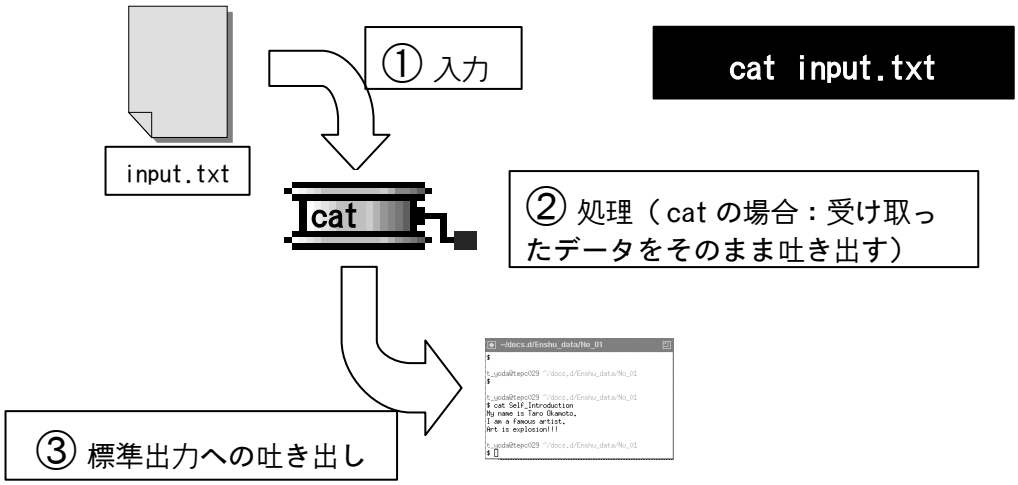
2.a. pdb から座標データをダウンロード

テキスト処理の練習台として、PDB から 2 つの立体構造ファイルをダウンロードしよう。

1. 端末ウィンドウを開き、前回の実習で作成した、ディレクトリ works/db/pdb へ移動する。
2. FireFox を起動し、PDB へ行き、1CLL をダウンロード。ダウンロードしたファイルを上記のディレクトリへ移動させる。
3. 次に、PDB から 1UAO をダウンロード。同様に移動させる。

2.b. リダイレクションとパイプの復習

リダイレクションは、UNIX の重要かつ便利な機能である。2 回生前期の実習でも習ったが、復習しておく。



1 Perl の特徴はテキスト処理が得意なことであり、それゆえ、バイオインフォマティクスに適している。

多くのコマンドやソフトウェアは、①なんらかの情報を受け取り、②処理し、③結果を出力する（前ページの図）。

cat の処理結果は端末ウィンドウに表示されるが、これは cat が結果を標準出力に吐き出した(出力した)からである。通常、標準出力に吐き出された情報は端末ウィンドウに表示されるようになっている。また、多くのコマンドは、ファイルからだけでなく、標準入力という経路からも入力データを受け付ける。特に指定しない場合、キーボードでタイプした文字列が標準入力に流れ込むようになっていることが多い。

コマンドが結果を標準出力に吐き出す性質を利用すると、リダイレクションによって、その結果を新しいファイルに格納したり、既存のファイルの末尾に付け加えたりすることができる。リダイレクト（redirect）とは、「・・・のあて名を変える」という意味だ。

標準出力に吐き出された内容を新しいファイルに格納するには > を利用する。例えば、old_file.txt の内容を cat を用いて new_file.txt に書き込むには、端末ウィンドウのプロンプトに対して次のようにする。

```
cat old_file.txt > new_file.txt ↵        （cat を使ってテキストファイルをコピー）
```

このとき、cat の出力は new_file.txt へリダイレクトされているので、端末ウィンドウには表示されない。

標準出力を既存のファイルの末尾につなげるように格納するには、>> を利用する。例えば、another_file.txt の中身を、先ほど作られた new_file.txt の末尾に追加するには、次のようにする。

```
cat another_file.txt >> new_file.txt ↵
                （cat を使って、another_file.txt の内容を new_file.txt の末尾に付加）
```

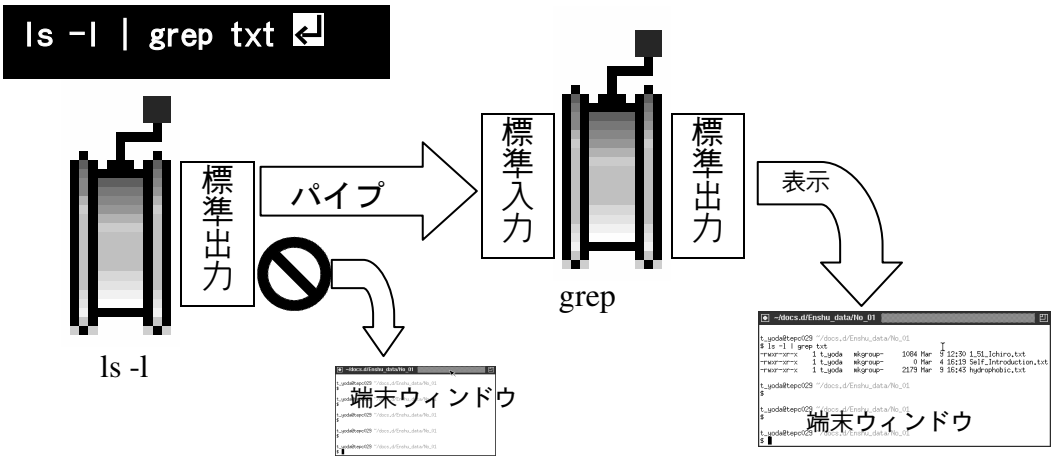
コマンドが標準出力へ吐き出した結果を、端末ウィンドウやファイルではなく、別のコマンド（やプログラム）の標準入力に流し込むこともできる。この機能をパイプという。

パイプを利用するときは | （縦棒）を使う。例えば、ls -l で表示されるファイルのリストの全てではなく、”txt”を含むファイル名の情報だけを表示させたいときには次のようにする。これの結果と、単に ls -l を実行したときの結果とを比較してみよう。

```
ls -l | grep txt ↵
```

なお、grep は、入力データ（ファイルまたは標準入力）の中から、指定した文字列を含む行のみを抜き出して表示する（標準出力へ吐き出す）コマンドである(後述)。

パイプの機能



2.c. ワイルドカード

シェルでは、ファイル名の指定のために、ワイルドカード(*) を使うことができる。例えば、次のように使う。

`ls -l *.pdb` ↵
(カレントディレクトリにある、ファイル名の末尾が .pdb であるファイル全てのリストの表示)

`rm *` ↵
(全てのファイルを削除。危険なので安易に実行しないこと)

`mv dir1/*.fasta dir2/` ↵
(ディレクトリ dir1 にある、ファイル名の末尾が .fasta のファイルを全て dir2 へ移動)

`more *genome*` ↵
(ファイル名に genome を含む全てのファイルを(1 ページずつ)表示)

＊ は任意の文字列の代替になるワイルドカードだが、任意の 1 文字の代替になるワイルドカードとして、? がある。例えば、次のように使う。

ls ?? ↵
(2 文字のファイル名を全て表示)

rm x?? ↵
(ファイル名の先頭が x の、３文字のファイル名のファイルを全て削除)

2.d. テキスト関連の UNIX コマンド（フィルタ）

2.d.1. cat

ファイルの中身を表示するコマンド cat が、ファイルの連結に使えることは以前説明したとおりである。例えば、次のような便利な使い方ができる。[2]

```
cat  seq1.fasta  seq2.fasta  seq3.fasta > seqs_123.fasta ↵
```

(seq1.fasta と seq2.fasta と seq3.fasta の内容を順につなぎ、seqs_123.fasta に出力)

```
cat  seq*.fasta  >  all_seqs.fasta ↵
```

(ファイル名が seq ではじまり、.fasta で終わる全てのファイルを順番に[3]連結し、all_seqs.fasta というマルチ FASTA 形式のファイルを作る)

```
cat  model5.pdb >>  models.pdb ↵
```

(model5.pdb の内容を、models.pdb の末尾に追加)

2.d.2. head, tail

head は、入力されたテキストの最初の 10 行のみを表示する。表示する行の数を変えることもできる。例えば、次のように使う。

```
head 1CLL.pdb ↵
```

(1CLL.pdb の最初の 10 行を表示)

```
tail 1CLL.pdb ↵
```

(1CLL.pdb の最後の 10 行を表示)

```
head -1  seq*.txt ↵
```

(ファイル名が seq で始まり、.txt で終わる全てのファイルの 1 行目を(全て)表示)

```
tail  -23  1CLL.pdb ↵
```

(1CLL.pdb の最後の 23 行を表示)

2 そのほかの便利機能として、-n オプション (行番号を付加して表示) がある。
3 ファイル名のアルファベット順

2.d.3. grep

grep は、指定された文字列を含む行だけを表示する。例えば、次のように使う。検索文字列では正規表現を使うこともできる。また、-n オプションを使うと、それが入力の何行目か、を表す数字も表示される。

```
grep CA 1CLL.pdb ↵
```

(1CLL.pdb で、文字列「CA」を含む行[4]のみを表示。)

```
grep “>” seqs_123.fasta ↵
```

(> を含む行のみを表示。シェルがリダイレクションのための＞として解釈してしまうことを防ぐために、引用符が必要。)

```
grep ‘^SEQRES’ 1UAO.pdb ↵
```

(1UAO.pdb で、行頭が SEQRES の行のみを表示)

```
grep -n ‘^SEQRES’ 1UAO.pdb ↵
```

(1UAO.pdb で、行頭が SEQRES の行のみを表示。それが何行目かも表示)

2.d.4. wc

wc は、入力された文字の総数とか、行の総数とか、単語の個数を表示するコマンドである。ここでは、行の総数を表示させる -l オプションを使う例のみをあげる。

```
wc -l 1UAO.pdb ↵
```

(1UAO.pdb の行数を表示)

```
grep "^ENDMDL " 1UAO.pdb | wc -l
```

(1UAO.pdb に含まれる ENDMDL 行の個数 (＝含まれる構造モデルの個数)を表示。^は、ENDMDL が行頭にあるときにだけマッチさせるためのおまじない)

4 α 炭素の座標の行がこれにあたる。

2.d.5. diff

diffは、指定された2つのファイルを比較し、どこが異なるか、を指摘する。２つのファイルの内容がまったく同一なら、何も表示しない。

```
diff ファイル１ ファイル２ ↵
```

2.d.6. sort

sortは、入力されたテキストの行の順序をソートする。通常は行の内容が文字コード順 (ほぼアルファベット順) になるように並べ替える。行全体を見て比較するのではなく、フィールドのみを比較したいときには -k オプションを、文字コードとしてではなく、数値として比較したいときには -n オプションをつける。

今、右のような内容のhanshin.txtというファイルがあるとする。

このデータファイルに対するsortコマンドのいろいろな実行例を挙げる。

53	Akahoshi	0.319
1	Toritani	0.276
4	Sheets	0.290
7	Imaoka	0.281
6	Kanemoto	0.325
39	Yano	0.274
9	Fujimoto	0.251
24	Hiyama	0.278

hanshin.txt

```
sort hanshin.txt ↵
```

(文字列として行と行を比較するので、4, 6 よりも 24, 39 の方が上になる。)

1	Toritani	0.276
24	Hiyama	0.278
39	Yano	0.274
4	Sheets	0.290
53	Akahoshi	0.319
6	Kanemoto	0.325
7	Imaoka	0.281
9	Fujimoto	0.251

hanshin5.txt

この結果をあとで使うのでファイルに保存しておこう。フィルタの実行結果を保存するときには次のようにしてリダイレクションを使えばよい。次の例では、結果がhanshin5.txtというファイルに保存される。

```
sort hanshin.txt > hanshin5.txt ↵
```

```
sort -b -k 2,2 hanshin.txt ↲
```

(第２フィールドの人名の順序にソートする。-k 2,2 は、オプションで２つめのフィールドを指定するときの正式な方法である[5]。フィールド同士を隔てる空白文字の個数が統一されていないファイルに対して -k オプションを適用する場合は、-b オプションを併用した方がよい。[6])

```
53  Akahoshi      0.319
9  Fujimoto       0.251
24  Hiyama        0.278
7  Imaoka         0.281
6  Kanemoto       0.325
4  Sheets         0.290
1  Toritani       0.276
39  Yano          0.274
```

```
sort -n -k 3,3 hanshin.txt ↲
```

（第３フィールドの打率を数値的に比較して、低い順にソートする。[7]）

```
9  Fujimoto       0.251
39  Yano          0.274
1  Toritani       0.276
24  Hiyama        0.278
7  Imaoka         0.281
4  Sheets         0.290
53  Akahoshi      0.319
6  Kanemoto       0.325
```

```
sort -n -k 3,3 -r hanshin.txt ↲
```

（打率の大きい順にソートする。−r を指定すると、並べる順序が逆順になる。）

```
6  Kanemoto       0.325
53  Akahoshi      0.319
4  Sheets         0.290
7  Imaoka         0.281
24  Hiyama        0.278
1  Toritani       0.276
39  Yano          0.274
9  Fujimoto       0.251
```

sort コマンドには他にもたくさんのオプションが用意されている。man sort を参照のこと。

2.d.7. uniq

uniq は、前後の行を比較して、まったく同じ内容の行が連続していたら、１つを残して他を削除する。uniq を利用する前に sort を使うことによって、応用の幅がひろがる。

次の例は、右に示す入力データに対して uniq を実行した結果である。

eibun.txt

```
This is a pen.
It is a pencil.
It is a pen.
This is a pin.
This is a pen.
This is a pencil.
```

5　この例では、-k 2 のように簡略表記できる。しかし、-n オプションと併用するときには本文に示したように書かないと、思わぬ結果が得られる場合がある。

6　詳しくは man sort を参照のこと。

7　この例から分かるように、-n オプションを使うと、数値の小さい順のソートになる。

```
uniq eibun.txt ↵
```
（結果は、eibun.txt と同じ。）

```
This is a pen.
It is a pencil.
It is a pen.
This is a pin.
This is a pen.
This is a pencil.
```

次の例は、パイプを使った sort と uniq の連携技の例である。

This is a pen. という行が一行だけになったことに注目。

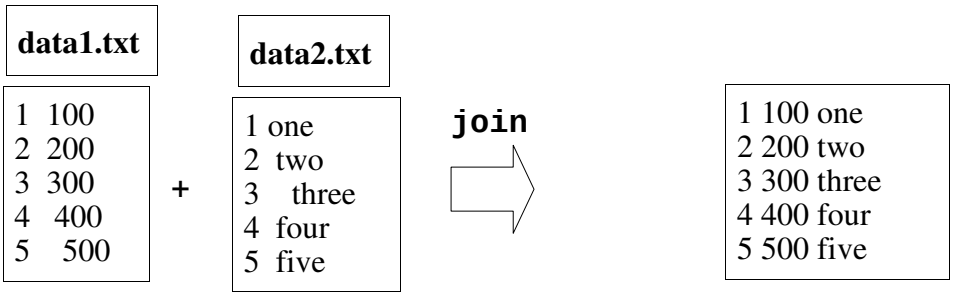
```
sort eibun.txt | uniq ↵
```
（sort してから uniq を実行した。）

```
It is a pen.
It is a pencil.
This is a pen.
This is a pencil.
This is a pin.
```

2.d.8. join

join は、2 つの表形式のファイルを横方向に連結する[8]。使いこなせばとても便利。

```
join data1.txt data2.txt ↵
```



2.d.9. paste

paste は、join の低機能版で、機械的に同じ行番号の行同士を横に連結する。また、-s オプションを付けると、入力ファイルから改行コードを削除する。

```
paste -s data1.txt ↵
```
（data1.txt から改行を取り除く）

```
1  100  2  200  3  300  4   400 5    500
```

8 デフォルトでは、両方のファイルの第 1(最左の)フィールド同士が一致する行同士を横方向に連結する。このとき 2 つめのファイルの第 1 フィールドは削られる。また、join を使うときには、事前に、両方の入力ファイルを sort -b でソートしておくこと。詳しくは、man join を参照。

2.d.10. split

split は、入力ファイルを指定した行数の断片に分割し、その断片を新しいファイルに書き込む。[9]

```
split -100 1UAO.pdb ↵
```

（1UAO.pdb の内容を 100 行毎に分割し、結果をそれぞれ新しいファイルに格納）

2.d.11. cut

cut は、入力データの各行から、指定された領域だけを取り出して表示する。-c オプションを使うと、各行の左から数えて何文字目(から何文字目まで)を表示するかを指定できる。[10]

```
cut -c18-20 1CLL.pdb ↵
```

（1CLL.pdb の各行の、左から 18 文字目から 20 文字目までを表示。実際に実行してみよ。1UAO.pdb についてもやってみよ。）

2.e 課題 1

以下の問に答えよ。

第 1 問

マルチ FASTA 形式のファイル(ファイル名は multi.fasta とする)から、ヘッダ行のみを取り出して表示するためのコマンド文字列を書け。

第 2 問

1UAO.pdb から、アミノ酸配列の情報が記されている、SEQRES レコードのみを取り出して表示するためのコマンド文字列を書け。

第 3 問

1UAO.pdb から、原子座標が書かれている ATOM レコードのみを取り出して表示するためのコマンド文字列を書け。

第 4 問

1CLL.pdb に HETATM レコードが何行あるかを知るためのコマンド文字列を書

9 新しくできるファイルの名前は、xaa, xab, xac のようになる。これの変え方などは、man split を参照。
10 他のオプションについては、man cut を参照。

け。パイプを使うとよい。

第5問

1UAO.pdb の ATOM レコードのうち、α 炭素のレコードのみを取り出して表示するためのコマンド文字列を書け。

第6問

このファイルのタンパク質のアミノ酸配列を表示するコマンド文字列を書け。まず、α 炭素の ATOM レコードだけを取り出すコマンド文字列を書き、パイプで他のコマンドにつなげると良い。

第7問

1CLL.pdb に含まれるアラニンの個数を知るためのコマンド文字列を書け。

第8問

第 6 問で、アミノ酸配列を 1 行で(つまり、改行コードを含まずに)表示するためのコマンド文字列を書け。

第9問

1UAO.pdb の ATOM レコードに記されている原子の x 座標の最小値を知るためのコマンド文字列を書け。

第10問

1UAO.pdb の第 500 行から第 520 行までを表示するコマンド文字列を書け。head や tail をうまく組み合わせること。

第11問

第 10 問で、1UAO.pdb における行番号も一緒に表示するにはどうすればよいか。

第12問

第 6 問のコマンド文字列を編集して、1CLL.pdb に含まれるアミノ酸の組成(どのアミノ酸が何個あるか)を表示するコマンド文字列を書け。最後に、uniq コマンドを-c オプションとともに使う[11]とよい。

11 そうするとどうなるのか、は、man uniq で確認せよ。

第 13 問

4 ページで登場した hanshin5.txt と、次に示す hanshin6.txt をもとにして、hanshin4.txt のような表を作るコマンド文字列を書け。

hanshin5.txt

```
1   Toritani      0.276
24  Hiyama        0.278
39  Yano          0.274
4   Sheets        0.290
53  Akahoshi      0.319
6   Kanemoto      0.325
7   Imaoka        0.281
9   Fujimoto      0.251
```

hanshin6.txt

```
1   144   7
24  117   8
39  137  19
4   135  19
53  143   1
6   144  39
7   144  29
9   117   1
```

hanshin4.txt

```
1 Toritani 0.276 144 7
4 Sheets 0.290 135 19
6 Kanemoto 0.325 144 39
7 Imaoka 0.281 144 29
9 Fujimoto 0.251 117 1
24 Hiyama 0.278 117 8
39 Yano 0.274 137 19
53 Akahoshi 0.319 143 1
```

第 14 問

1UAO.pdb の ATOM レコードのうち、原子の y 座標が負の数である行だけを表示するコマンド文字列を書け。

3. awk 入門

課題 1 の最後の問題は(おそらく)できなかったと思う。これは通常の UNIX コマンドの組合せだけでは実現困難な処理の例である。

もちろん perl のプログラムを作成すれば簡単に実現できるのだが、このような単純な処理を行うたびにいちいち perl プログラムを書くのは面倒だ。UNIX の既存のコマンドの組合せでは難しいが、perl プログラムを作成するほど複雑ではない、そういう場合に役に立つツールがここで紹介する awk である。

awk を使用するときの基本の構文を次に示す。

```
awk 'パターン{ 処理 }' 入力データファイル
```

上の構文で awk を実行すると、入力データファイルが一行づつ読み込まれ、もしその行が「パターン」の部分に記述された条件に合致していれば、その行に対して、「処理」の部分に記述された処理を行う。

例えば、課題 1 の第 15 問は、次のようにすれば解決だ。

```
awk ‘$1=="ATOM"&&$8<0{print}' 1UAO.pdb ↵
```

(1UAO.pdb の ATOM レコードのうち、原子の y 座標が負の数である行だけを表示)

`$1=="ATOM"&&$8<0` が「パターン」である。これは、行の第１要素(`$1`)が「`ATOM`」であり、かつ(`&&`)、第８要素(`$8`)が負の数である(`<0`)という条件を表している。

上の例における、「処理」[12]は、`print` である。読んで字のごとく、「表示せよ」という意味である。

パターンを省略した場合は、入力された全てのデータ行に対して「処理」を実行する。例えば、次のコマンド文字列を実行してみよ。

```
awk  ‘{print $1}’ 1UAO.pdb ↵
```

また、{ 処理 }を省略した場合は、{print}と書いた場合と同じように動作する。よって、課題１の第 15 問は、次のようにするのでも良い。

```
awk ‘$1=="ATOM"&&$8<0' 1UAO.pdb ↵
```

(1UAO.pdb の ATOM レコードのうち、原子の y 座標が負の数である行だけを表示)

awk の使い方の詳しい内容は、実習時に別に配布するプリントを見よ。また、次に挙げるのは、関連する URL である。

GNUawk の日本語マニュアル　http://www.kt.rim.or.jp/%7Ekbk/gawk/gawk_toc.html

awk の便利な一行野郎 http://www.kt.rim.or.jp/%7Ekbk/gawk/gawk_7.html#SEC45

12 awk では、この「処理」の部分のことを、「アクション」と呼ぶことが多い。