

UNIX基礎

基礎生物学研究所

ゲノムインフォマティクス・トレーニングコース
2016春

準備編・UNIXとRの基礎

三輪朋樹 (miwa@nibb.ac.jp)

2

UNIXを使う理由

- UNIXでしか使えないアプリケーション
 - 最新の研究用ソフト
 - 並列化・大容量メモリ対応ソフト
- たくさんの処理を一度に行う
 - スクリプトを用いたコマンドの連續実行
- 自作プログラム
 - シェルスクリプト, Perl, Ruby, バイオ系ライブラリ
- Webサーバ、データベースサーバ
 - 高い安定性
 - apacheやmySQL, Postgresなどのフリーウェア

PCでUNIXを使うには

Mac	OSX自体がUNIX (#1)	アプリケーション→ターミナルを起動 UNIX端末として使用できる
	リモートログイン	UNIXサーバへリモートログイン ターミナルからsshを使用する
Windows	Cygwin	Windows上で動作するUNIXライクな環境
	VMware + Linux	仮想マシンを構築してLinuxそのものをインストールする
	リモートログイン	UNIXサーバへリモートログイン TeraTermからsshを使用する

#1) フリーウェアなどのインストールが必要な場合は「OSXでのUNIX環境構築方法」を参照

実習 1

● OSXのUNIX環境を確認する

1. 画面最下部にあるDockメニューから「ターミナル」を起動する。



(ターミナルの在処は、アプリケーション/ユーティリティ)

講習を始める前に

● コマンドプロンプト

- 端末に表示されている"\$"や "%"などの記号

今回の環境は `dh00-216:- nibb$`

- コマンド入力待ちの状態を表す

続けてコマンドを入力し、改行キーで実行する

● ASCII文字

- コマンドの入力は全て半角のASCII文字を使用

- 入力文字が全角になる日本語IMEはOFFにする

講習を始める前に

● 本講習では次のディレクトリを使用します

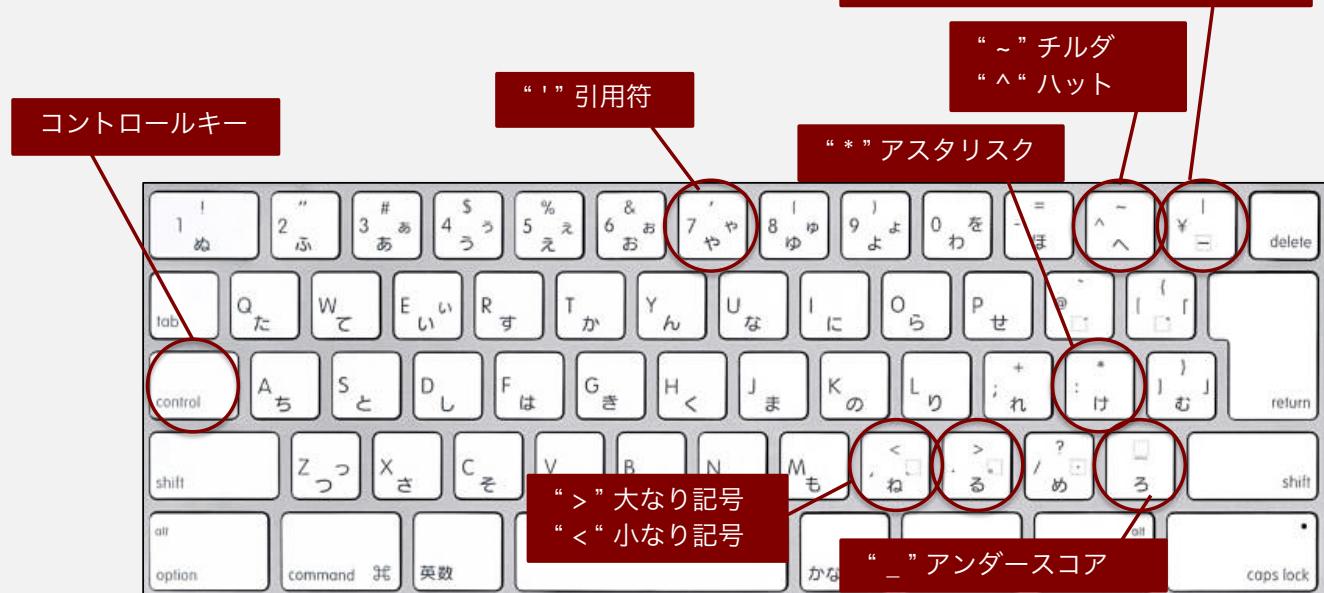
```
~/data/1_unix  
~/unixtest
```

キーボード配置の確認

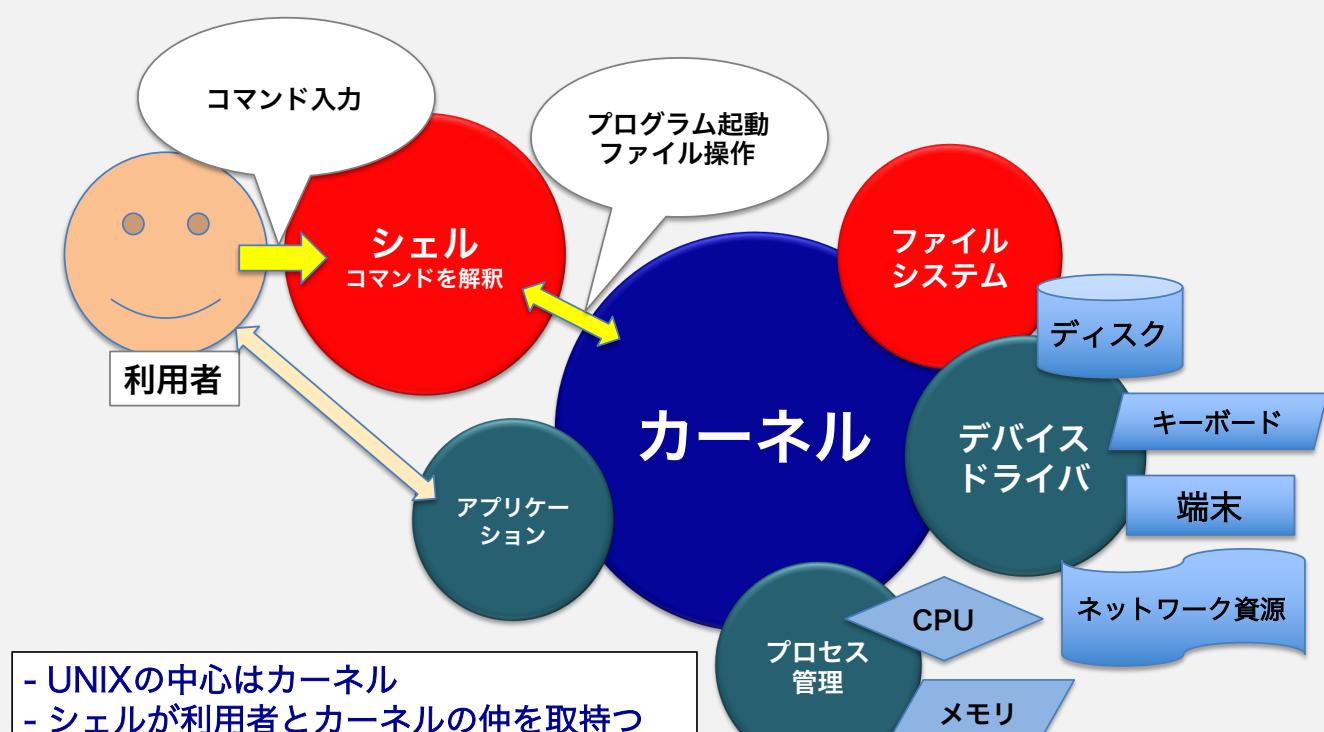
- 普段使用しない記号を多用します。

– キーの位置を確認しましょう。

“¥” バックスラッシュ(¥)
“|” 縦棒、バーティカルバー



UNIXオペレーティングシステム

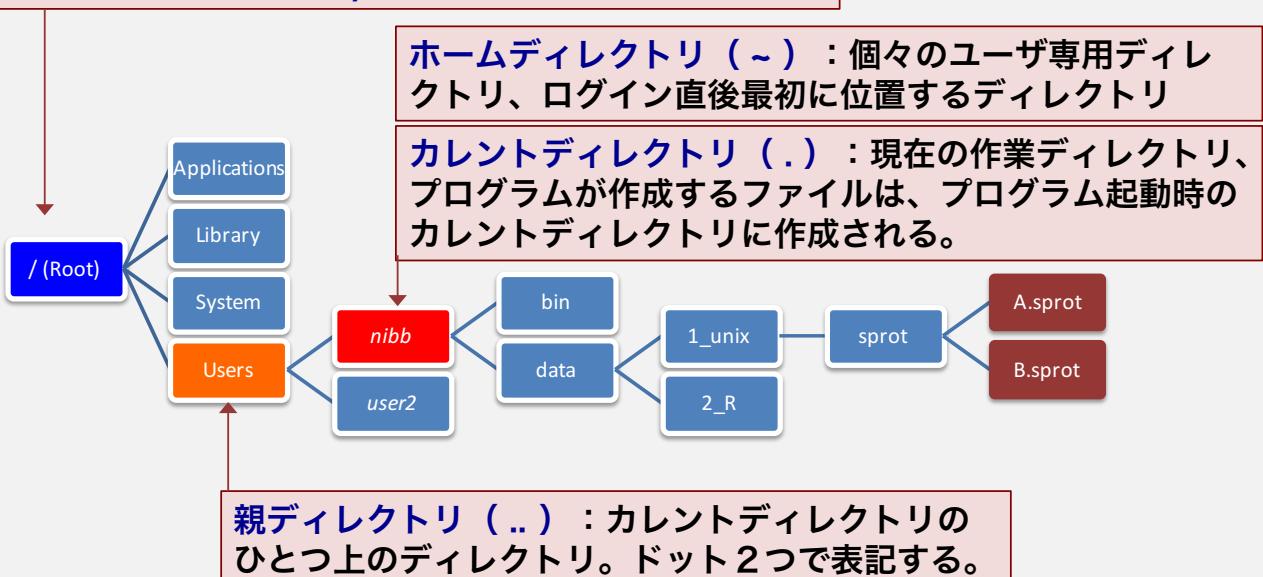


ファイルシステム

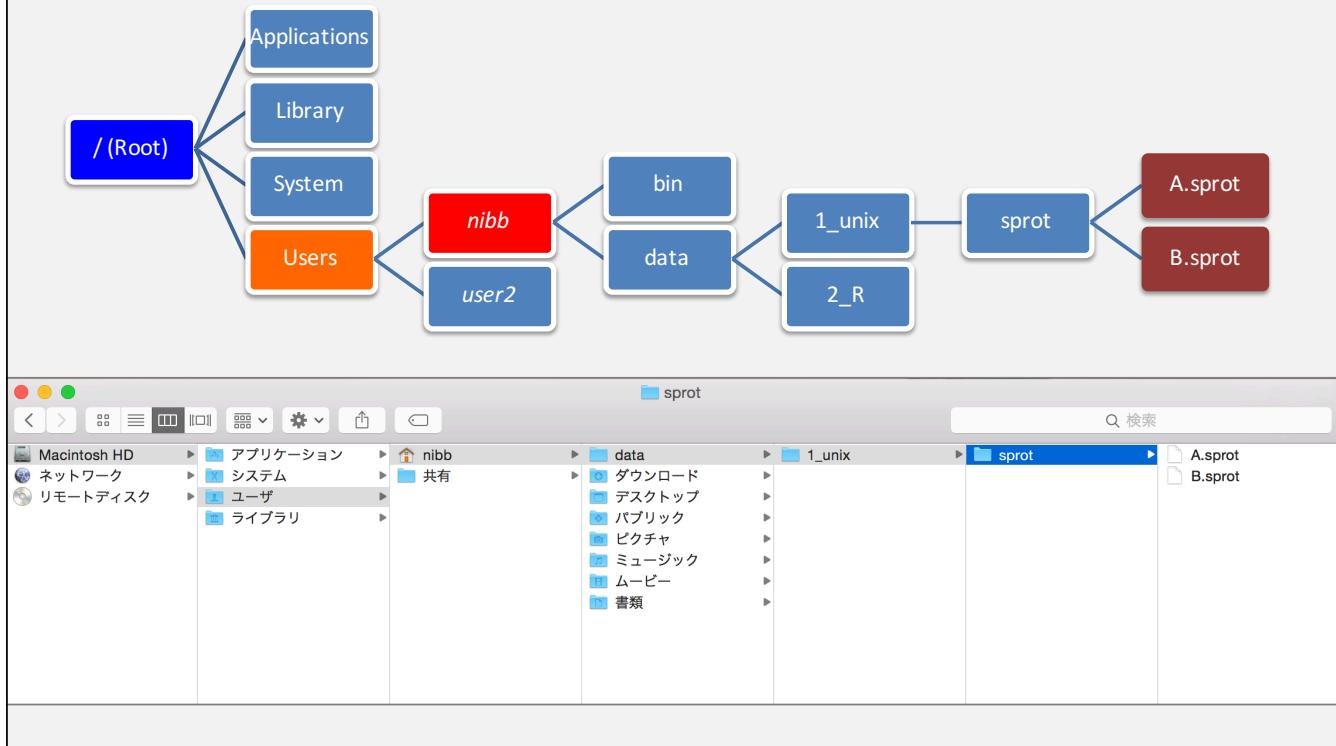
階層型ディレクトリ

トップのルートディレクトリ下に、子ディレクトリ、孫ディレクトリがあり、ファイルを配置する

ルートディレクトリ (/) : ファイルシステムの頂点



階層型ディレクトリ



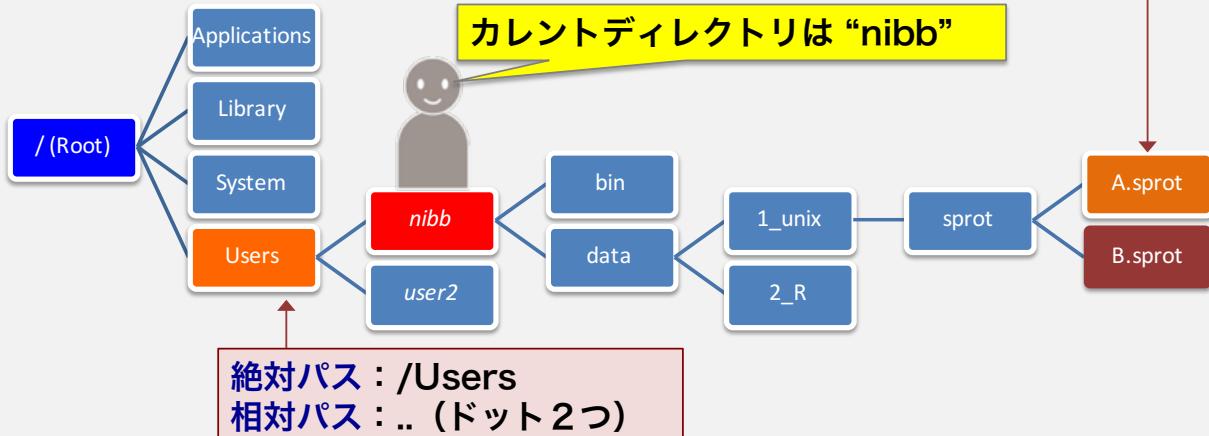
ファイル／ディレクトリの指定方法

- **パス (Path)**
 - ファイル やディレクトリを指定する記述方法
 - ディレクトリをスラッシュ “ / ” で区切る
 - 「絶対パス」と「相対パス」2通りの記述方法
- **絶対パス**
 - ルートディレクトリから目的のファイルやディレクトリへの道筋の記述
 - 行頭は必ずルートディレクトリ “ / ” となる
 - 例) /Users/nibb/data/1_unix/sprot/A.fasta
- **相対パス**
 - 起点となる現在位置から目的のファイルやディレクトリへの道筋の記述
 - 行頭はスラッシュ “ / ” 以外で始まる
 - 例) data/1_unix/sprot/A.fasta (カレントディレクトリは /Users/nibb)
 - 上位ディレクトリはドット2つ “ .. ” で記述する
 - 例) ../user2 (同じ階層の user2 ディレクトリ)

ファイル／ディレクトリ名の指定方法

nibb がログイン直後 : Users と A.sprot のパス

絶対パス : /Users/nibb/data/1_unix/sprot/A.sprot
相対パス : data/1_unix/sprot/A.sprot



ディレクトリの中身を見る (ls)

- **ls**
 - カレントディレクトリの内容（ファイル名のリスト）を表示する
- **ls ディレクトリ名**
 - 指定したディレクトリの内容を表示する

```
$ ls data    dataディレクトリの内容を表示
$ ls /    ルートディレクトリの内容を表示
$ ls ..  ひとつ上のディレクトリの内容を表示
$ ls .   カレントディレクトリの内容を表示 (lsと同じ)
```
- **ls -F**
 - ファイル名の末尾に種類に応じた記号を付けて表示する

```
/ : ディレクトリ、@ : シンボリックリンク、* : 実行権付きファイル
```
- **ls -a**
 - ファイル名の先頭がドット（.）で始まる隠しファイルを表示する

```
.login : ログイン時に実行される処理を記述したファイル
.bash_profile, .bashrc : シェル起動時に実行される処理を記述したファイル
```

実習2

● ディレクトリの中身を見る (ls)

1. ホームディレクトリ上でlsと入力してリターンキーを押す (=実行)

2. 続いて下記のコマンドもそれぞれ実行する。

\$ ls .. ホームディレクトリのひとつ上の中身を見る

\$ ls / ルートディレクトリの中身を見る

3. 隠しファイルを表示する。

\$ ls -a

補完機能

● コマンド名やファイル／ディレクトリ名の補完

- ファイル名を途中まで入力して、タブキーを押す
“ls u” と入力して、タブキーを押す
- 一意に決まらない場合は、一意になるところまでを展開する
“ls data/1_unix/sprot/143”と入力後タブキーを押す
- それ以上展開できない場合は再度（つまり2回）タブキーを押すと、候補ファイルの一覧を表示する
“ls data/1_unix/sprot/1433”の状態で2回続けてタブキーを押す

● 以下のメリットがあるので積極的に活用すること

- キーボード入力を省略できる
- 素早い入力が可能
- 複雑なファイル名の入力ミス防止
- うろ覚えのファイルでも入力できる
- コマンド名も補完できる

実習3

● ファイル名の補完

- ホームディレクトリ上で `ls da` と入力後、`<tab>` (タブキー) を押してファイル名の補完機能を試してみる。
- 続けて `ls data/1_unix/sprot/143` まで補完機能を使って動作を確認する。

```
$ ls data/1 <tab>
$ ls data/1_unix/
$ ls data/1_unix/sp <tab>
$ ls data/1_unix/sprot/
$ ls data/1_unix/sprot/143 <tab>
$ ls data/1_unix/sprot/1433
$ ls data/1_unix/sprot/1433 <tab><tab>
.... (候補一覧表示)
$ ls data/1_unix/sprot/1433B_H <tab>
$ ls data/1_unix/sprot/1433B_HUMAN.
$ ls data/1_unix/sprot/1433B_HUMAN.f <tab>
$ ls data/1_unix/sprot/1433B_HUMAN.fasta
```

コマンドヒストリと編集

● 過去に実行したコマンドの呼び出しと編集

キーバインド	説明
Control + p (↑キー)	ひとつ前のコマンド (<u>P</u> revious)
Control + n (↓キー)	ひとつ後のコマンド (<u>N</u> ext)
Control + b (←キー)	カーソルを左に移動 (<u>B</u> ack)
Control + f (→キー)	カーソルを右に移動 (<u>F</u> orward)
Control + a	カーソルを行頭に移動 (<u>st</u> <u>A</u> rt)
Control + e	カーソルを行末に移動 (<u>E</u> nd)
Control + u	行全体を削除しバッファへコピー
Control + k	カーソルから後を削除しバッファへコピー
Control + y	バッファの内容をペースト
Control + l (エル)	現カーソル行を画面上部に移動、画面消去

ファイル名のみ変更するなど長い入力行の一部を修正できるので、活用すること

実習4

● コマンドヒストリ

1. **Control + P** (コントロールキーとPを同時に押す) を何度か入力する。
2. **Control + N** を何度か入力する。
3. 表示されたコマンドラインのカーソル移動を行う。

Control + B

Control + F

Control + A

Control + E

ディレクトリを移動する (cd)

● cd ディレクトリ名

- 指定したディレクトリに移動する
- カレントディレクトリの変更

\$ cd data dataディレクトリに移動

\$ cd .. ひとつ上のディレクトリに移動

\$ cd ~/data ホーム下のdataディレクトリに移動

● cd

- ディレクトリ名を省略すると、ホームディレクトリに移動する

● pwd

- カレントディレクトリの確認

実習5

●ディレクトリを移動する

1. カレントディレクトリを確認する。

```
$ pwd
```

2. ディレクトリdata/1_unixに移動してpwdを入力する。

```
$ cd data/1_unix
```

```
$ pwd
```

```
$ ls
```

3. sprotディレクトリに移動し、カレントディレクトリを確認する。

```
$ cd sprot
```

```
$ pwd
```

ワイルドカード

- ファイル名を指定するときに使う「任意の文字」
- パターンマッチにより複数のファイル名を一度に指定

- アスタリスク (*) : 任意の文字列 (0文字以上)

```
$ ls *.fasta
```

```
$ ls *_HUMAN*
```

- クエスチョン (?) : 任意の1文字

```
$ ls 1A2?_HUMAN.fasta
```

- [文字列] : []に含まれる文字中の1文字

[12345]は[1-5]と書くこともできる

[!文字列]で含まれない1文字

```
$ ls 1A2[1-5]*.fasta
```

- {文字列1, 文字列2} : "文字列1"または"文字列2"

```
$ ls 1A25_HUMAN.{fasta,phylip}
```

実習6

● ワイルドカード

(カレントディレクトリは ~/data/1_unix/sprot)

- 下記コマンドを実行して、ワイルドカードの動作を確認する。

```
$ ls *.fasta
$ ls *_HUMAN*
$ ls 1A2[1-5]*.fasta
$ ls 1A25_HUMAN.{fasta,phylip}
$ ls *
```

ファイルの内容を一括表示する(cat)

● cat ファイル名

- con-cat-nate

つなぐ、連結する

- ファイルの内容を表示

```
$ cat 1A25_HUMAN.fasta
```

- ファイル名を複数指定すると内容を連結して表示

```
$ cat *.fasta
```

実習7

● ファイルの内容を一括表示する。

(カレントディレクトリは `~/data/1_unix/sprot`)

1. 下記コマンドを実行する。

```
$ cat 1A25_HUMAN.fasta
$ cat *.fasta
```

ファイルの部分表示(head,tail)

● **head** [-行数] ファイル名

- ファイルの先頭から指定した行数を出力
- 行数を省略すると10行出力

```
$ head 1A25_HUMAN.sprot
```

- 巨大なファイルの内容を簡単に確認する場合に便利

● **tail** [-行数] ファイル名

- ファイルの最後から指定した行数を出力

```
$ tail -20 1A25_HUMAN.sprot
```

- “`-n +行数`”とすると、先頭から数えて指定された行以降を出力

```
$ tail -n +2 1A25_HUMAN.fasta
```

(FASTAファイルの配列のみ表示)

実習8

● ファイルの部分表示

(カレントディレクトリは `~/data/1_unix/sprot`)

1. 下記コマンドを実行する。

```
$ head 1A25_HUMAN.sprot
$ tail -20 1A25_HUMAN.sprot
$ cat 1A25_HUMAN.fasta
$ tail -n +2 1A25_HUMAN.fasta
```

ファイルの中身を見る(less)

● less ファイル名

- ファイルの内容を閲覧する
- ページの移動には独自のキー操作が必要

キー操作	説明
f, SPACE	1ページ先へ進む
b	1ページ前へ戻る
j, k	1行づつ前(j)後(k)へ移動
g, G	ファイルの先頭(g)、末尾(G)へ移動
/文字列	文字列の検索、n, Nで前後のヒット行へ移動
q	終了

実習9

● ファイルの内容を見る

(カレントディレクトリは ~/data/l_unix/sprot)

1. 1433B_HUMAN.sprotの内容をlessコマンドで見る。

```
$ less 1433B_HUMAN.sprot
```

2. ページを順に表示して、元に戻す。

```
<space> , b , j , k
```

3. ファイルの末尾に移動してから、先頭に戻る。

```
G , g
```

4. "binding"文字列を検索する。

```
/binding
```

5. 次にヒットする場所に移動する。

```
n
```

6. lessを終了する。

```
q
```

ディレクトリの作成と削除 (mkdir, rmdir)

● mkdir ディレクトリ名

- 新規ディレクトリの作成

```
$ mkdir unixtest
```

● rmdir ディレクトリ名

- ディレクトリの削除

```
$ rmdir unixtest
```

- ディレクトリ内にファイルがあると削除できない

実習10

●ディレクトリの作成と削除

(カレントディレクトリは ~/data/1_unix/sprot)

1. ホームディレクトリに移動後、実習用のディレクトリ `unixtest` を作成する。

```
$ cd  
$ pwd  
$ mkdir unixtest
```

2. 作成した `unixtest` ディレクトリに移動し、カレントディレクトリを確認する。

```
$ ls  
$ cd unixtest  
$ pwd
```

ファイル/ディレクトリのコピー (cp)

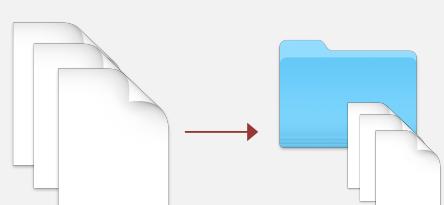
● cp file1 file2

- `file1` のコピーを `file2` として作成



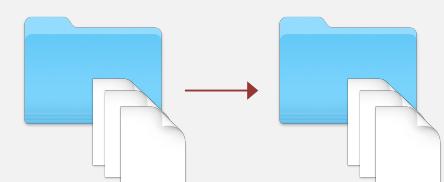
● cp file1 [file2...] dir

- `file1` (`,file2,...`) のコピーを `dir` 内に作成



● cp -r dir1 dir2

- `dir1` を `dir2` としてディレクトリごとコピーを作成
- コピー先のディレクトリが存在する場合は、そのディレクトリ以下に `dir1` として作成



同一名ファイルが存在すると上書きされるので注意

実習11

● ファイルのコピー

(カレントディレクトリは ~/unixtest)

1. ~/data/1_unix/sprot/1433B_HUMAN.sprotをカレントディレクトリにコピーする。

```
$ cp ~/data/1_unix/sprot/1433B_HUMAN.sprot .
```

2. コピーした1433B_HUMAN.sprotを更にcopyfileという名前でカレントディレクトリにコピーする。

```
$ cp 1433B_HUMAN.sprot copyfile
```

3. Fastaという名前のディレクトリを作成する。

```
$ mkdir Fasta
```

4. ~/data/1_unix/sprot/以下のfastaファイル (.fasta拡張子を持つファイル)だけを、3.で作成したFastaディレクトリにワイルドカードを使用してコピーする。

```
$ cp ~/data/1_unix/sprot/*.fasta Fasta
```

ファイル/ディレクトリ名の変更、移動 (mv)

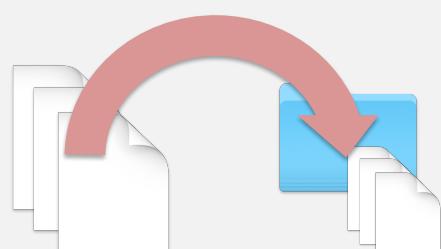
● mv file1 file2

- file1の名前を file2 に変更



● mv file1 [file2...] dir

- file1 (,file2,...) をdir内に移動



同一名ファイルが存在すると上書きされるので注意

実習12

● ファイル名の変更とファイルの移動

(カレントディレクトリは ~/unixtest)

- copyfileの名前をnewfileに変更する。

```
$ mv copyfile newfile
```

- 新規にHumanディレクトリを作成。

```
$ mkdir Human
```

- Fastaディレクトリ内のHUMANが付いたファイルのみをHumanディレクトリに移動する。

```
$ ls Fasta/*HUMAN*
```

```
$ mv Fasta/*HUMAN* Human
```

実際にワイルドカードを使用して移動や削除を行う場合は、事前にlsで対象のファイルを確認すること。

シンボリックリンクの作成 (ln -s)

● ln -s ファイル/ディレクトリ名 シンボリックリンク名

- ファイル/ディレクトリの「別名」を作成
- OSXのエイリアスやWindowsのショートカットに相当

```
$ ln -s dir/file1 .
```

dir/file1のシンボリックリンクをカレントディレクトリ(.)に作成

- オリジナルのファイルが移動・消去されると、シンボリックリンクを通した参照はできなくなる

● 用途

- 他ディレクトリへの容易なアクセス

```
$ ln -s ~/data/l_unix/sprot .
```

よく使うディレクトリをホーム配下にシンボリックリンクする

実習13

● シンボリックリンクの作成

(カレントディレクトリは ~/unixtest)

- Human/1433B_HUMAN.fastaのシンボリックリンクをカレントディレクトリに作成する。

```
$ ln -s Human/1433B_HUMAN.fasta .
```

- 作成したシンボリックリンクファイルの内容を表示する。

```
$ cat 1433B_HUMAN.fasta
```

`mv`や`cp`と同様に、最後の引数がディレクトリの場合は、そのディレクトリ上にオリジナルと同じ名前のシンボリックリンクが作成される。

参考：-sオプションを付けない場合は、ハードリンクと呼ばれ、オリジナルファイルを移動・消去してもリンクを通した参照が維持される。ただし、ディレクトリのハードリンクが作れないことや、ボリュームを跨いだハードリンクが作れないなどの制約がある。

ファイル情報の表示 (ls -l)

● ls -l ファイル／ディレクトリ名

– ファイルの大きさや更新日などの情報をリスト表示

● ls -lt ファイル／ディレクトリ名

– 日付順に表示

```
$ ls -l 1433B_HUMAN.sprot
```

-rw-r--r--	1	nibb	staff	21675	2011-03-09 05:23	1433B_HUMAN.sprot
		アクセス権				
		ハードリンク	所有者			
		数				
		ファイルの種類		グループ	サイズ	更新日付
		- : 通常のファイル				ファイル名
		d : ディレクトリ				
		l : シンボリックリンク				

実習14

● ファイル情報の表示

(カレントディレクトリは ~/unixtest)

1. カレントディレクトリの詳細情報リストを表示する。

```
$ ls -l
```

2. 詳細情報を更新日時順に表示する。

```
$ ls -lt
```

ファイルのアクセス権

● 全てのファイル／ディレクトリに付けられている

- OSXやWindowsでは気にしないが、UNIXでは意外と重要

● 誰に対して、何ができるか

- 誰に対して：

- (u) ファイルの所有者

- (g) 同一グループの利用者

- (o) その他

- 何ができるか：

- 「読む(r)」「書く(w)」「実行する(x)」

- 「実行する(x)」

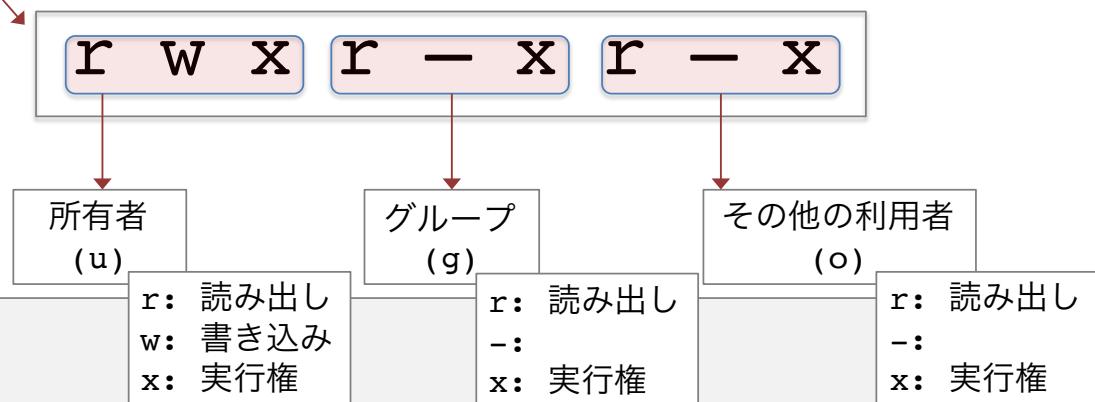
ファイルをプログラムとして実行できる

ディレクトリの場合は「移動が可能」

アクセス権モードの確認

● ls -l ファイル／ディレクトリ名

```
$ ls -l myfile
-rwxr-xr-x 1 nibb staff 21675 2011-03-09 05:23 myfile
```



アクセス権モードの変更

● chmod [モード] ファイル名

- ファイルのアクセス権モードの変更

- モードの書式を記述する

```
$ chmod u+x file    所有者に実行権を与える
```

```
$ chmod go-rwx file 所有者以外のアクセスを禁止
```

- 8進数3桁で各ユーザのレベルを列挙

```
$ chmod 600 file 所有者のみ読み書きできる
```

	所有者(u)			グループ(g)			その他(o)		
パーミッション	r	w	x	r	w	x	r	w	x
8進数	4	2	1	4	2	1	4	2	1
設定値	合計値			合計値			合計値		

実習15

● アクセス権の確認と変更

(カレントディレクトリは ~/unixtest)

1. カレントディレクトリにあるディレクトリ、ファイルのアクセス権を確認する。

```
$ ls -l
```
2. `copyfile`のアクセス権から自身のRead権限を削除する（本来はこのようなことはしませんが）。

```
$ chmod u-r copyfile
```
3. `less`で`copyfile`を読む。

```
$ less copyfile
```

エラーになります。
4. 再度`copyfile`のアクセス権に自身のRead権限を追加する。

```
$ chmod u+r copyfile
```
5. `less`でエラーなく`copyfile`が読めることを確認する。

```
$ less copyfile
```

ファイルの削除 (rm)

● rm ファイル名 ...

- 指定したファイルを削除する

● rm -rf ディレクトリ名

- ディレクトリの中身を確認なしで全て消去した上で
ディレクトリを削除する
- 確認しながら消去するには `rm -ri` とする

● 「ゴミ箱」はありません

- 削除したファイルを復活することはできません
- ワイルドカードを使用したファイルの削除は注意
必ず `ls` で対象ファイルを確認しましょう

実習16

● ファイルの削除

(カレントディレクトリは ~/unixtest)

1. newfileを削除する。

```
$ rm newfile
```

2. Fastaディレクトリ全体を削除する。

```
$ rm -rf Fasta
```

コマンドの実行

UNIXコマンドの基本形

ls -l gbre1.txt

コマンド

オプション

コマンドの動作を変えるスイッチ

オペランド

ファイルなどコマンドが処理する対象

引数

コマンドマニュアルの参照 (man)

● man コマンド名

\$ man ls

[]で囲まれている引数は省略可能

NAME

ls - list contents of directory

SYNOPSIS

ls [-RadLCxmlnogrtucpFbqisf1AMSDP] [names]

DESCRIPTION

For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information

指定可能なオプション

下線付きは変数

is sorted alphabetically by default. When the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. ls processes supplementary code set characters according to the locale specified in the LC_CTYPE and LC_COLLATE environment variables [see LANG on environ(5)], except as noted under the -b and -q options below.

man -k keyword で keyword に関するコマンド一覧を表示

実習17

● マニュアルの参照

(カレントディレクトリは ~/unixtest)

- lsコマンドのマニュアルを表示する。

```
$ man ls
```

行数・単語数のカウント(wc)

● wc ファイル名

- ファイルの行数、単語数、文字数を出力

```
$ wc 1433B_HUMAN.fasta
```

```
6 25 481 1433B_HUMAN.fasta
```

(6行、25単語、481文字)

- ファイル名を省略するとキーボード入力に対して実行

```
$ wc
```

```
This is a pen.
```

(Control-D)

```
1 4 15
```

(1行、4単語、15文字)

実習18

● 行数・単語数のカウント

(カレントディレクトリは ~/unixtest)

1. `1433B_HUMAN.fasta` の単語数をカウントする。

```
$ wc 1433B_HUMAN.fasta
```

2. キーボードから "This is a pen." を入力し、その単語数をカウントする。

```
$ wc
```

```
This is a pen.
```

(Control-D)

パターン検索(grep)

● 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

```
$ grep GO 1433B_HUMAN.sprot
```

`1433B_HUMAN.sprot` から GO を含む行を検索

```
$ grep ^FT 1433B_HUMAN.sprot
```

`1433B_HUMAN.sprot` から FT で始まる行を検索

("^" は行の先頭を意味する)。

- `grep -v` とすると (オプション -v) パターンを含まない行を出力する。

- ファイル名を省略すると、やはり端末から文字列を読み込んでパターンを検索する。

実習19

● パターン検索

(カレントディレクトリは ~/unixtest)

1. 1433B_HUMAN.sprotの内容をlessで確認し、GOとFTを検索する。

```
$ less 1433B_HUMAN.sprot
```

2. 1433B_HUMAN.sprotから"GO"が含まれる行を表示する。

```
$ grep GO 1433B_HUMAN.sprot
```

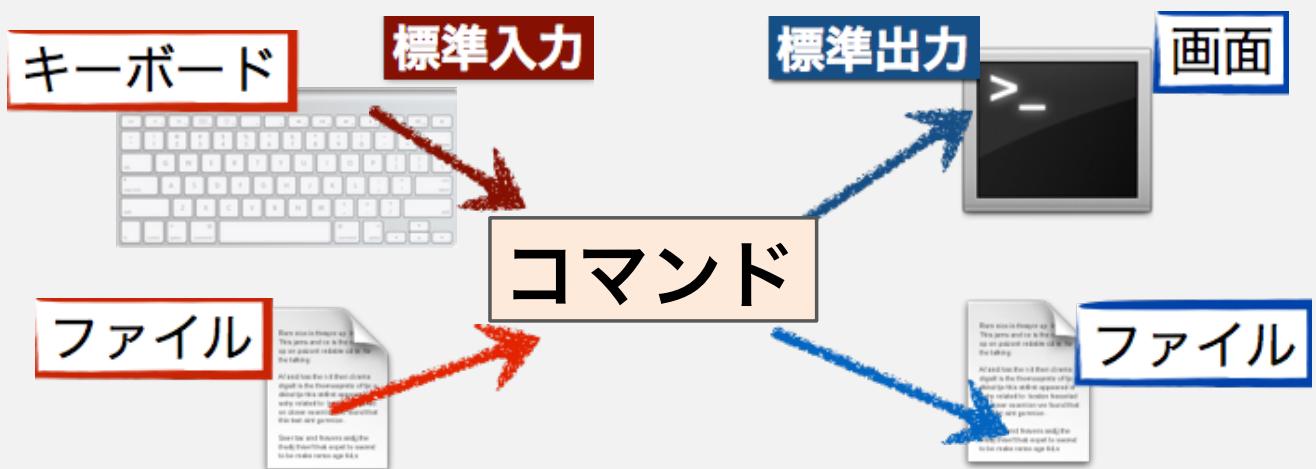
3. 1433B_HUMAN.sprotからFeature table data(FT)の行を表示する。

```
$ grep ^FT 1433B_HUMAN.sprot
```

入出力のリダイレクション

● リダイレクション

- コマンドへの入力元、出力先を切換える機能
- 標準入力=キーボード、標準出力=画面



リダイレクションの例・出力

- 端末画面に出力

```
$ grep GO 1433B_HUMAN.sprot
```

- ファイルに保存 (>)

- コマンドの右に "`> filename`" を加える
- 結果を `GO_count` というファイルに保存

```
$ grep GO 1433B_HUMAN.sprot > GO_count
```

(同名ファイルがある場合は上書きされる)

- 存在するファイルに追加書き (>>)

- コマンドの右に "`>> filename`" を加える

```
$ grep GO * 1433?_HUMAN.sprot >> GO_counts
```

(ファイルが存在しない場合は作成される)

リダイレクションの例・入力

- ファイルから入力

- コマンドの右側に "`< filename`" を加える
- wc コマンドで、`GO_count` ファイルの行数を数える

```
$ wc < GO_count
```

(`wc GO_count` と同じ)

実習20

● リダイレクト

(カレントディレクトリは ~/unixtest)

1. `1433B_HUMAN.sprot`から "GO" が含まれる行を検索して、その結果をリダイレクト（出力）を使用して `GO_count` ファイルに保存。

```
$ grep GO 1433B_HUMAN.sprot      (ファイル保存前に確認)
```

```
$ grep GO 1433B_HUMAN.sprot > GO_count
```

2. ファイルの中身を `less` で確認する。

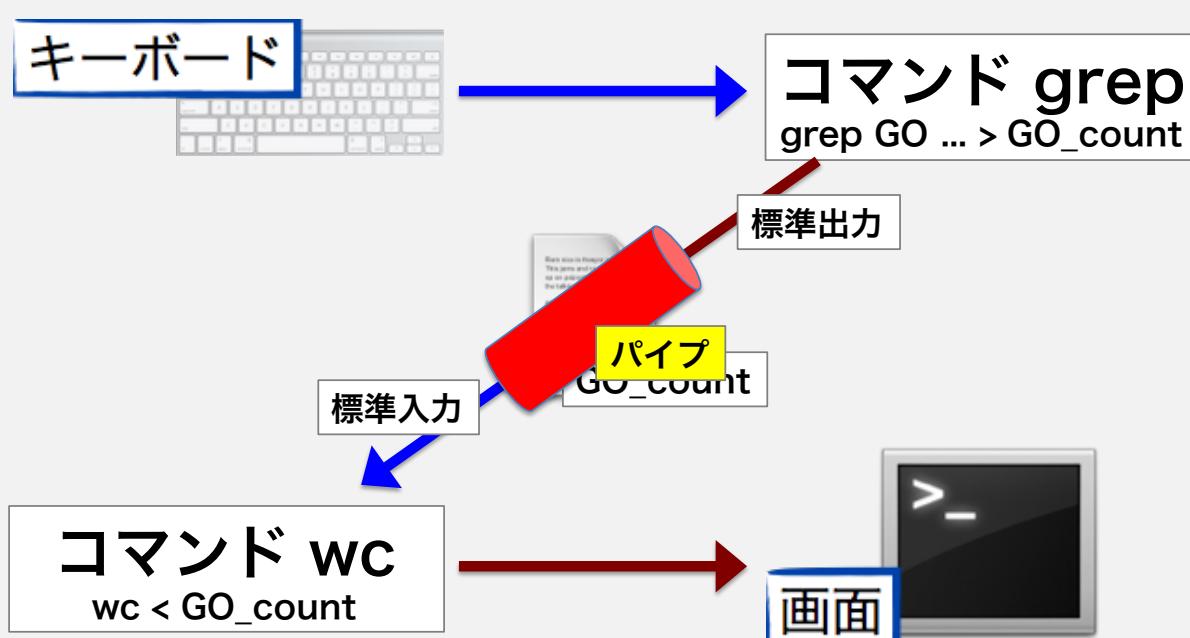
```
$ less GO_count
```

3. リダイレクト（入力）を使用して `GO_count` の単語数を数える。

```
$ wc < GO_count
```

パイプによるコマンドの結合

● コマンドの標準出力→標準入力をつなげる



パイプの例

- パイプはコマンド間を縦棒(|)でつなげる

- grep の出力行数をwcで数える

```
$ grep GO 1433B_HUMAN.sprot | wc
```

- grep の結果を less で見る

```
$ grep ^FT 1433B_HUMAN.sprot | less
```

- grepの結果をさらにgrepして絞り込む。

```
$ grep ^FT 1433B_HUMAN.sprot | grep HELIX  
| less
```

- コマンドはいくつでも結合できる

実習21

- パイプ

(カレントディレクトリは ~/unixtest)

(検索対象は1433B_HUMAN.sprot)

1. “GO”を検索した結果をパイプを使用してwcで単語数をカウントする。

```
$ grep GO 1433B_HUMAN.sprot | wc
```

2. “FT”から始まる行を検索した結果をパイプを使用してlessで表示する。

```
$ grep ^FT 1433B_HUMAN.sprot | less
```

3. “FT”から始まる行を検索した結果から更に“HELIX”を検索してlessで表示する。

```
$ grep ^FT 1433B_HUMAN.sprot | grep HELIX | less
```

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
- 例) data/bin/testpg の内容

```
#!/bin/sh
pwd
ls -la
```

- 実行

- ファイルを単体で実行させるためには実行権が必要

```
$ chmod +x testpg
```

- 実行はパスを省略せずに入力

```
$ ./testpg
```

もしくは /Users/<user>/data/1_unix/sprot/testpg

- コマンドサーチパスに登録されている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。

実習22

- シェルスクリプト

(カレントディレクトリは ~/unixtest)

1. ~/data/1_unix/sprot/testpgをカレントディレクトリにコピーし中身を確認する。

```
$ cp ~/data/1_unix/sprot/testpg .
$ less testpg
```

2. testpgを実行する。 (相対パス／絶対パスのいずれかを付けて実行)

```
$ ./testpg ("Permission denied"のエラー)
```

3. 実行権がなくエラーとなるため、実行権を付与する。

```
$ ls -l testpg    (実行権の確認)
$ chmod +x testpg
$ ls -l testpg
$ ./testpg
```

4. パスを付けずに実行した場合の動作を確認する。

```
$ testpg ("command not found"のエラー)
```

コマンドサーチパス

- コマンドサーチパス
 - パス無しのコマンドは、「コマンドサーチパス」リストから探し出される
例) `ls` コマンドは `/bin/ls` の下にある
- PATH変数
 - コマンドサーチパスが格納されている変数
 - 通常 “`.bash_profile`” で設定する
 - コロン (:) 区切りで複数のパスをつなげて指定
 - 内容は “`echo $PATH`” コマンドで確認できる
`echo`コマンドは、引数の内容を標準出力に出力
 - 同じ名前のコマンドが複数のパスに存在すると、最初のパスが優先
- エラー “Command not found”
 - プログラムが(正しく)インストールされていない。
 - コマンドサーチパスが正しく設定されていない。

コマンドサーチパスの設定

- 一時的に設定する場合
 - PATH変数にコマンドサーチパスを設定


```
PATH=$PATH:-/bin
```

 既存のPATH変数の後に “`-/bin`” を加える
- 恒常に設定する場合
 - ファイル `~/.bash_profile` に設定を書き込む


```
PATH=$PATH:-/bin
```

`.bash_profile`はログイン時に実行されるファイル

実習23

● コマンドサーチパス

(カレントディレクトリは ~/unixtest)

- 現在のコマンドサーチパスが格納されているPATH変数を確認する。

```
$ echo $PATH
```

- ~/binをコマンドサーチパスに一時的に追加する

```
$ PATH=$PATH:~/bin
```

ユーザは~/binディレクトリへ自分で作成したコマンドをコピーし、パスの記述なしで呼び出す事ができる。

- 前の実習で使用したtestpgを~/binディレクトリにコピーしてパス無しで動作するかを確認する。もし~/binディレクトリは存在しない場合はで作成すること。

```
$ cd
```

```
($ mkdir bin)
```

```
$ testpg # エラーになる
```

```
$ cp unixtest/testpg bin
```

```
$ testpg
```

文字列を標準出力へ出力 (echo)

● echo [-n] 文字列

- 画面に “Hello World”と表示する

```
$ echo 'Hello World'
```

- ファイル “.bash_profile” に文字列を書き込む

```
$ echo '#!/bin/sh' > .bash_profile
```

リダイレクトで標準出力をファイルに向ける

- 追記で文字列を書き込む

```
$ echo 'PATH=$PATH:~/bin' >> .bash_profile
```

既存のファイルへ追記する場合は “>>” を使用

実習24

● 恒常にコマンドサーチパスを追加

(カレントディレクトリは ~ ホームディレクトリへ移動)

1. ホームディレクトリに移動

```
$ cd
```

2. ログイン時に実行される**.bash_profile**へechoコマンドを使用して設定を書き込む。

```
$ echo 'PATH=$PATH:~/bin' > .bash_profile
```

3. **.bash_profile**の内容を確認する。

```
$ less .bash_profile
```

4. ターミナルで新しいウィンドウを表示してPATH変数を確認する。

Command + N で新規ウィンドウ

```
$ echo $PATH
```

エイリアス(alias)

● alias 別名='コマンド名'

● コマンドの別名を作成する

- lsの初期オプションを"-a"(隠しファイル表示)にする

```
$ alias ls='ls -a'
```

- rmの初期オプションを"-i"(確認あり)にする

```
$ alias rm='rm -i'
```

● エイリアスを取り消す

- ls のエイリアス設定をクリアする

```
$ unalias ls
```

- バックスラッシュ(¥)でエイリアスを一時的に取り消す

```
$ \$ls
```

● 設定ずみエイリアスの一覧表示

```
$ alias
```

実習25

● エイリアス

(カレントディレクトリは ~)

- エイリアスを使ってlsの動作を"隠しファイル表示 (-a)"且つ"ファイル種類表示 (-F)"に変更する。

```
alias ls='ls -aF'
```

- 現在設定されているエイリアスを確認する。

```
alias
```

- エイリアスではない本来のlsを実行した結果を比べてみる。

```
$ ls
```

```
$ $ls
```

- rmコマンドのオプション設定を確認あり (-i) にする。

```
$ alias rm='rm -i'
```

- unixtest/1433B_HUMAN.fastaを削除しエイリアスされたrmの動作を確認する。

```
$ rm unixtest/1433B_HUMAN.fasta
```

確認を求められるので、良ければ"y"と入力、それ以外の文字を入力するとキャンセルとなる。

メタキャラクタ

● シェルにとって特別な意味を持つ記号

例) * ? [] < > | ! \$; & 等

● 引数に現れる時は要注意

- スペース、メタキャラクタ
- 引数全体を引用符 (') で囲む
特別な意味を抑止する
- 下記例はリダイレクトによりファイルを上書きする

(誤) \$ grep ^> 1A01_HUMAN.fasta

(正) \$ grep '^>' 1A01_HUMAN.fasta

● ファイル名の付け方

- 英数字、ドット(.)、アンダーバー(_)、ハイフン(-)を使用
- 上記以外の記号、2バイト文字(日本語)は使用しない

実習26

● メタキャラクタ

(カレントディレクトリは ~/)

1. `grep ^> 1A31_HUMAN.fasta`の動作を確認する。

作業ディレクトリは~/unixtestとする。

```
$ cd unixtest
```

念のためコピーしたファイルを使用する。

```
$ cp Human/1A31_HUMAN.fasta .
```

```
$ grep ^> 1A31_HUMAN.fasta
```

2. 結果が戻ってこないので、適当な文字を入力し、`Control-D`で終了する。

このコマンドラインは、キーボードから入力された文字列から行頭 (^) を検索して `1A31_HUMAN.fasta` ファイルにリダイレクト出力 (>) するという意味になる。

ジョブの中止と中断

● コマンド（ジョブ）の実行中

- 端末は結果待ち状態
- 端末からの入力を受け付けない
- 実行が終了すると、プロンプトを表示

● 実行中のジョブを途中で中止

- コントロールキーを押しながら "C"

`Control + C`

● ジョブを一時中断

- コントロールキーを押しながら "Z"

`Control + Z`

● 一時中断したジョブを再開

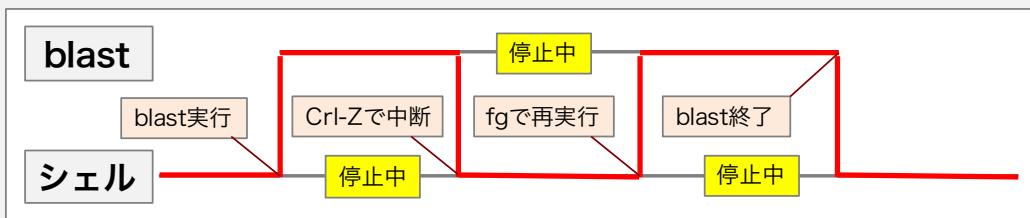
- `fg` : フォアグラウンドで再開 (端末は結果待ちの状態)
- `bg` : バックグラウンドで再開 (端末からの入力ができる)

はじめからバックグラウンドでジョブを実行するには、コマンド行末尾に "&"

フォアグラウンドとバックグラウンド

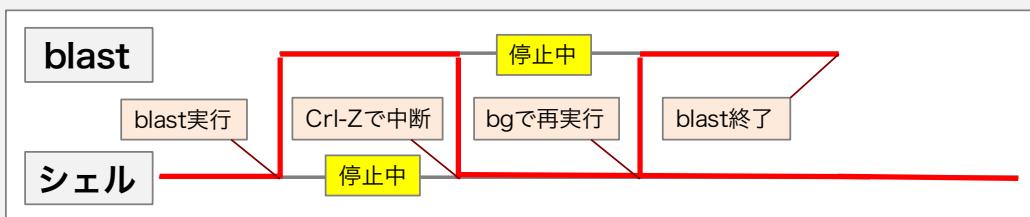
● フォアグラウンド実行 (fg)

- 実行中はコマンド入力を受け付けない



● バックグラウンド実行 (bg)

- 実行中でもコマンド入力が可能



プロセスの表示と停止(ps,kill)

● プロセス

- OSが管理する単位
- ジョブはシェルが管理する単位

● プロセスの一覧表示

- ps コマンド

PID: プロセスID, **TTY:** 端末

TIME: CPU時間, **CMD:** コマンド

\$ ps			
PID	TTY	TIME	CMD
218074	pts/12	00:00:00	bash
223974	pts/12	00:00:01	ls
223975	pts/12	00:00:00	ps

● 実行中のプロセス停止

- kill PID...
- PIDをpsで調べて停止

\$ kill 223974			
\$ ps			
PID	TTY	TIME	CMD
218074	pts/12	00:00:00	bash
223980	pts/12	00:00:00	ps
[1]+	Terminated	ls -lR /	

プロセスの稼働状況を把握(**top**)

● **top**

- UNIXマシンの稼働状況をリアルタイムに表示

```
$ top

Processes: 276 total, 3 running, 13 stuck, 260 sleeping, 1604 threads 15:41:31
Load Avg: 1.98, 2.24, 2.22 CPU usage: 3.46% user, 1.47% sys, 95.5% idle
SharedLibs: 1604K resident, 0B data, 0B linkedit.
MemRegions: 183073 total, 8815M resident, 143M private, 3238M shared.
PhysMem: 15G used (2148M wired), 6234M unused.
VM: 711G vsize, 1026M framework vsize, 18564(0) swapins, 72355(0) swapouts.
Networks: packets: 69788526/45G in, 43784897/21G out.
Disks: 14135478/253G read, 30377589/763G written.

PID      COMMAND      %CPU      TIME      #TH      #WQ      #PORTS      #MREGS      MEM      RPRVT
95057    installd    0.0      00:11.75 2      0      53      161      12M      12M
95056    suhelperd   0.0      00:05.01 2      0      58      164      5840K     5448K
95054    softwareupda 0.0      01:45.06 11     0      175      372      96M      95M
87334    plugin-conta 0.0      00:30.01 6      1      230      182      3344K     2748K
87088    mdworker     0.0      00:00.07 4      0      54      80       6264K     5392Ks
```

- 終了する時は“q”を押下

実習27

● プロセスの確認

(カレントディレクトリは `~/unixtest`)

1. **ps**コマンドを使用して、プロセス一覧を表示する。

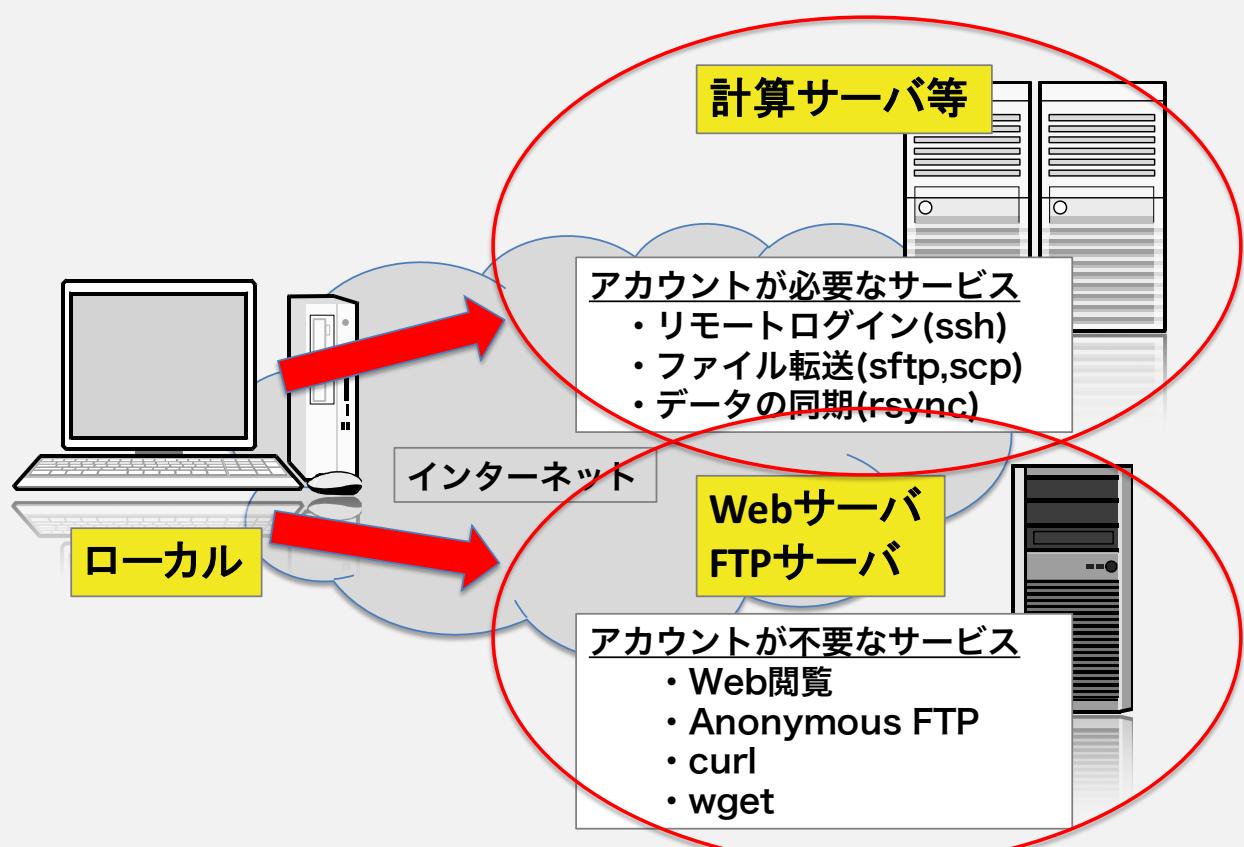
```
$ ps
```

2. **top**コマンドを使用して、プロセスの稼働状況を確認する。

```
$ top
```

ネットワークを経由した利用

ネットワークを介したサービス



リモートログイン(ssh)

● ssh ユーザ名@ホスト名

- または `ssh -l ユーザ名 ホスト名`
- ネットワーク経由で別の計算機にログイン

```
$ ssh guest@bias4.nibb.ac.jp
Are you sure you want to continue connecting (yes/no)? yes
guest@nibb.ac.jp's password: your_password
[guest@bias4 ~]$
[guest@bias4 ~]$ exit    (ログアウトして元のセッションに戻る)
$
```

● ssh -X ユーザ名@ホスト名

- X11対応ソフトをローカルの画面に表示可能とする
リモート上で作成した画像ファイルの表示などに使用
ローカルの環境にx11ソフトが必要
`osxはxquarts, Windowsはcygwin/x`

リモートファイルコピー(scp)

● scp コピー元 コピー先

- ネットワーク経由でファイルのコピーを行う
`$ scp text.txt user@bias4.nibb.ac.jp:data/text.txt`
- ネットワーク上のコピー先指定方法
`user@host:<where_to_copy_path>`
 - * コピー元、コピー先のいずれかに指定する
 - 例) `nibb@bias4.nibb.ac.jp:data/sprot/1433S_HUMAN.sprot`
 - * ディレクトリ丸ごとコピーする場合 `-r` オプション
 - * `<where_to_copy_path>`を指定しない場合はホームディレクトリ直下にコピー

● cpコマンドと同じ感覚で使用できる

ファイル転送(sftp/ftp)

● sftp リモートホスト名

- sftp/ftpログイン後、専用コマンドを使用して操作
bias4へはsftpのみ使用可能。**bias4**からは両方使用可。

コマンド	用途
ls	リモートのファイル一覧表示
lls	ローカルのファイル一覧表示
cd	リモートのディレクトリ移動
lcd	ローカルのディレクトリ移動
get	ファイルをリモートからローカルに転送
mget	複数のファイルを転送。ワイルドカード使用可
put	ファイルをローカルからリモートに転送
mput	複数のファイルを転送。ワイルドカード使用可
help	使用できるコマンドの簡易ヘルプ

データ転送(curl)

● curl URL

- HTTPやFTP経由のファイル取得

```
$ curl http://www.nibb.ac.jp
```

www.nibb.ac.jpのトップページを標準出力に書き出し

オプション	用途
-o file url	urlのデータをfileに保存
-o	url上のファイル名で保存 (http://hoge.com/fig.jpgのように「ファイル名がある」urlのみ)
-o url[start-end]	http://www.hoge.com/[01-10].jpg等とすることで、01.jpg, 02.jpg, 03.jpg ... 10.jpgのファイル全て取得
-h	ヘルプを表示

データの同期(rsync)

- rsync [オプション] コピー元 / コピー先 /
– ディレクトリ間の同期をとる

```
$ rsync -auv user@bias4:dir/ ./dir/
```

オプション	用途
-a	アーカイブモード（属性などを一致させるなど）
-u	コピー先の同一ファイルが新しい場合コピーしない
-v	コピー状況の表示
--delete	コピー元にないファイルをコピー先から削除
--exclude-file filename	filename内のファイル名リストは対象にしない

ローカル内やリモート間のファイル群コピーの他、手軽なバックアップの手段としてよく用いられる。

実習28

- ディレクトリの同期

（カレントディレクトリは ~/unixtest）

1. 作業用ディレクトリ unixtest の同期を ~/backup/unixtest ディレクトリに行う。 backup ディレクトリは作成すること。

```
$ cd  
$ mkdir backup  
$ rsync -auv unixtest/ backup/unixtest/
```

2. 同期元から unixtest/1433B_HUMAN.sprot を削除

```
$ rm unixtest/1433B_HUMAN.sprot
```

3. 再度 rsync で同期を行い同期元から削除されたファイルがどうなるか確認する。オプションとして --delete を使用する。

```
$ rsync -auv --delete unixtest/ backup/unixtest/
```

ファイルの圧縮と解凍

● データ圧縮

- 圧縮アルゴリズムを使い、情報を失わずにサイズを小さくする
- 圧縮ファイルは使用する前に元に戻す（解凍）

● gzip (拡張子 : .gz or .Z)

圧縮 : `gzip` ファイル名 (ファイル名.gzを作成)

解凍 : `gunzip` ファイル名.gz または `gzip -d` ファイル名.gz

● bzip2 (拡張子 : .gz2)

圧縮 : `bzip2` ファイル名

解凍 : `bunzip2` ファイル名.bz2 または `bzip2 -d` ファイル名.bz2

アーカイブの作成と展開(tar)

● アーカイブ

- 複数のファイルやディレクトリを1つのファイルにまとめること
- 圧縮と組み合わせて、データ配布やバックアップ保存などに用いる
- 通常アーカイブファイルは拡張子.tarをつける

● 基本的な使用方法

- dirをarchive.tarとしてアーカイブ

```
$ tar cvf archive.tar dir
  c: 新しいアーカイブを作成
  v: 処理したファイルの一覧を出力
  f file: fileというアーカイブ・ファイルを使う
```

- archive.tarをカレントディレクトリに展開

```
$ tar xvf archive.tar
  x: アーカイブからファイルを抽出
```

- 圧縮・解凍の同時実行

```
$ tar zxvf archive.tar.gz
  z: gzipで圧縮・解凍の処理を行う、j: bzipで圧縮・解凍の処理を行う
```

実習29

●圧縮アーカイブの作成と展開

(カレントディレクトリは ~)

- 先ほどの~/backup/unixtestディレクトリの圧縮アーカイブを、~/backup/unixtest.tar.gzという名前で作成する。圧縮形式はgzip形式を使用し、作成後は元のディレクトリを削除する。

```
$ cd backup
$ tar zcvf unixtest.tar.gz ./unixtest
$ rm -rf ./unixtest
```

- unixtest.tar.gzの内容を確認する。

```
$ tar ztvf unixtest.tar.gz
```

- ~/backup/unixtest.tar.gzを~/backupディレクトリ内に解凍展開する。

```
$ tar zxvf unixtest.tar.gz
```

プログラムソースからのインストール

● ソース一式の取得

- Anonymous FTPやWebからアーカイブをダウンロード
- ftp や curl を使用

● アーカイブを展開

- tar や gzip を使用

● コンパイル

- インストール方法や注意点を確認
内包するREADME, INSTALLなどのファイルを読む
必ずしも下記のとおりと限らないので、記述されているインストール手順に従うこと
- 一般的なコンパイル手順

\$ configure	環境にあったコンパイルの設定を行う
--------------	-------------------

\$ make	プログラムをコンパイル
---------	-------------

\$ make install	所定の場所へインストール
-----------------	--------------

- インストール先の書込権限がない場合、root権限で行う必要あり
\$ sudo make install

実習30

● SAMtools 1.3のインストール

(カレントディレクトリは ~)

1. SAMtoolsのダウンロードと解凍

<http://www.htslib.org> (Download->samtools-1.3)

~/Downloads/にbzip圧縮アーカイブとして保存される

```
$ mv Downloads/samtools-1.3.tar.bz2 unixtest
```

```
$ cd unixtest
```

```
$ tar jxvf samtools-1.3.tar.bz2
```

```
$ cd samtools-1.3
```

2. コンパイルとインストール

ここでは自分のホームディレクトリ以下にインストール

```
$ make
```

```
$ make prefix=~ install
```

makeコマンドの-nオプションで実際の実行を行わずどのように処理がされるか表示できる。makeコマンドを実行する前に確認してみよう。

お疲れさまでした

この講習で使用したコマンド一覧は
資料末尾に添付しましたので
参考にしてください。

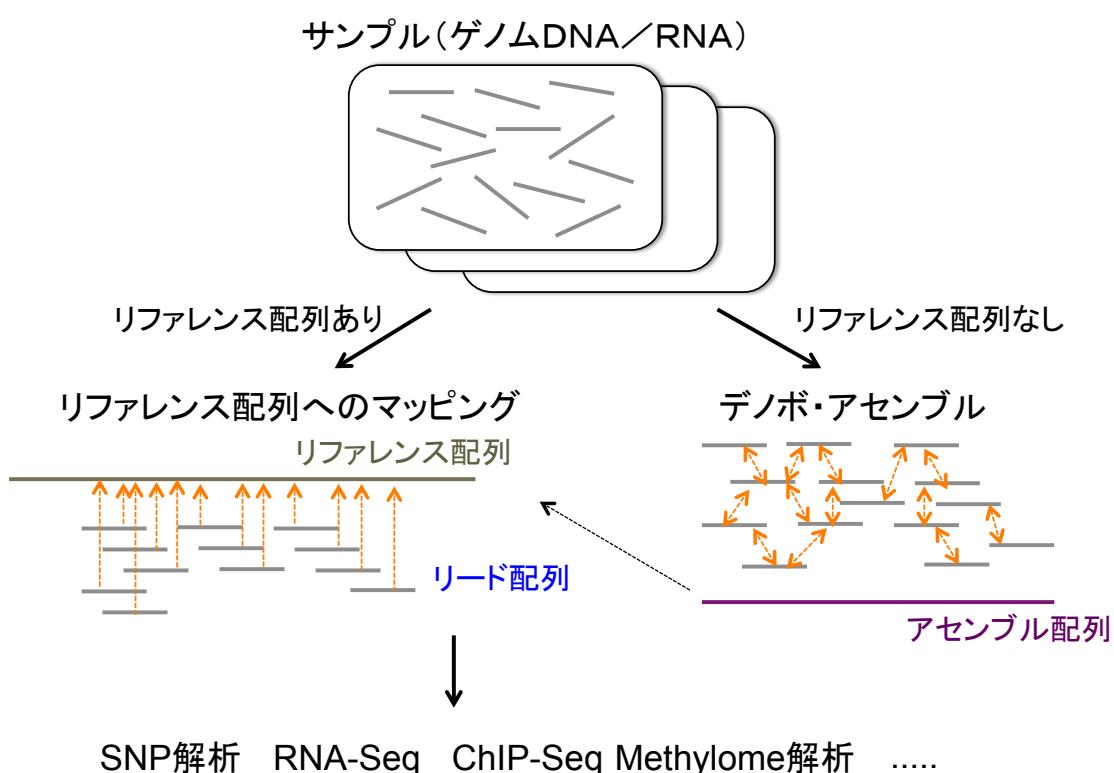
次世代シーケンサ用 データ解析コマンド

基礎生物学研究所

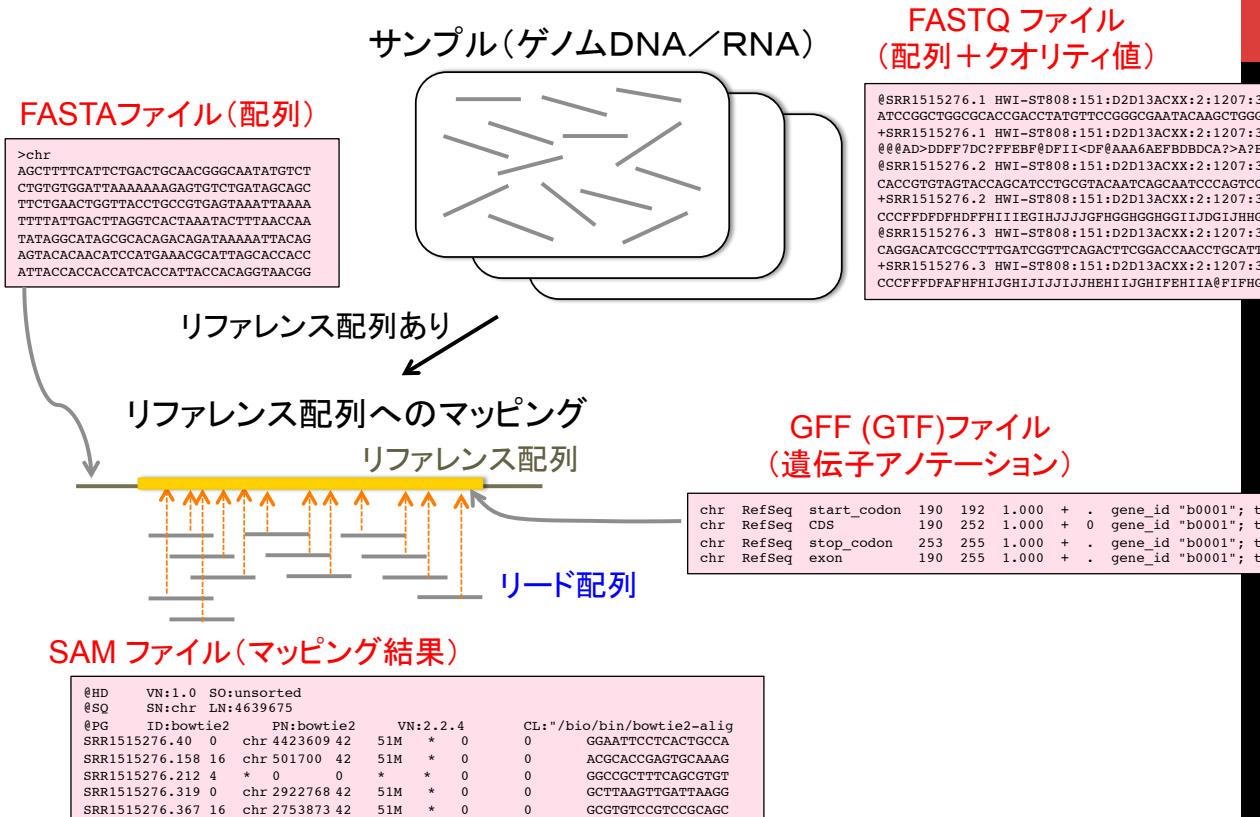
ゲノムインフォマティクストレーニングコース

内山 郁夫 (uchiyama@nibb.ac.jp)

次世代シーケンサデータ処理の概要



次世代シーケンサデータ処理の概要



配列データファイル: **FASTA format**

```
>NM_174292.2 Bos taurus crystallin gamma S (CRYGS) mRNA
TGCACCAAACATGTCTAAAGCTGGAACCAAAATTACTTTCTTGAAAGACAAAAAC
TTTCAAGGCCGCCACTATGACAGCGATTGCGACTGTGCAGATTCCACATGTACC
TGAGCCGCTGCAACTCCATCAGAGTGGAAAGGAGGCACCTGGGCTGTGTATGAAAG
GCCCAATTTCGCTGG
>NM_174294.1 Bos taurus casein kappa (CSN3) mRNA
TTCACTTACAGTGGAAAGGCCAACTGAACCTACTGCCAAGCAAGAGCTGACGGTC
ACAAGGAAAGGTGCAATGATGAAGAGTTTTCTAGTTGTGACTATCCTGGCAT
TAACCCCTGCCATTTCGGGTGCCAGGAGCAAAACCAAGAACCAATAACGCTG
TGAGAAAGATGAAAG
```

<- >配列ID(説明)
<- 塩基配列

- 配列ファイルの標準フォーマット
- >で始まる行がタイトル行、その後に配列が続く
- タイトル行の最初の単語が配列ID、以降は説明(省略可)
- タイトル行の長さに制限はないが、途中に改行は入らない
- 配列は途中に改行が入ってもよい

配列データファイル: FASTQ format

```

@ERR004063.1 IL33_2678:6:1:0:902/2          ← @配列ID (説明)
CTGTAAATATGTACAGGATATTCTGACCATTCTTC          ← 塩基配列
+                                                 ← +
CCCDDCC8@DD8DDCC5?,D7??&=-4&9*&&+-.'      ← クオリティ値

@ERR004063.2 IL33_2678:6:1:0:1059/2
AAATGGAGACTTATTCTTGCTTTGGTAATCAATC
+
@@@=@;>CC;<;DD>>7AA>88>DE>AC96@;&<C7>
@ERR004063.3 IL33_2678:6:1:0:1320/2
AGCATTGGAGTGGCTTTTTTTGTTTTTTAAA
+
07ECEEE=CDCA<AC108D@0(*4(,:0;6*0*(4(((

```

- 配列とクオリティ値をひとつにまとめたもの
- 各データの先頭は「@」、塩基配列とクオリティ値配列の間に「+」
- クオリティ値は「アスキーコード」に従って文字列で表示される
→数字を使う場合と比べてサイズが圧縮され、高速な処理が可能
- 塩基配列、クオリティ値配列ともに1行で書くのが基本
- @や+で始まる行がタイトル行やクオリティデータ開始行であるとは限らない(@や+はクオリティ値の中にも現れるので)

クオリティ値

- PHRED Quality: エラー率 p_e に対して、以下で定義される

$$Q_{PHRED} = -10 \log_{10}(p_e)$$

例) $Q_{PHRED}=30$ の場合、エラー率は 10^{-3}

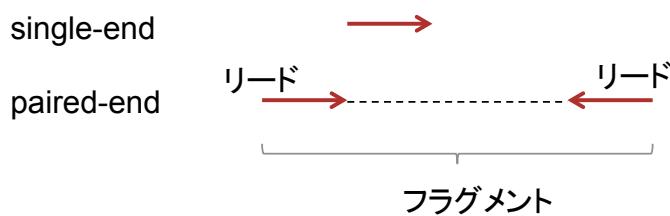
- $Q_{PHRED}+33$ の値が、以下のアスキーコード表に従って文字列として表示される

	30	40	50	60	70	80	90	100	110	120
0:	(2	<	F	P	Z	d	n	x	
1:)	3	=	G	Q	[e	o	y	
2:	(SP)	*	4	>	H	R	\	f	p	z
3:	!	+	5	?	I	S]	g	q	{
4:	"	,	6	@	J	T	^	h	r	
5:	#	-	7	A	K	U	_	i	s	}
6:	\$.	8	B	L	V	'	j	t	~
7:	%	/	9	C	M	W	a	k	u	(DEL)
8:	&	0	:	D	N	X	b	l	v	
9:	'	1	;	E	O	Y	c	m	w	

リファレンス配列へのマッピング

Bowtie, BWA, SOAPなどのコマンド

- 長大なリファレンス配列に大量の短いリード配列を若干のミスマッチを許して照合する
- リファレンス配列に対して、あらかじめ全文検索インデックスを作成することにより高速に検索を行う
- paired-end read に対応。insert sizeに制約をつけられる



Bowtie コマンド

- BowtieとBowtie2がある。後者はギャップを考慮した検索を行うので、感度がより高い

- インデックスの作成

```
bowtie2-build 配列ファイル インデックス名
```

- マッピングの実行

(single-end read の場合)

```
bowtie2 -x インデックス名  
-u リードファイル -s 出力ファイル
```

(paired-end read の場合)

```
bowtie2 -x インデックス名  
-1 リード1 -2 リード2 -s 出力ファイル
```

(改行せずに1行で打つ)

実習

データ: 大腸菌RNA-Seqデータ(GEO:SRP044366)の一部を抜粋したもの

ディレクトリ: `~/data/2_ngs`

リードファイル: `test_fastq/ecoli.[1-12].fastq`

大腸菌ゲノム配列: `ecoli_genome.fa`

大腸菌遺伝子テーブル: `ecoli.gtf`

- 作業ディレクトリに移動

```
$ cd ~/data/2_ngs
```

Bowtie実習

- リファレンス配列(`ecoli_genome.fa`)に対してインデックスを作成する

```
$ bowtie2-build ecoli_genome.fa ecoli_genome
```

- リードファイル `test_fastq/ecoli.1.fastq` をリファレンス配列上にマッピングする

```
$ bowtie2 -x ecoli_genome  
        -U test_fastq/ecoli.1.fastq  
        -S ecoli.sam
```

(改行せずに1行で打つ)

マッピング結果ファイル: SAM format

- リファレンス配列に多数の配列をマップした結果を表す形式
- 最初の@付きの行はヘッダ行、以降はタブ区切りで記述されたマッピング結果のデータ

@HD VN:1.3 SO:coordinate										
@SQ SN:ref LN:45										
r001	163	chr1	7	30	8M2I4M1D3M	=	37	39	TTAGATAAAGGATACTG	*
r002	0	chr1	9	30	3S6M1P1I4M	*	0	0	AAAAGATAAGGATA	*
r003	0	chr1	9	30	5H6M	*	0	0	AGCTAA	* NM:i:1
r004	0	chr1	16	30	6M14N5M	*	0	0	ATAGCTTCAGC	*
r003	16	chr1	29	30	6H5M	*	0	0	TAGGC	* NM:i:0
r001	83	chr1	37	30	M	=	7	-39	CAGGCCAT	*

テンプレート名 フラグ マップ結果 アライメント (CIGAR) 対となるリードの位置情報 リードの配列 オプション

リファレンス配列 chr1 12345678901234 5678901234567890123456789012345
ACGATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

マッピング結果ファイル: SAM format

ヘッダ行 @HD VN: バージョン番号 SO:並び順 @SQ SN: リファレンス配列名 LN:リファレンス配列長 @RG リードグループの情報、@PG プログラムの情報										
@HD VN:1.3 SO:coordinate @SQ SN:ref LN:45										
r001	163	chr1	7	30	8M2I4M1D3M	=	37	39	TTAGATAAAGGATACTG	*
r002	0	chr1	9	30	3S6M1P1I4M	*	0	0	AAAAGATAAGGATA	*
r003	0	chr1	9	30	5H6M	*	0	0	AGCTAA	* NM:i:1
r004	0	chr1	16	30	6M14N5M	*	0	0	ATAGCTTCAGC	*
r003	16	chr1	29	30	6H5M	*	0	0	TAGGC	* NM:i:0
r001	83	chr1	37	30	M	=	7	-39	CAGGCCAT	*

フラグ フラグ マップ結果 アライメント (CIGAR) 対となるリードの位置情報 リードの配列 オプション

SAM形式各カラムの情報

- 1. QNAME: クエリー配列名
- 2. FLAG: 各種情報を保持したフラグ
- 3. RNAME: リファレンス配列名
- 4. POS: リードがマッピングされた左端位置
- 5. MAPQ: マッピング クオリティ- $10\log_{10} p_e$
- 6. CIGAR: CIGAR 文字列(アライメント)
- 7. RNEXT: ペアの相方が載ったリファレンス配列名 (=は同一配列、*は情報なし)
- 8. PNEXT: ペアの相方のマップ位置
- 9. TLEN: フラグメントの長さ
- 10. SEQ: リードの配列
- 11. QUAL: リードのクオリティ(アスキー)
- 12. OPT: tag:type:value で表す任意の情報

BAM format

- SAM形式のテキストファイルをバイナリファイルに変換し、かつデータ圧縮をかけてコンパクトかつ効率的に処理できるようにしたもの。
- **samtools**を使って変換する
 - SAM→BAM変換

```
samtools view -Sb file.sam > file.bam
    • -S 入力がSAMファイル; -b 出力がBAMファイル
```
 - BAM→SAM変換

```
samtools view -h file.bam > file.sam
    • -h 出力にヘッダを含める
```

samtools

- SAM/BAM形式のマッピング結果ファイルを扱うためのコマンド群

```
samtools <command> [options]
```

2番目の引数の<command>によって機能を切り替える

- view SAM↔BAM 変換して表示
- sort 並べかえ(デフォルトはマップされた位置で)
- index ソートされたBAMファイルをインデックスづけ
- idxstats 各リファレンス配列にマップされたリード数を表示
- depth リファレンスの各位置にマップされたリード数を表示
- mpileup リファレンスの各位置にマップされた塩基を表示

samtools 実習1

- BAMファイルの作成

```
$ samtools view -bS ecoli.sam > ecoli.bam
```

- BAMをSAMに変換して表示

```
$ samtools view ecoli.bam | less
```

samtools 実習2

BAMファイルは、ソートしてインデックスづけを行うことによって、より便利に使えるようになる。

- BAMファイルをソート

```
$ samtools sort ecoli.bam ecoli_sorted
```

ソート後のファイル名は、最後の引数に.bamが付加されたものになる

- ソートされたBAMファイルをSAMに変換してlessで表示

```
$ samtools view ecoli_sorted.bam | less
```

- ソートされたBAMファイルに対してインデックスを作成

```
$ samtools index ecoli_sorted.bam
```

samtools 実習3

以下は、インデックスづけされたBAMファイルを使う

- 指定した領域内にマッピングされたリードのみを表示

```
$ samtools view ecoli_sorted.bam chr:200-500
```

染色体名 : 開始位置一終了位置

- 各染色体にマッピングされたリード数を表示

```
$ samtools idxstats ecoli_sorted.bam
```

マッピングされなかったリードは、染色体名が "*"として表示される

- mpileupコマンドで、リファレンスの位置ごとにマッピングされた塩基を表示

```
$ samtools mpileup -f ecoli_genome.fa ecoli_sorted.bam | less
```

RNA-Seq 解析

- ゲノム上にマッピングされたリードを遺伝子領域ごとに集めて数をカウント



通常、カウントした数を遺伝子の長さ、およびマップされたリード全体の数で割って標準化する

RPKM (Read Per Kilobase per Million mapped reads) または
FPKM (Fragment Per Kilobase per Million mapped reads)

遺伝子アノテーションファイル: **GFF format**

- ゲノム上の特徴配列(Feature segment)をタブ区切りテキスト形式で表現する標準的なフォーマット
- 最後のカラムに任意の情報を格納できるため、様々な特徴配列の情報を記述できるが、とくに遺伝子構造を記述するために特化した形式を **GTF format** という

1	2	3	4	5	6	7	8	9
chr	RefSeq	start_codon	190	192	1.000	+	.	gene_id "b0001"; transcript_id "b0001";
chr	RefSeq	CDS	190	252	1.000	+	0	gene_id "b0001"; transcript_id "b0001";
chr	RefSeq	stop_codon	253	255	1.000	+	.	gene_id "b0001"; transcript_id "b0001";
chr	RefSeq	exon	190	255	1.000	+	.	gene_id "b0001"; transcript_id "b0001";

1. Seqname 配列名
2. Source 予測プログラム、データベース名など
3. Feature 特徴セグメントの種類
4. Start開始位置
5. End 終了位置
6. Score スコア、省略可(.)
7. Strand ストランド(+/-)、省略可(.)
8. Frame 読み枠(0/1/2)、省略可(.)
9. Attribute (Optional)
セミコロンで区切られたタグ-値の対

マッピング結果を領域ごとに 集計するコマンド: **htseq-count**

- **htseq-count** マッピングファイル(SAM) 遺伝子ファイル(GFF)

3つの照合モード: union, intersection Strict, intersection Nonempty



<http://www-huber.embl.de/users/anders/HTSeq/doc/count.html>

htseq-countを用いた実習

- アノテーションテーブルecoli.gtf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

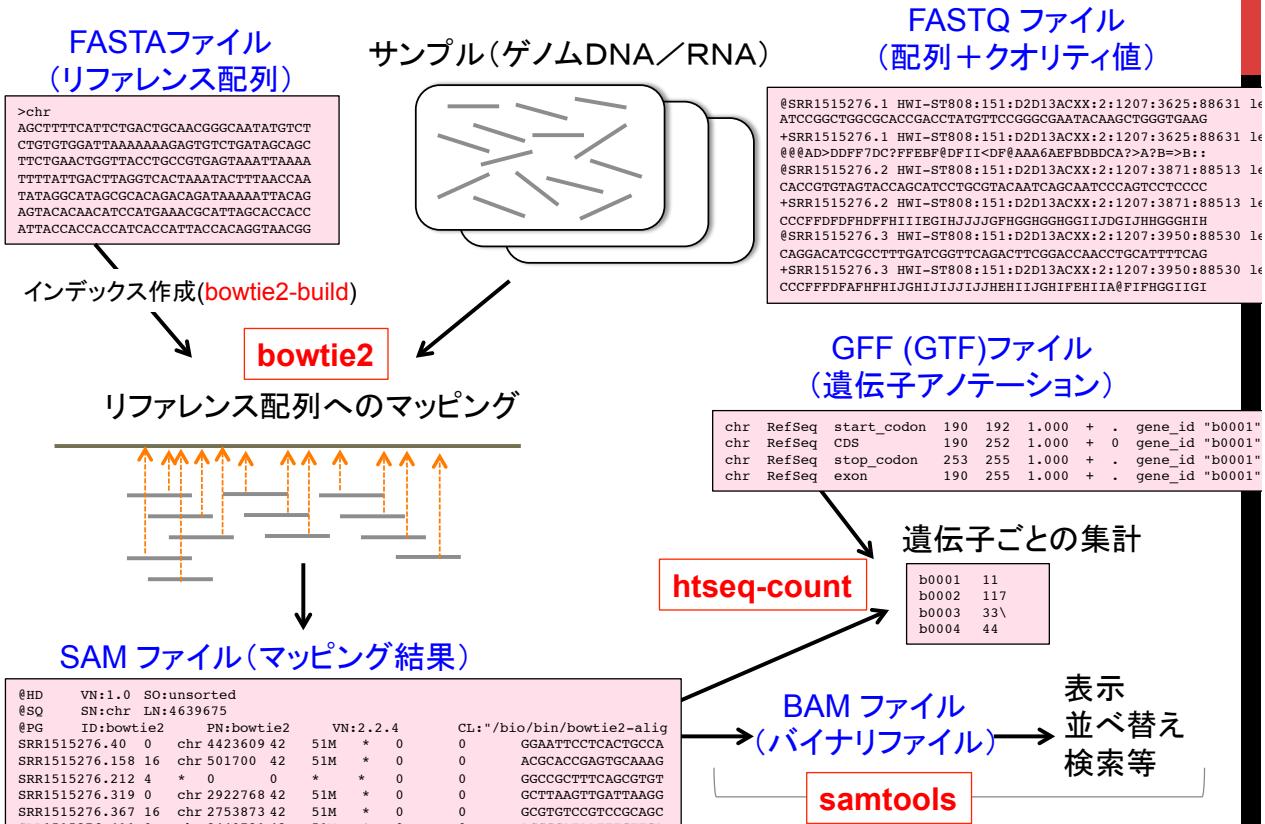
```
$ htseq-count ecoli.sam ecoli.gtf > ecoli.htseq
```

- 遺伝子アノテーション(GFF)ファイルの中で、どのFeatureの行を使うかはオプション -t で、遺伝子IDとして何を使うかについては -i で指定する。
- 標準的なGTF形式のファイルであれば、デフォルトのまま(Featureがexon の行を使い、遺伝子IDは gene_id を使う)で動作するようになっている。

出力結果

b0001	11
b0002	117
b0003	33
b0004	44
b0005	3
b0006	14
b0007	4
b0008	181

今回使ったツールとファイルのまとめ



R入門

基礎生物学研究所

ゲノムインフォマティクストレーニングコース

内山 郁夫 (uchiyama@nibb.ac.jp)

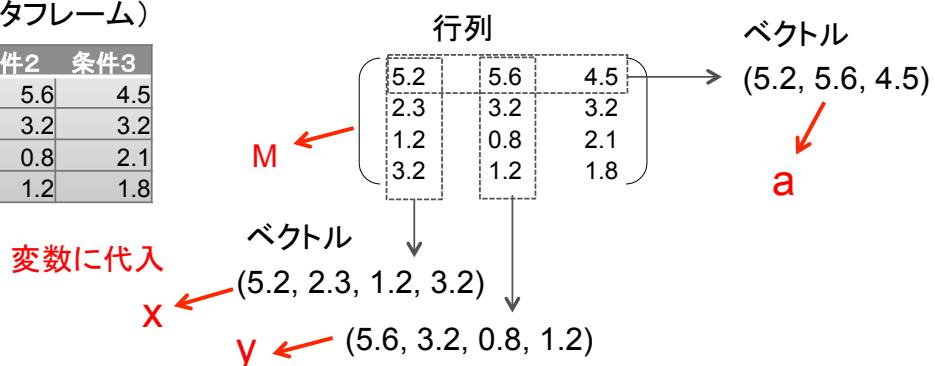
Rとは

- ベル研究所で開発された統計処理言語「S」を基に、フリーソフトとして開発された統計解析環境・プログラミング言語
- コマンドラインインターフェイスが基本。対話的なコマンド実行による解析のほか、プログラム(スクリプト)を書いて一括処理を行うことも可能
- ベクトル・行列演算が簡便かつ効率的に行える
- 独自の関数を作成することによって機能拡張が可能
- 作成した関数等をパッケージ単位でまとめることにより、機能拡張が容易に行える。様々なパッケージが公開されており、これらを導入することによって、最先端の統計手法を用いることができる。

Rにおけるデータ処理の概要

表データ（データフレーム）

	条件1	条件2	条件3
遺伝子1	5.2	5.6	4.5
遺伝子2	2.3	3.2	3.2
遺伝子3	1.2	0.8	2.1
遺伝子4	3.2	1.2	1.8

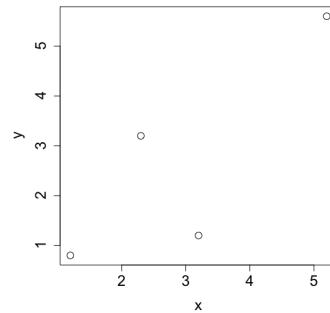


ベクトル演算

$$x / y \quad \begin{pmatrix} 5.2 & 2.3 & 1.2 & 3.2 \\ 5.6 & 3.2 & 0.8 & 1.2 \end{pmatrix}$$

プロットの作成

`plot(x, y)`



作業ディレクトリの設定

- 作業ディレクトリ：読み込むデータファイルや結果を書き出すファイルを保存するディレクトリ。
- メニューから、その他→作業ディレクトリの変更をえらんでディレクトリを選択する。

作業ディレクトリ `~/data/3_R` に移動

- 作業ディレクトリを常に固定したい場合は、環境設定から起動タブを開いて初期ディレクトリを設定する。
- 作業ディレクトリをコマンドで設定することも可能
`setwd("ディレクトリ名")` 作業ディレクトリの設定
`getwd()` 作業ディレクトリの確認

スカラー演算

```
> 1+3
```

```
[1] 4
```

```
> 1+3*5
```

通常の通り、*(かけ算)は
+(足し算)より優先する

```
[1] 16
```

```
> a <- 1+3*5
```

結果を変数 a に代入する

```
> a
```

a とだけタイプすると、変
数aの内容が表示される

```
[1] 16
```

```
> a > 10
```

論理演算は、論理値
TRUEまたはFALSEを返
す

```
[1] TRUE
```

変数と代入

● 計算結果を変数に代入することにより、再利用できる

- 変数名にはアルファベット、数字、「(dot)、
'_(underscore)が利用できる。ただし、先頭はアルファベット。
- 大文字と小文字は区別される。

● 代入を表す演算子には、<- = -> の3通りがある。

以下の3つはいずれも a に 4 が代入される。

```
> a <- 1 + 3
```

```
> a = 1 + 3
```

```
> 1 + 3 -> a
```

ヒストリー(履歴)

- コンソール上で、上下の矢印キー($\uparrow\downarrow$)によって、コマンドの履歴を前後にたどることができる。
- 左右の矢印キー($\leftarrow\rightarrow$)によって、カーソルを左右に動かせる。これによってコマンドの編集ができる。
- 以下のコントロールキーを使った操作も可能
 - Control+P 履歴を前に移動(\uparrow と同じ)
 - Control+N 履歴を後ろに移動(\downarrow と同じ)
 - Control+B カーソルを左に移動(\leftarrow と同じ)
 - Control+F カーソルを右に移動(\rightarrow と同じ)
 - Control+A カーソルを行の先頭に移動
 - Control+E カーソルを行の最後に移動
- GUIからヒストリーパネルを使って履歴をたどることも可能



ベクトル

```
> a <- c(1, 3, 7, 4, 6) ベクトルは関数 c を用いて  
    > a 作成する  
    [1] 1 3 7 4 6  
    > b <- 3:7 3から7までの連続した整数  
    > b  
    [1] 3 4 5 6 7  
    > length(a) ベクトルaの長さ(要素数)  
    [1] 5  
    > b[3] bの3番目の要素  
    [1] 5
```

ベクトル演算

a=c(1,3,7,4,6); b=c(3,4,5,6,7) と設定されている

```
> 1 + a          aの各要素に1を加える  
[1] 2 4 8 5 7  
  
> 2 * a          aの各要素を2倍する  
[1] 2 6 14 8 12  
  
> a + b          aとbの要素ごとの和をとる  
[1] 4 7 12 10 13  
  
> a * b          aとbの要素ごとの積をとる  
[1] 3 12 35 24 42  
  
> a > 3          aの要素ごとに3より大きいか比較する  
[1] FALSE FALSE TRUE TRUE TRUE  
  
> a < b          aとbを要素ごとに大小を比較する  
[1] TRUE TRUE FALSE TRUE TRUE
```

基本統計量の計算

a=c(1,3,7,4,6)と設定されている

```
> length(a)      aの長さ(要素数)  
> sum(a)         aの要素の合計値  
> mean(a)        aの要素の平均値  
> median(a)      aの要素の中央値  
> var(a)         aの要素の不偏分散  
> sd(a)          aの要素の標準偏差
```

問題) mean(a)をsum(a)とlength(a)を使って計算してみよう。

ベクトル要素の抽出

`a=c(1,3,7,4,6)`と設定されている

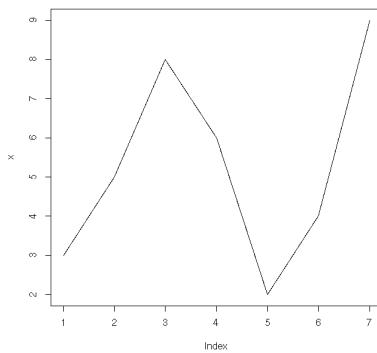
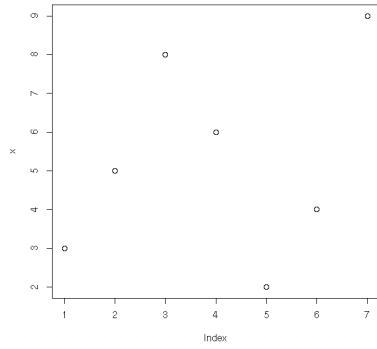
```
> a[2:4]          # 2番目から4番目までの要素  
[1] 3 7 4  
  
> a[c(3,5,2)]    # 3, 5, 2番目の要素  
[1] 7 6 3  
  
> a[c(T,T,F,F,T)] # T(TRUE)である要素だけ出力  
[1] 1 3 6  
  
> a[a > 3]        # a>3がTRUEである要素だけ出力  
[1] 7 4 6
```

並べかえ(ソート)

```
> x <- c(3,5,8,6,2,4,9)  
  
> sort(x)          # 小さい順に並べかえ  
[1] 2 3 4 5 6 8 9  
  
> sort(x, decreasing=TRUE) # 大きい順  
[1] 9 8 6 5 4 3 2
```

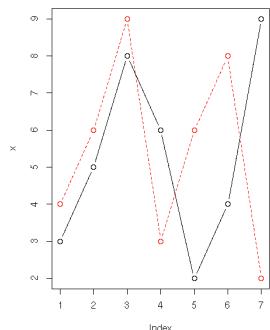
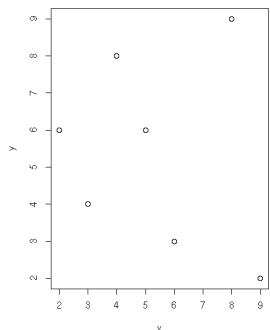
プロットの作成(1)

```
> x  
[1] 3 5 8 6 2 4 9  
> plot(x)  
> plot(x, type="l")  
> barplot(x)
```



プロットの作成(2)

```
# x <- c(3,5,8,6,2,4,9)と設定されている  
> y <- c(4,6,9,3,6,8,2)  
> par(mfrow=c(1,2)) # 2つのプロットを表示  
# x,y をx軸、y軸にとって散布図としてプロット  
> plot(x,y)  
# x,yを別々にプロット。まずxをプロット。点と線を両方描く。  
> plot(x, type="b")  
# yを重ねてプロット。linesは枠を描き直さずに線だけを引く。  
# lty はline type, col は colorを指定。  
> lines(y, type="b", lty=2, col=2)
```



Rにおける基本データ型

- Rの「原子データ型」として以下のものがある
 - 数値型 numeric(実数または整数)
 - 論理型 logical(TRUE/FALSE)
 - 文字列型 character("abc", "123" のように、二重引用符で囲まれた文字列として表す)
- Rの原子データはベクトルである。スカラー値も長さ1のベクトルである。すなわち、ベクトルは型を持ち、すべての要素は同じ型のデータからなる。
- 異なる型のデータを集めた構造として「リスト」がある。
- mode(x) によって、変数 x の型を調べられる

演算子

● 算術演算子

+ (加算) - (減算) * (乗算) / (除算),
%/%(整数除算) %% (剰余) ^ (累乗)
例) 5 / 2 (=2.5) 5 % / % 2 (=2) 5 %% 2 (=1)

● 論理演算子

& (論理積「かつ」) | (論理和「または」) ! a (aの否定)

● 比較演算子

>, >=, <, <= (不等号) == (等しい) != (等しくない)

これらの演算子はベクトルの要素ごとにはたらく

(例) a=c(1,3,7,4,6)と設定されている

> a >= 2 & a < 5

[1] FALSE TRUE FALSE TRUE FALSE

ワークスペースとオブジェクト

- コンソール上で `ls()` または `objects()` とすると、変数に保存されたデータ(オブジェクト)のリストを参照できる。

```
>ls()
```

```
[1] "a" "b" "x"
```

- ワークスペースブラウザ(メニューバー「ワークスペース」から起動)でより詳細な情報を閲覧可能
- `rm(変数名)` で、オブジェクトを消去できる。

関数

関数名(引数1, 引数2, ...)

- 関数は、一般に複数の引数を入力としてとり、何らかの計算を行って一つのオブジェクトを返す(戻り値)。
- 引数には、必須のものと省略可能なものがある。後者は省略すると「デフォルト値」が使用される。
- 引数は、順番によって指定する方法と、「名前=値」の形式で指定する方法がある。通例、必須の引数は前者、選択可能な引数は後者で指定する。

例1) ベクトル `x` を小さい順(increasing order)でソートする

```
sort(x)
```

例2) ベクトル `x` を大きい順 (decreasing order)でソートする

```
sort(x, decreasing=TRUE)
```

マニュアルの表示

- `help(関数名)` または `?関数名` でマニュアルを表示する

> `help(median)`

```
median      package:stats    R Documentation
Median Value
Description:
  Compute the sample median.
Usage:
  median(x, na.rm = FALSE)
Arguments:
  x: an object for which a method has been defined, or a numeric
     vector containing the values whose median is to be computed.
  na.rm: a logical value indicating whether 'NA' values should be
     stripped before the computation proceeds.
Details:
  This is a generic function for which methods can be written.
  However, the default method makes use of 'sort' and 'mean' from
  package 'base' both of which are generic, and so the default
  method will work for most classes (e.g. "Date") for which a
  median is a reasonable concept.
Value:
  The default method returns a length-one object of the same type as
  'x', except when 'x' is integer of even length, when the result
  will be double.
  If there are no values or if 'na.rm = FALSE' and there are 'NA'
  values the result is 'NA' of the same type as 'x' (or more
  generally the result of x[FALSE][NA]).
References:
  Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
  Language_, Wadsworth & Brooks/Cole.
See Also:
  'quantile' for general quantiles.
Examples:
  median(1:4) #= 2.5 [even number]
  median(c(1:3,100,1000)) #= 3 [odd, robust]
```

Description: 関数の簡単な説明

Usage: 関数の呼び出し方

`median(x, na.rm = FALSE)`

x: 必須の引数

na.rm: 省略可能な引数

デフォルト値はFALSE

Arguments: 各引数の詳しい説明

Details: 関数の動作の詳しい説明

Values: 戻り値の説明

Reference: 方法に関する文献

See Also: 関連するコマンド

Examples: 実行例

(参考) plotのオプション

- `main="title"` グラフのタイトル
- `xlab(ylab)="label"` x軸(y軸)のラベル
- `log="xy"` 対数軸の指定
- `xlim(ylim)=c(0,100)` x軸(y軸)の値の範囲の設定
- `type="l" "p"(点), "l"(線), "b"(両方), "n"(枠だけ) など`
- `lty=1` プロットする線の種類
- `pch=1` プロットする点の種類(文字)
- `col=2` プロットする点および線の色
- `cex=0.8` プロットする文字の大きさ

ベクトルとして指定することにより、
点ごとに色や種類を変更することも
できる。例) `pch=c(1,2,1,3,2)`

.	▲	+	×	◇	▽	▣	✳	◆	⊕
pch	1	2	3	4	5	6	7	8	9
col	1	2	3	4	5	6	7	8	9
cex	0.25	0.5	0.75	1	1.25	1.5	1.75	2	2.25

- ❖ plotのオプションは、`help(plot)`だけでは調べられない。一部のオプションは `help(plot.default)`, `plot(par)`を参照する必要がある。
- ❖ 既存のグラフ上に線を重ねる場合は`lines`, 点を重ねる場合は`points`を使う。
- ❖ 凡例をつけたいときは`legend`関数を使う。

宿題の確認

```
> 0:20  
0から20までの整数を並べたベクトル(0, 1, 2, ..., 20)
```

```
> 0:20/20  
上のベクトルを20で割ったもの(0, 1/20, 2/20, ..., 1)
```

```
> 0:20/20*pi  
上のベクトルに円周率πをかけたもの(0, π/20, 2π/20, ..., π)
```

```
> sin(0:20/20*pi)  
上のベクトルの各要素のサインをとったもの(0, sin(π/20), ... sin(π))
```

```
> plot(sin(0:20/20*pi))  
上のベクトルの各要素の値を順にY軸にとってプロットしたもの
```

```
> plot(sin(0:20/20*pi), type="l")  
上のプロットを線でつないだもの (0≤x≤π の範囲のサインカーブ)
```

エディタからのコマンドの入力と実行

- メニューから ファイル→新規文書 でエディタが開く
- Rのコマンド群を入力(複数行にまたがってよい)



- マウスで実行するコマンド群を選択して Command+Return を押すと、そのコマンド群がコンソール上にコピーされて実行される

行列の作成

一つのベクトルを行列の形に並べる

```
> v <- 1:9  
> m1 <- matrix(v, 3, 3)  
> m1  
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8  
[3,]    3    6    9
```

9つの要素を持つベクトル

これを3行3列の行列として定義

複数のベクトルを行または列として束ねる

```
> a <- c(1,3,5)  
> b <- c(2,4,9)  
> c <- c(3,5,7)  
> m2 <- cbind(a,b,c)  
> m2  
      a b c  
[1,] 1 2 3  
[2,] 3 4 5  
[3,] 5 9 7
```

3つの要素を持つベクトル3つ

これらを列(縦)に並べて行列とする

行列の要素の取り出し

```
> m1  
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8  
[3,]    3    6    9  
> m1[2,3]  
[1] 8          2行3列目の要素  
> m1[2,]  
[1] 2 5 8      2行目の要素すべてをベクトルとして取り出す  
> m1[,3]  
[1] 7 8 9      3列目の要素すべてをベクトルとして取り出す  
> m1[1:2,2:3]  
      [,1] [,2]  
[1,]    4    7  1,2行目と2,3列目の要素からなる  
[2,]    5    8  部分行列を取り出す  
> dim(m1)  
[1] 3 3          行列の大きさ(3行3列)
```

行列の演算

```

> m1+m2
      a   b   c
m1 = [1,] 2   6 10 要素ごとの足し算
  1 4 7 [2,] 5   9 13
  2 5 8 [3,] 8 15 16
  3 6 9 > m1*m2

      a   b   c
m2 = [1,] 1   8 21 要素ごとのかけ算
  1 2 3 [2,] 6 20 40
  3 4 5 [3,] 15 54 63
  5 9 7 > m1 %% m2

      a   b   c
[1,] 48  81  72 行列積
[2,] 57  96  87
[3,] 66 111 102

> t(m1)
[,1] [,2] [,3]
[1,]    1    2    3 転置行列(行と列の入れ換え)
[2,]    4    5    6
[3,]    7    8    9

```

データフレーム

- Rの統計解析でもっとも基本的となるデータ。
- 行列のような表形式のデータだが、各列が一般に異なる型のデータを持ちうる（行列はすべてが同じ型のデータ）。
- 多くの場合、各行が個体を、各列が個体が持つ属性を表す。
- 通常、タブ区切りテキストやエクセルファイル等から読み込んで処理する。

lung_cancer.txt
肺がん患者の遺伝子発現
プロファイルデータ(GEO:
GDS3257)の抜粋

tissue: 腫瘍or正常
smoking: 喫煙歴
stage: 癌のステージ
gender: 性別
gene1 – gene6:
遺伝子発現データ

ID_REF	tissue	smoking	stage	gender	gene1	gene2	gene3
GSM254629	tumor	never	stage I	female	7.4191	5.9318	5.67496
GSM254648	tumor	never	stage I	female	7.5627	6.93398	5.76701
GSM254694	tumor	never	stage I	female	7.54599	7.53287	5.84134
GSM254701	tumor	never	stage I	female	8.31452	7.88291	5.44759
GSM254728	tumor	never	stage I	female	7.19835	6.58398	4.79089
GSM254726	tumor	never	stage I	male	11.9811	8.45595	5.7083
GSM254639	tumor	never	stage II	female	7.41762	7.75681	4.74084
GSM254652	tumor	never	stage II	female	7.62703	7.7446	4.69937
GSM254700	tumor	never	stage II	female	7.40064	10.1866	4.92612
GSM254625	tumor	never	stage II	male	11.9	8.89865	6.69416
GSM254636	tumor	never	stage III	female	7.09852	6.39122	4.54743
GSM254659	tumor	never	stage III	female	7.39159	7.06924	5.12113
GSM254680	tumor	never	stage III	female	7.32462	7.24284	4.90953
GSM254686	tumor	never	stage III	female	7.65883	7.93856	5.09535
GSM254718	tumor	never	stage III	female	7.67937	7.03277	5.64789
GSM254674	tumor	never	stage IV	male	11.0711	6.3368	5.48673
GSM254668	tumor	former	stage I	male	10.9011	6.52462	5.19564

データフレームの読み込み(1)

```
read.table(file, options... )  
read.delim(file, options... )
```

Options: sep=列の区切り文字(タブ、空白文字など)
header=ヘッダの有無(TRUE/FALSE)
row.names=各行の名前を何カラム目からとるか など

タブ区切りテキストファイル lung_cancer.txt からデータの読み込み
区切り文字はタブ(\t)、ヘッダ行あり、1列目をヘッダ列として指定

```
> cancer <- read.table("lung_cancer.txt", sep="\t", header=T  
row.names=1)  
  
> head(cancer)          データの先頭数行を表示して内容を確認  
  tissue smoking   stage gender    gene1    gene2 ...  
GSM254629  tumor  never  stage I female  7.41910 5.93180 ...  
GSM254648  tumor  never  stage I female  7.56270 6.93398 ...  
GSM254694  tumor  never  stage I female  7.54599 7.53287 ...  
GSM254701  tumor  never  stage I female  8.31452 7.88291 ...  
...  
> fix(cancer)          エクセルデータを使ったcancerの編集
```

データフレームの読み込み(2)

	ヘッダなし	ヘッダ行あり	ヘッダ行、ヘッダ列あり																																																	
ファイルの形式	<table border="1"><tr><td>198</td><td>119</td></tr><tr><td>192</td><td>83</td></tr><tr><td>191</td><td>110</td></tr><tr><td>191</td><td>91</td></tr></table>	198	119	192	83	191	110	191	91	<table border="1"><thead><tr><th>Height</th><th>Weight</th></tr></thead><tbody><tr><td>198</td><td>119</td></tr><tr><td>192</td><td>83</td></tr><tr><td>191</td><td>110</td></tr><tr><td>191</td><td>91</td></tr></tbody></table>	Height	Weight	198	119	192	83	191	110	191	91	<table border="1"><thead><tr><th>ID</th><th>Height</th><th>Weight</th></tr></thead><tbody><tr><td>1</td><td>198</td><td>119</td></tr><tr><td>2</td><td>192</td><td>83</td></tr><tr><td>3</td><td>191</td><td>110</td></tr><tr><td>4</td><td>191</td><td>91</td></tr></tbody></table>	ID	Height	Weight	1	198	119	2	192	83	3	191	110	4	191	91	<table border="1"><thead><tr><th></th><th>Height</th><th>Weight</th></tr></thead><tbody><tr><td>1</td><td>198</td><td>119</td></tr><tr><td>2</td><td>192</td><td>83</td></tr><tr><td>3</td><td>191</td><td>110</td></tr><tr><td>4</td><td>191</td><td>91</td></tr></tbody></table> <p>先頭行のみ要素数が一つ少ない</p>		Height	Weight	1	198	119	2	192	83	3	191	110	4	191	91
198	119																																																			
192	83																																																			
191	110																																																			
191	91																																																			
Height	Weight																																																			
198	119																																																			
192	83																																																			
191	110																																																			
191	91																																																			
ID	Height	Weight																																																		
1	198	119																																																		
2	192	83																																																		
3	191	110																																																		
4	191	91																																																		
	Height	Weight																																																		
1	198	119																																																		
2	192	83																																																		
3	191	110																																																		
4	191	91																																																		
read.table オプション	header=FALSE	header=TRUE	header=TRUE, row.names=1	(オプションなしで 自動認識)																																																

read.table が読み込みのための基本的な関数だが、デフォルト値の異なる読み込み関数がいくつか用意されている

read.table(file)	デフォルトでセパレータまたは空白文字、header=FALSE
read.delim(file)	デフォルトでセパレータがタブ、header=TRUE
read.csv(file)	デフォルトでセパレータがコンマ、header=TRUE

データフレームの情報表示

```
str(data)      data のデータ構造の表示  
summary(data) data の内容サマリの表示
```

```
> str(cancer)    各列のデータ型と内容の一部を表示
```

```
'data.frame': 107 obs. of 10 variables:  
 $ tissue : Factor w/ 2 levels "normal","tumor": 2 2 2 2 2 2 2 2 2 ...  
 $ smoking: Factor w/ 3 levels "current","former",...: 3 3 3 3 3 3 3 3 3 ...  
 $ stage   : Factor w/ 4 levels "stage I","stage II",...: 1 1 1 1 1 1 1 2 2 2 ...  
 $ gender  : Factor w/ 2 levels "female","male": 1 1 1 1 1 2 1 1 1 2 ...  
 $ gene1   : num  7.42 7.56 7.55 8.31 7.2 ...  
 $ gene2   : num  5.93 6.93 7.53 7.88 6.58 ...  
 ....
```

```
> summary(cancer)  各列のデータの分布の要約を表示
```

	tissue	smoking	stage	gender	gene1	gene2
normal:49	current:40	stage I :45	female:38	Min. : 6.729	Min. : 5.728	
tumor :58	former :36	stage II :35	male :69	1st Qu.: 7.543	1st Qu.: 6.761	
	never :31	stage III:21		Median :11.224	Median : 8.452	
		stage IV : 6		Mean :10.074	Mean : 8.894	
				3rd Qu.:11.875	3rd Qu.:11.130	
				Max. :12.659	Max. :12.293	

因子 factor

- **gender (male, female)**のように、限られた数のカテゴリーで表現されるもの。カテゴリー変数、要因などともいう。
- 因子の取り得る値を水準levelという。
例) genderは2つ(female, male)、smokingは3つ(current, former, never)の水準を持つ
- 文字列として表示されるが、内部的には整数値で保持されている。

データフレームの要素の取り出し

```
> cancer[5,]          5行目の人のデータの取り出し
  tissue smoking stage gender gene1 gene2 gene3 gene4 gene5 gene6
GSM254728 tumor never stage I female 7.19835 6.58398 4.79089 7.26575 7.46492 5.02376

> cancer[,5]          5列目(gene1)の取り出し
[1] 7.41910 7.56270 7.54599 8.31452 7.19835 11.98110 7.41762 7.62703
[9] 7.40064 11.90000 7.09852 7.39159 7.32462 7.65883 7.67937 11.07110
[17] 10.90110 9.30280 11.42620 12.10840 11.00710 11.43570 9.58576 11.37820
[25] 11.46580 11.98080 7.17584 11.62510 11.22410 7.45557 7.70254 11.65150
[33] 11.01420 12.03040 7.00001 7.53943 10.81220 11.47220 12.05180 11.37000
....
```

```
> cancer$gene1  gene1 データの取り出し(名前によるアクセス。cancer[,5]と同じ)
```

```
> cancer["GSM254728",] 行の取り出し(名前によるアクセス。cancer[5,]と同じ)
```



```
> cancer[7:10,2:7]    7-10行、2-7列の要素からなる部分データの取り出し
  smoking stage gender gene1 gene2 gene3
GSM254639 never stage II female 7.41762 7.75681 4.74084
GSM254652 never stage II female 7.62703 7.74460 4.69937
GSM254700 never stage II female 7.40064 10.18660 4.92612
GSM254625 never stage II male 11.90000 8.89865 6.69416
```

データフレームの操作 条件式による部分データの抽出

subset(データフレーム, 条件式)

性別が女性でgene1の発現が8以上

```
> subset(cancer, gender=="female" & gene1 > 8)
  tissue smoking stage gender gene1 gene2 gene3 gene4 gene5 gene6
GSM254701 tumor never stage I female 8.31452 7.88291 5.44759 5.99769 7.66485 5.06010
GSM254687 tumor current stage III female 10.15150 7.64551 6.61338 7.24318 7.91872 7.36862
```

喫煙歴がある人(過去または現在に) 結果を変数に保存

```
> cancer.smoking <- subset(cancer, smoking %in% c('former', 'current'))
```

その数(行数をカウント)

```
> nrow(cancer.smoking)
```

```
[1] 76
```

A %in% B ベクトルAの各要素について、ベクトルBの要素のいずれかと一致すればTRUE、そうでなければFALSEを返す。

```
> c("A", "B", "C", "B", "A") %in% c("B", "C")
[1] FALSE TRUE TRUE TRUE FALSE
```

データフレームのファイルへの書き出し

```
write.table(data, file="", options... )
```

タブ区切りテキストとしてファイルに保存

```
> write.table(cancer.smoking, sep="\t",
  file="cancer.smoking.txt")
```

そのまま読み込んで表示してみる

```
> read.table("cancer.smoking.txt")
```

Rによるデータ解析 plotによる散布図の作成

各カラム対総当たりの散布図を作成する

```
> plot(cancer)
```

pairs(cancer)としても同じ

gene3とgene4の散布図を作成する

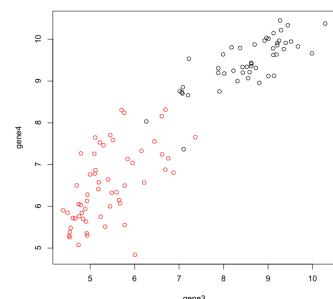
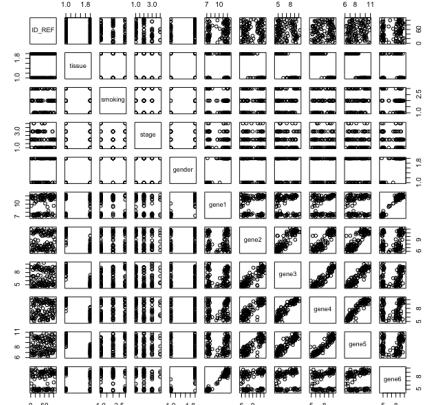
```
> plot(cancer$gene3, cancer$gene4)
以下もほぼ同じプロットを生成する
```

```
> plot(gene4 ~ gene3, cancer)
```

tissue (cancer/normal) によって点の色づけする

```
> plot(gene4 ~ gene3, cancer,
  col=tissue)
```

cancer\$tissue は normal=1, tumor=2 として定義されており、各点が 1=black, 2=redで色づけされる。



Rによるデータ解析

回帰直線の追加

gene3とgene4の散布図(前掲)

```
> plot(gene4 ~ gene3, cancer)
```

線型モデルを用いた直線への当てはめ(回帰直線の決定)

```
> lm(gene4 ~ gene3, cancer)
```

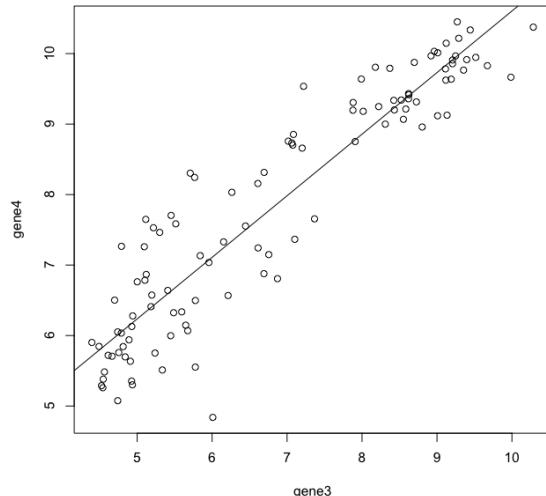
... (結果は画面に出力される) ...

結果を変数 result.lm に代入

```
> result.lm <- lm(gene4 ~ gene3,  
cancer)
```

この結果を使って回帰直線をプロットに追加

```
> abline(result.lm)
```



モデル式と線型モデル

- `lm` の第1引数で指定される式は、データを当てはめるモデル式を簡潔に表現したものとなっている

例) x_1, x_2 を説明変数、 y を目的変数として、

$$Y = a * x_1 + b * x_2 + c + \varepsilon \quad (\text{残差})$$

という式に当てはめる(残差が小さくなるよう係数 a, b, c の最適な値を決める)ことを

$$Y \sim X_1 + X_2$$

と記述する。

- モデル式が係数の一次式で表されるモデルを**線型モデル**といい、最小二乗法で当てはめが行われる。`lm`関数では、係数の決定とともに、係数が有意に0でないといえるかの検定も行われる。
- 線型モデルへの当てはめは、回帰分析だけでなく、カテゴリ変数を説明変数とした分散分析においても用いられる。

Rによるデータ解析 回帰分析結果の詳細表示

```
> par(mfrow=c(2,2)) # 4つのプロットを 2x2 の領域に同時に表示
> plot(result.lm) # 解析結果からプロットの作成
```

プロットから誤差(残差)の分布が適切か(正規分布に従っているか)を確認 →

```
> summary(result.lm) # 解析結果の詳細表示
```

Call:

```
lm(formula = gene4 ~ gene3, data = cancer)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-2.27983	-0.41889	-0.01643	0.45094	1.44834

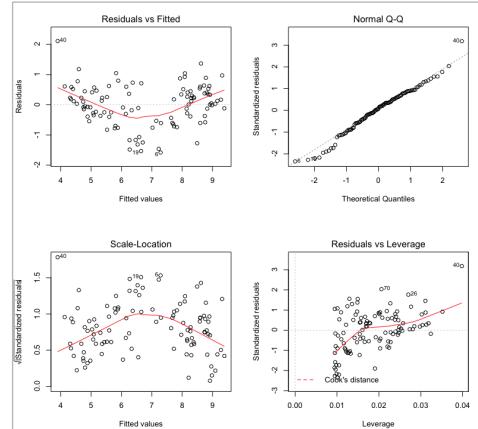
Coefficients:

	Estimate	Std. Error	t value	Pr(> t)		
(Intercept)	1.86604	0.24801	7.524	1.9e-11 ***		
gene3	0.87403	0.03514	24.870	< 2e-16 ***		

Signif. codes:	0 ****	0.001 ***	0.01 **	0.05 *	0.1 .	1

Residual standard error: 0.6395 on 105 degrees of freedom
Multiple R-squared: 0.8549, Adjusted R-squared: 0.8535
F-statistic: 618.5 on 1 and 105 DF, p-value: < 2.2e-16

解析が終わったらプロットウィンドウを閉じるか par(mfrow=c(1,1))して、プロットの設定を元に戻しておく



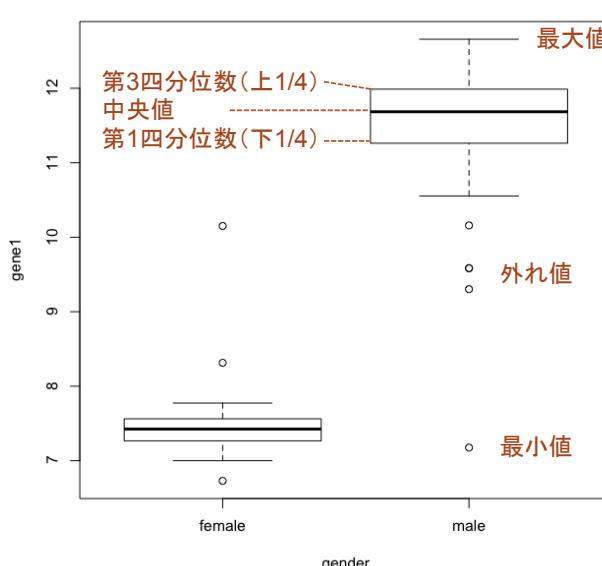
各回帰係数が0でない値を持つといえるかを統計的に検定

← 回帰分析全体の有意性

Rによるデータ解析 boxplot(箱ひげ図)の作成

性別ごとのgene1の発現量をboxplotで比較する

```
> plot(gene1 ~ gender, cancer)
```



または
boxplot(gene1 ~ gender, cancer)

Rによるデータ解析

t検定

2組の標本の平均値に差があるかどうかを検定する

男女でgene1の発現量に違いがあるかどうかをt検定で検定する

男性のgene1の発現量を抽出

```
> gene1.m <- subset(cancer, gender=="male")$gene1
```

女性のgene1の発現量を抽出

```
> gene1.f <- subset(cancer, gender=="female")$gene1
```

t検定の実行 (等分散性を仮定しないWelchの検定)

```
> t.test(gene1.m, gene1.f)
   Welch Two Sample t-test

data:  gene1.m and gene1.f
t = 30.734, df = 103.48, p-value < 2.2e-16
```

別法(データフレームから直接データを取り出して検定を実行する)

```
> t.test(gene1 ~ gender, cancer)
```

Rによるデータ解析

分散分析

- 線形モデルを用いて、因子の水準によって目的変数の平均値に違いがあるかを検定する
- 因子が3つ以上の水準をもつ場合も、また考慮する因子が複数ある場合でも使える

癌のステージ(I~IVの4つの水準がある)によって、gene1の発現に違いがあるかどうかを分散分析で検定する

```
> aov(gene1 ~ stage, cancer)
Call:
aov(formula = gene1 ~ stage, data = cancer)

Terms:
          stage Residuals
Sum of Squares    47.0675 399.6562
Deg. of Freedom      3        103

Residual standard error: 1.969812
Estimated effects may be unbalanced
```

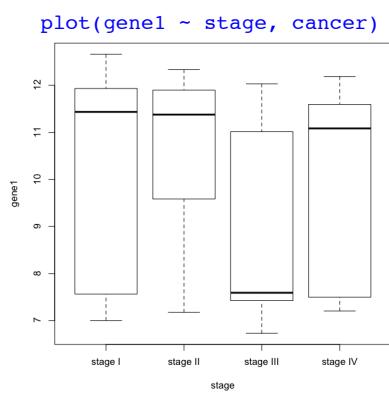
結果を変数 result.aov に格納してsummaryで出力

```
> result.aov <- aov(gene1 ~ stage, cancer)
> summary(result.aov)
```

Df	Sum Sq	Mean Sq	F value	Pr(>F)
stage	3	47.1	15.69	4.043 0.00921 **
Residuals	103	399.7	3.88	

分散分析表

stageがgene1の変動に及ぼす効果の検定



解析結果の保存

```
# 指定したオブジェクトを "cancer.Robj" という
# ファイルに保存
> save(cancer, result.lm, result.aov,
       file="cancer.Robj")

# 保存したオブジェクトをファイルからロードする
> load("cancer.Robj")

# 現在のワークスペース中のオブジェクトをすべて
# デフォルトの保存ファイル(".RData")に保存。
> save.image()

• 明示的にsave.image()コマンドを実行しなくても、終了時にワークスペースを
  保存するか聞かれるので、そこで保存を選択することにより保存できる。
• ここで保存したデータは、現在の作業ディレクトリが起動時のディレクトリと同じであれば、次回起動時に自動的にロードされるが、ディレクトリを変更した場合はload(".RData")で明示的にロードする必要がある。
```

リスト

- ベクトルは同一の型のデータを要素とするが、リストは任意の一般に異なる型のデータを要素として含むことができる。
> list(1, "a")
- リストは、ベクトルや行列、リストをも要素として含むことができるため、複雑なデータを表現できる。
> list(c(1,2), c("a","b"), list(1, "a"))
- リストの各要素は[[]]で参照できるほか、名前をつけて参照することもできる。
> x <- list(first=1, second="b")
> x[[1]] # 1番目の要素
[1] 1
> x\$second # second という名前の要素
[1] "b"
- データフレームのほか、各種の統計処理の結果などもリストとして表現されている。リストの内部構造はstr関数で確認できる。
> str(x)
> str(cancer)

オブジェクトの属性とクラス

- 統計解析の結果などは、一般に決まった型と名前の要素を持ち、一定のクラス名をアサインされたリスト(オブジェクト)として返される。

例) aov 関数の結果を格納した変数 result.aov

```
> class(result.aov)      クラス名の表示  
[1] "aov" "lm"  
> names(result.aov)    要素名の一覧の表示  
[1] "coefficients" "residuals" "effects" "rank" ...  
> str(result.aov)      データ構造の階層的な表示  
List of 13  
 $ coefficients : Named num [1:4] 10.334 0.195 -1.584 -0.224  
   ..- attr(*, "names")= chr [1:4] "(Intercept)" "stagestage II" ...  
 $ residuals      : Named num [1:107] -2.91 -2.77 -2.79 -2.02 ...  
   ..- attr(*, "names")= chr [1:107] "GSM254629" "GSM254648" ...
```

このオブジェクトの詳細は、`help(lm)` の `Value` セクションで確認できる

総称関数 generic function

- `plot`, `print`, `summary`など多くの関数は、引数となるオブジェクトのクラスによって、異なる関数が呼び出されるようになっている。これらを総称関数という

- 例) `plot` は、引数のクラスによって `plot.data.frame`, `plot.formula`, `plot.lm` などが呼び出される。デフォルトでは、`plot.default` が呼び出される。

- これにより、Rでは典型的には以下のようない定型的な処理の流れによって解析が進められる。

```
> result <- some.function(data, opt=value, ...)  
> summary(result)    解析結果の要約の表示  
> plot(result)       解析結果に基づくプロットの作成
```

参考) コンソールで単に `x` と入力したときは、`print(x)` が実行されている。
`print` は総称関数であり、`x` のクラスによって表示される内容は違っている。

関数の作成(1)

- **function** 関数によって、新たな関数を作成できる

```
function (引数リスト) { 関数本体 }
```

例) 引数の2乗を計算して返す関数を **square** として定義

```
square <- function(x) {  
  sq <- x^2  
  return(sq)  
}
```

引数:呼び出し時に指定した値がxに代入され、関数本体が実行される
戻り値:関数が返す値(ここではsq=x²)

```
> square(1:5)      引数1:5を与えて関数の呼び出し  
[1] 1 4 9 16 25
```

関数の作成はエディタを使って行うとよい

関数の作成(2)

- 関数は、通常はファイル上に作成し、読み込んで使う。ファイル上に作成したRコマンドを読み込むには**source("ファイル名")**とする。

ファイル plotAll.R (plotAll: 指定した2つの部分データフレーム間での総当たりプロットを作成する関数)

```
plotAll <- function(x, y=x) {  
  par(mfrow=c(ncol(x), ncol(y)))  
  for (i in 1:ncol(x)) {  
    x.name=names(x)[i]  
    for (j in 1:ncol(y)) {  
      y.name=names(y)[j]  
      if (x.name == y.name) {  
        plot(x[,i], main=x.name)  
      } else {  
        plot(x[,i], y[,j],  
              xlab=x.name, ylab=y.name)  
      }  
    }  
  }  
}
```

プログラムの読み込みと実行

```
> source("plotAll.R")  
> plotAll(cancer[,1:4], cancer[,5:10])
```

繰り返し

```
for (変数 in ベクトル) { 式 }
```

「変数」に「ベクトル」の要素を順次代入して、「式」を繰り返し実行する。

条件分岐

```
if (条件式) { 式1 } else { 式2 }
```

「条件式」がTRUEなら、「式1」を実行し、FALSEなら「式2」を実行する。

パッケージの利用

- 様々なパッケージをインストール、ロードすることによりRの機能を拡張できる。
- バイオインフォマティクス関連では、**Bioconductor**プロジェクトによって、膨大な種類のパッケージが提供されている。



The screenshot shows the R application window with the 'Packages' menu highlighted. The menu items include 'パッケージとデータ' (selected), 'その他' (Other), and 'Quartz'. Below the menu, there are three sub-options: 'パッケージマネージャ' (Package Manager), 'パッケージインストーラ' (Package Installer), and 'データマネージャ' (Data Manager).

パッケージインストーラ
パッケージのインストール/更新

CRAN (バイナリ)
一覧を取得 バイナリ形式パッケージ Q パッケージ検索

パッケージ	導入済みバージョン	リポジトリにあるバージョン
tsbugs	1.1	
TScmpare	2012.8-1	
Tsdbi	2012.8-1	
tsDyn	0.9-2	
TSfNN	1.1	
tsseries	0.10-30	
tsseriesChaos	0.1-11	
tsf	2012.4-1	
TSfame	2012.8-1	
TsgenSymbol	2012.8-1	
TSHistQuote	2012.8-1	
TSRRC	0.1-2	
tslars	1.0	
tsplot	1.0	
TSMySQL	2012.8-1	
tsne	0.1-2	
TSP	1.0-7	
TSPC	1.0	

インストールする場所
 システム (Rフレームワーク内)
 ユーザーリア
 特定の場所(インストール時に指定)
 libPaths()での指定に従う

選択をインストール □ すべてのパッケージも含める **すべてアップデート**

パッケージマネージャ
パッケージのロード

CRAN (バイナリ)
一覧を取得 バイナリ形式パッケージ Q パッケージ検索

パッケージ	説明
affy	Methods for Affymetrix Oligonucleotide Arrays
affylo	Tools for parsing Affymetrix data files
affypls	Interpretation of Affymetrix spotted data
ALL	A data package
annotation	Annotation microarray
annotationDbi	Annotation Database Interface
AnnotationForge	Code for Building Annotation Database Packages
Biobase	Biobase: Base Functions for Bioconductor
BiocGenerics	Common utilities for Bioconductor packages
BiocInstaller	Install/Update BiocMart and CRAN Packages
BiocManager	Manage BioMart databases (e.g. Ensembl, CCDS, Wormbase)
BiocNeighbors	String operations, matching, distances, and matching
biowebBase	Basic graphic utilities for visualization of genomic data
bitops	A class for vectors of bit booleans
blockmodeling	An R package for Generalized and classical blockmodeling of
boot	Bootstrap Functions (originally by Angelo Carty for S)

Methods for Affymetrix Oligonucleotide Arrays 

Documentation for package 'affy' version 1.36.0.

DESCRIPTION file
Overview of examples and package vignettes: [browse vignettes](#).
See also: [?affy](#) for details on how to run them.
Package NEWS.

Help Pages
A B C D E F G H I J L M S O P Q R S T U V X Msc
-- A --

コマンドラインからの実行

パッケージのインストール
`install.packages(パッケージ名)`

パッケージの更新
`update.packages(パッケージ名)`

パッケージのロード
`library(パッケージ名)`

82

Unixによるテキストファイル処理

2016/02/18

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/4_text/

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。

実習で使用するデータ

- 講習で使用するデータは以下のフォルダ内。
- ファイルがあることを確認してください。

```
~/data/4_text/
```

(確認するためのコマンド)

```
$ ls 4_text/
```

主なファイルの内容

batter.txt	2014年セリーグ打撃成績上位5名
ecoli.sam	マッピング結果ファイル
ecoli.gtf	アノテーションテーブルファイル
ecoli.htseq	アノテーションテーブルを使用して得られたリード数

コマンド復習

- **wc [ファイル名]**
 - ファイルの行数、単語数、文字数を出力する
- **head [-行数] [ファイル名]**
 - ファイルの先頭から指定した行数(指定しないと10行)を出力する
- **tail [-行数] [ファイル名]**
 - ファイルの最後から指定した行数(指定しないと10行)を出力する
- **less [ファイル名]**
 - ファイルの内容を閲覧する

本講で扱うテキスト処理コマンド

- grep 正規表現パターンの検索
- 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

正規表現による文字列検索 (grep)

- 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) grep 'GO' 1433T_HUMAN.sprot
 - 1433B_HUMAN.sprot から GO を含む行を検索する。

```
...  
DR Genevestigator; P27348; -.  
DR GermOnline; ENSG00000134308; Homo sapiens.  
DR GO; GO:0005813; C:centrosome; IDA:HPA.  
DR GO; GO:0005634; C:nucleus; IDA:HPA.  
DR Bgee; P27348; -.  
....
```



```
DR GO; GO:0005813; C:centrosome; IDA:HPA.  
DR GO; GO:0005634; C:nucleus; IDA:HPA.
```

正規表現による文字列検索 (grep)

- `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) `grep 'GO' 1433T_HUMAN.sprot`
 - 1433T_HUMAN.sprot から GO を含む行を検索する。
 - 例) `grep '^FT' 1433T_HUMAN.sprot`
 - 1433T_HUMAN.sprot から FT で始まる行を検索する。
 - `grep -v` パターンを含まない行を出力する。
 - `grep -i` 大文字小文字を区別しない。
 - `grep -w` パターンを単語としてマッチ
 - ファイル名は複数指定可能
 - ファイル名を省略すると、標準入力から文字列を読み込んでパターンを検索する

正規表現

grepは「正規表現」によってパターンを指定し、照合したい文字列集合を規定する

- 通常の文字列はそのまま表現される
 - 例) File1 (File1にマッチ)
- 特殊な意味を持つ文字(メタキャラクタ)によって規則を表現
 - 例) []は文字集合を規定する
`File[1-3]` (File1, File2, File3 のいずれにもマッチ)
 - \によってメタキャラクタの特殊な意味を打ち消せる
 - 例) \[abc\] ([abc] という文字列にマッチ)

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。

正規表現(一部)

- **. (ドット)　任意の1文字**
 - 例) a.c
abc, adc など、aとcの間に任意の1文字を含む文字列にマッチ
- **[](角形括弧) 文字の集合**
 - 例) [ad3@]
a, d, 3, @のいずれにもマッチ
 - 例) [a-d]
a, b, c, dのいずれにもマッチ
 - 例) [^abd]
a, b, d 以外のいずれにもマッチ
- **^ 行の先頭 \$ 行の終端**
 - 例) ^ID
行の先頭がIDである行とマッチ
- *** 0回以上の繰り返し**
 - 例) a.*m
aとmの間に任意の文字列を含む(am, arm, alarm, am amなど)

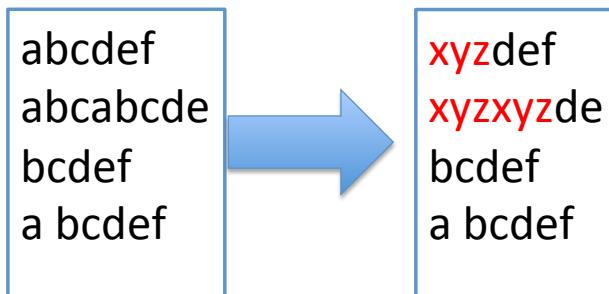
演習 (grep)

- ecoli.sam ファイルから、grep (egrep)コマンドを用いてヘッダ行(行頭に@を含む)を表示せよ。

文字の置換など (sed)

```
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) sed 's/abc/xyz/g' file
file中の文字列abcをすべてxyzに置き換える。



文字の置換など (sed)

```
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) sed 's/abc/xyz/g' file
file中の文字列abcをすべてxyzに置き換える。
- 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) sed 's/:/ /' file
各行で最初に出現した : をスペースに置き換える

```
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るので、パターンの指定は常に '' で囲むようにする。
```

文字の置換など (sed)

通常、wcを使ってファイルの行数を出すと、ファイル名まで出力されてしまう。行数だけ取り出してみよう。

```
$ wc -l ecoli.gtf | sed 's/ .*//g'
```

wcの出力結果から、スペース以降の文字を削除(=何もないものに置換)している。

```
$ wc -l ecoli.gtf
```

演習) この2つのコマンドをそれぞれ実行し、違いを見よ。

行の並べかえ(sort)

- sort [オプション] [ファイル名...]
 - ファイルを行単位で並べかえる。
 - -k FLD1,FLD2 ソートのキーを、スペース文字で区切られたフィールド単位で指定できる(FLD1開始フィールド、FLD2終了フィールド)。
 - -k 2,2 — 第2フィールドをキーとしてソート

Murton	T	.338	14	84
Kikuchi	C	.325	11	58
Yamada	S	.324	29	89
Ooshima	D	.318	2	28
Luna	D	.317	17	73

Kikuchi	C	.325	11	58
Luna	D	.317	17	73
Ooshima	D	.318	2	28
Yamada	S	.324	29	89
Murton	T	.338	14	84

- -k 2,2 -k 3,3nr — 第2フィールドを1番目のキーとし、第3フィールドを2番目のキーとして、数値として(n)逆順(大きい順)で(r)ソート

Murton	T	.338	14	84
Kikuchi	C	.325	11	58
Yamada	S	.324	29	89
Ooshima	D	.318	2	28
Luna	D	.317	17	73

Kikuchi	C	.325	11	58
Ooshima	D	.318	2	28
Luna	D	.317	17	73
Yamada	S	.324	29	89
Murton	T	.338	14	84

演習 (sort)

- ecoli.htseq を使い、リード数をカウントされた回数が多いものから20個を表示せよ。
また、少ないものから20個を表示するにはどうすればよいか？

テキストファイルの処理(awk)

awk 'コマンド' ファイル

- テキストファイルを処理する多機能なコマンド
- コマンドの一般形式は
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。

テーブルデータの処理(awk)

- ・ テーブルカラムの抽出

```
awk '{print $3,$4,$5}' datafile
```

- 3,4,5カラム目を出力
- 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。

ecoli.gtf

```
...
chr eschColi_K12_refSeq stop_codon 253 255 ...
chr eschColi_K12_refSeq exon 190 255 ...
chr eschColi_K12_refSeq start_codon 337 339 ...
chr eschColi_K12_refSeq CDS 337 2796 ...
```



```
# awk '{print $3,$4,$5}' ecoli.gtf
stop_codon 253 255
exon 190 255
start_codon 337 339
CDS 337 2796
```

テーブルデータの処理(awk)

- ・ 条件を指定したフィルタリング

```
awk '$4<200 {print}' datafile
```

- 4カラム目が200未満の行を出力
- 出力フィールドが指定されていないので行全体を出力

ecoli.gtf

```
...
chr eschColi_K12_refSeq stop_codon 253 255 ...
chr eschColi_K12_refSeq exon 190 255 ...
chr eschColi_K12_refSeq start_codon 337 339 ...
chr eschColi_K12_refSeq CDS 337 2796 ...
```



```
# awk '$4<200 {print}' ecoli.gtf
chr eschColi_K12_refSeq start_codon 190 192 ...
chr eschColi_K12_refSeq CDS 190 252 ...
chr eschColi_K12_refSeq exon 190 255 ...
```

テーブルデータの処理(awk)

- テーブルカラムの抽出

```
awk '{print $1,$2,$5}' datafile
```

- 1,2,5カラム目を出力
- 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

```
awk '$3<200 {print}' datafile
```

- 3カラム目が200未満の行を出力
- 出力フィールドが指定されていないので行全体を出力

- 複数の条件の指定

```
awk '$2~/target/ && $3<200{print}' datafile
```

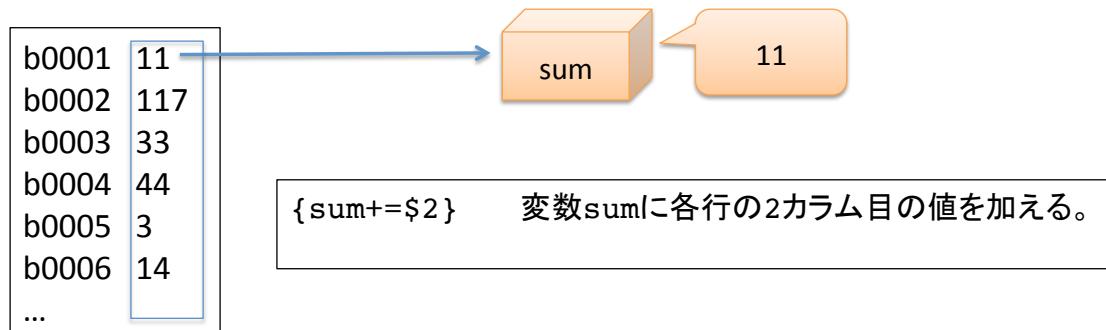
- 2カラム目にtargetを含み、3カラム目が200以下の行を出力
- 変数~/パターン/ は正規表現の照合

awk 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、常に '' で囲むようにすると良い

算術計算(awk)

- 合計値の出力

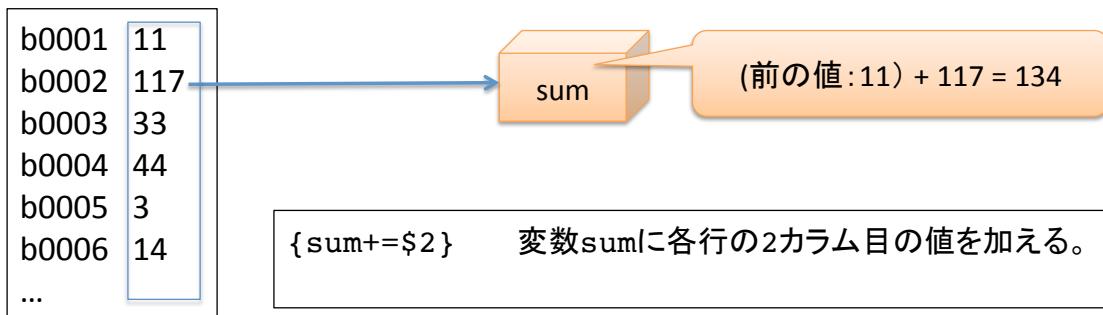
```
awk '{sum=sum+$2} END{print sum}' ecoli.htseq
```



算術計算(awk)

- 合計値の出力

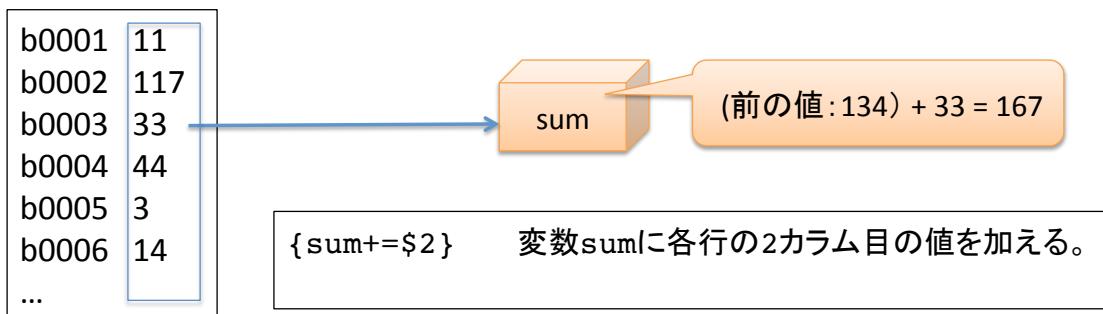
```
awk '{sum=sum+$2} END{print sum}' ecoli.htseq
```



算術計算(awk)

- 合計値の出力

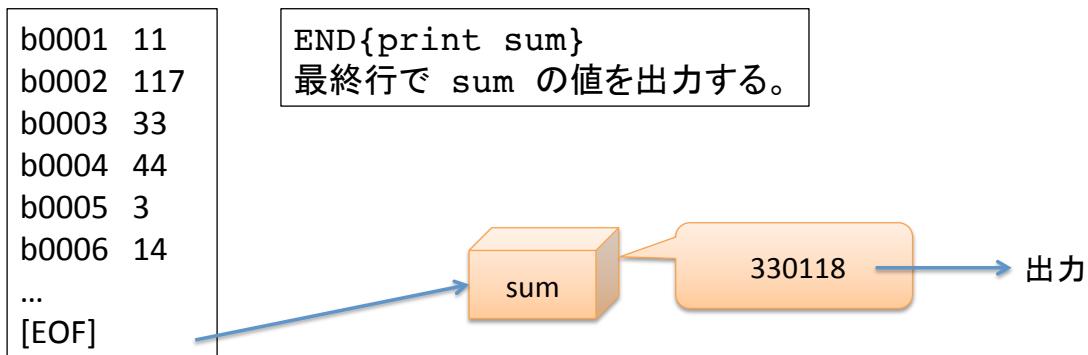
```
awk '{sum=sum+$2} END{print sum}' ecoli.htseq
```



算術計算(awk)

- 合計値の出力

```
awk '{sum=sum+$2} END{print sum}' ecoli.htseq
```



算術計算(awk)

- 合計値の出力

```
awk '{sum=sum+$2} END{print sum}' ecoli.htseq
```

- 2カラム目の合計値を出力

- プログラムは2つのブロックからなる

{sum=sum+\$2} パターン部がないのですべての行が対象となる。
変数sumに各行の2カラム目の値を加える。

sum+=x は sum=sum+xと同じ。

END{print sum} パターンENDは最終行のみにマッチ。
最終行で sum の値を出力する。

参考) パターン BEGINは先頭行のみにマッチする。これを用いて変数の初期化などができる。

例) BEGIN{sum=0} {sum+=\$2} END{print sum}

最初に変数 sum を0に初期化する。これはデフォルトの動作として省略できるため、上記のプログラムと同じ結果になる。

演習 (awk)

- awkコマンドを用いて、ecoli.htseq の2カラム目(カウント数)の平均値を出力せよ。
 - ヒント: 行数を数える必要がある。変数`ln`を使って行数を数えるにはどうすればよいか？
 - カラムの和を変数`sum`を用いて表せば、最後に`sum`を行数で割り算することで平均が出せる。割り算は `a/b` で計算できる。

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、それ以後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

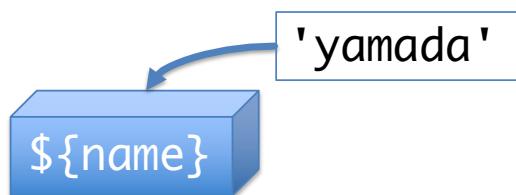
- ・ シェルでは「変数」が使用できる。
- ・ 任意でつけた名前の「変数」に、プログラムの実行状況によって、文字列や数値などさまざまなデータを記憶しておく。

`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。



クオーテーションの違い

- ・ シングルクオーテーション [']
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。
- ・ ダブルクオーテーション ["]
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}'`

`echo "${name}"`

`${name}` と表示される。

`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等は使わないこと
 - 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



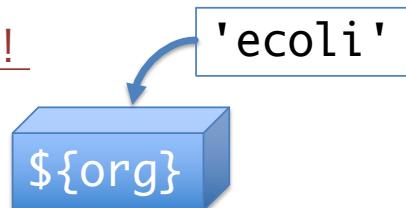
変数の使用

- ・ 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ては意味がない

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。

```
cp test.txt test.copy
```

コマンド 引数1 引数2

- 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

```
 ${1} 1番目の引数 ${2} 2番目の引数 … ${9} 9番目の引数  
 ${0} コマンド自身      ${*} 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

- GUIで使う
 - Atom
 - <https://atom.io/>
 - mi
 - <http://www.mimikaki.net/>
- コマンドラインで使う
 - emacs
 - vi

Atom

```
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

```
hiroyo — emacs — 77x18
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
```

-uu:---F1 ex3.sh All L1 (Shell-script[sh])-----
Indentation setup for shell type sh

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
- 一件づ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
```

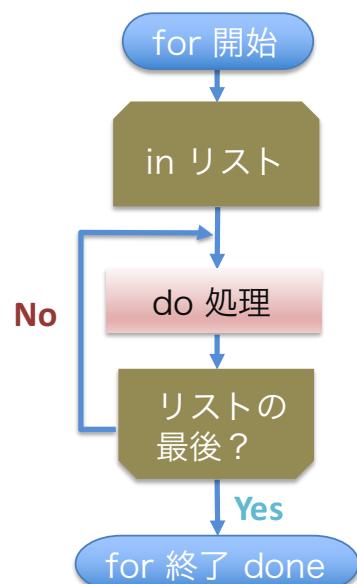
```
#!/bin/sh
for file in ./*.fasta
do
    echo ${file}
done
```

```
$ ./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つづつ代入され、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つづつ代入される）

```
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
```

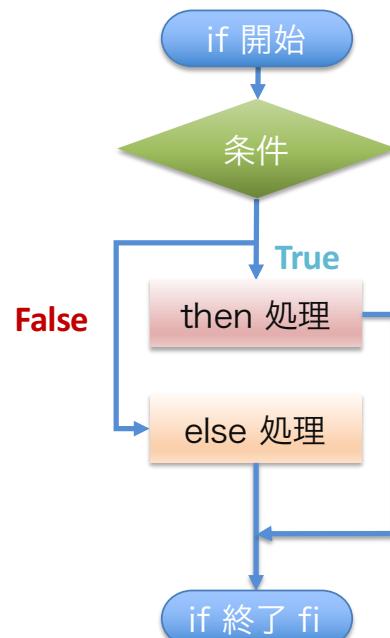
```
#!/bin/sh
if [ -f ecoli.3.sam ]
then
    echo 'ok'
else
    echo 'not ok'
fi
```

```
$ ./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
```

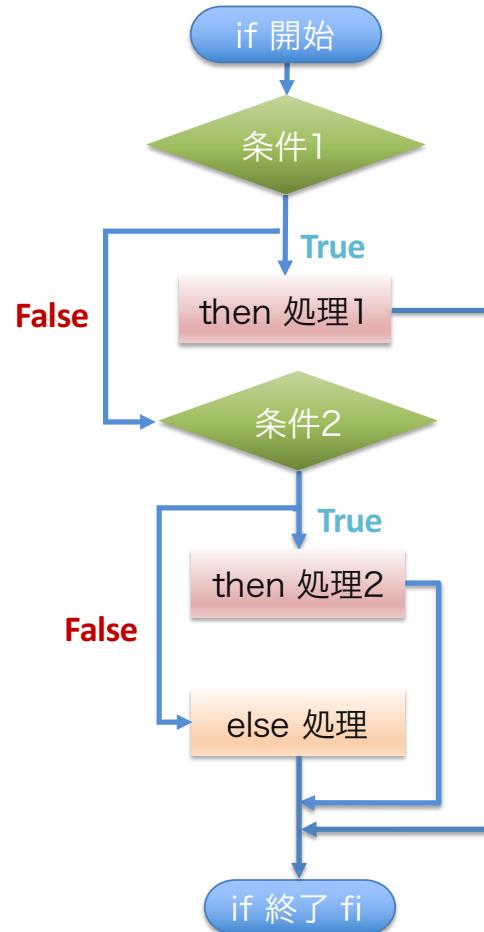


- if → 条件 → then or else → fi
- 「if」は必ず「fi」で終わらねばならない
- 条件を複数設定したい場合は「elif」を使う

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	両辺が等しくないと真
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

生物情報解析システムの使い方

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

生物情報解析システムの紹介

Computer system for biological information analysis

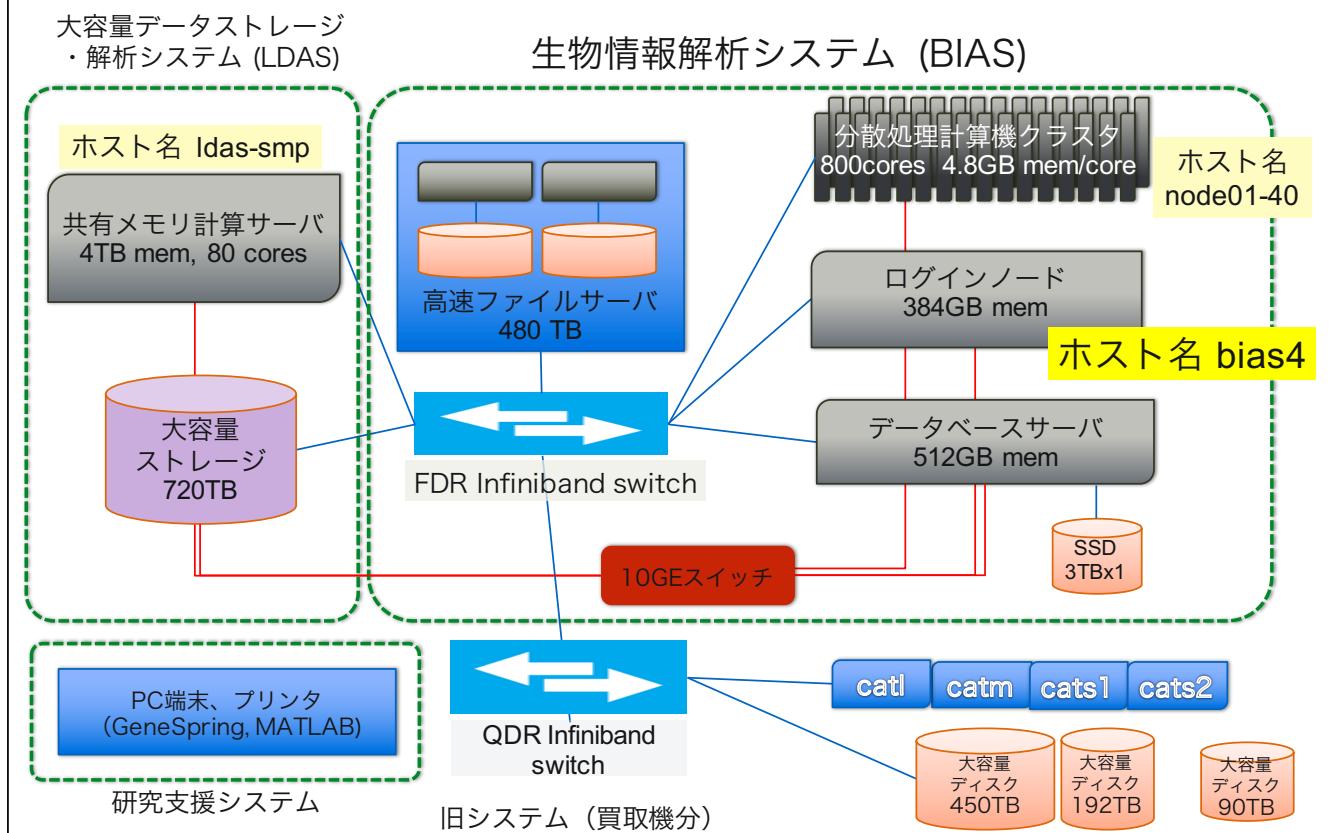


分散処理計算機クラスタ
SGI Rackable server C2112-4RP
Intel Xeon (2.8GHz) 20core/node
96GB/node Memory, 40node, 800core



高速ファイルサーバ
DDN SFA7700
Lustre file system : 480TB

生物情報解析システムの構成



生物情報解析システム 基本的な使い方：ログイン先

bias4.nibb.ac.jp

- 基生研外の方は、ユーザーアカウントの申請が必要
<http://www.nibb.ac.jp/cproom/global/appli/index.html>
- sshで接続する (telnetなどは利用できません)
- ログインノード上での作業は、プログラムの作成やファイル管理などの軽い処理にとどめ、大きな計算処理の実行はジョブ管理システム (SGE) を介して行う
- 正確なマシン名は、bias4-login.nibb.ac.jp だが、ログイン時には -login を省略可

ログイン方法：Macユーザ

- ターミナル上で
\$ ssh *username@bias4.nibb.ac.jp*
- と入力してリターン

username@bias4.nibb.ac.jp's password:

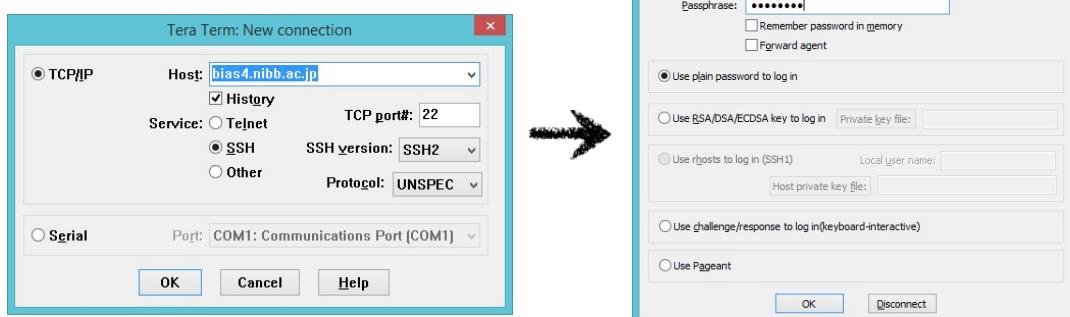
- と出たらパスワードを入力してリターン
 - 画面には何も出ません！*****等もなし！

```
$ ssh username@bias4.nibb.ac.jp
```

```
The authenticity of host 'bias4.nibb.ac.jp (133.48.33.122)' can't be established.  
RSA key fingerprint is 7b:94:9a:36:ac:60:ae:a0:14:2a:7c:0f:3c:bc:fe:24.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'bias4.nibb.ac.jp' (RSA) to the list of known hosts.  
username@bias4.nibb.ac.jp's password:  
Last login: Fri Jan 17 15:23:05 2014 from bigfox-01.nibb.ac.jp  
[username@bias4-login ~]$
```

ログイン方法：Windowsユーザ

- TeraTermProをインストール→
<http://ttssh2.sourceforge.jp/>
 - Host: bias4.nibb.ac.jp
 - Service: SSH, TCP port#: 22, SSH version: SSH2, Protocol: UNSPEC
 - User name: ユーザ名
 - Passphrase: パスワード



生物情報解析システム：ログインしてみましょう

```
$ ssh username@bias4.nibb.ac.jp
```

The authenticity of host 'bias4.nibb.ac.jp (133.48.33.122)' can't be established.

RSA key fingerprint is
7b:94:9a:36:ac:60:ae:a0:14:2a:7c:0f:3c:bc:fe:24.

最初の一回だけyes入力

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'bias4.nibb.ac.jp' (RSA) to the list of known hosts.

username@bias4.nibb.ac.jp's password: #パスワード入力

Last login: Fri Jan 17 15:23:05 2014 from bigfox-01.nibb.ac.jp

[username@bias4-login ~]\$ pwd

[username@bias4-login ~]\$ ls

- pwd でログイン後のディレクトリを、
- ls でディレクトリの内容を確認

生物情報解析システムからログアウト

```
[username@bias4-login ~]$ exit
```

```
$
```

- exitコマンドで手元のマシンに戻る

生物情報解析システムにデータをコピー

- scp コマンドで離れたUNIXマシンにデータをコピーできる
- scp (secure copy) 暗号化して送信
- コピーが終わったらbias4にログインしてファイルを確認

```
$ cd ~/data  
$ scp -r 6_sge bias4.nibb.ac.jp:  
username@bias4.nibb.ac.jp's password:  
  
$ ssh username@bias4.nibb.ac.jp  
username@bias4.nibb.ac.jp's password:  
[username@bias4-login ~]$ ls
```

scp コマンド



- 手元のMac（ローカル）から生物情報解析システム（リモート）へ

```
$ scp コピーしたいファイル username@リモートホスト名:コピー先のパス
```



- リモートからローカルへ

```
$ scp username@リモートホスト名:コピーしたいファイルのパス ローカルのパス
```

- コピー先で「：」以降を省略するとホームディレクトリになる

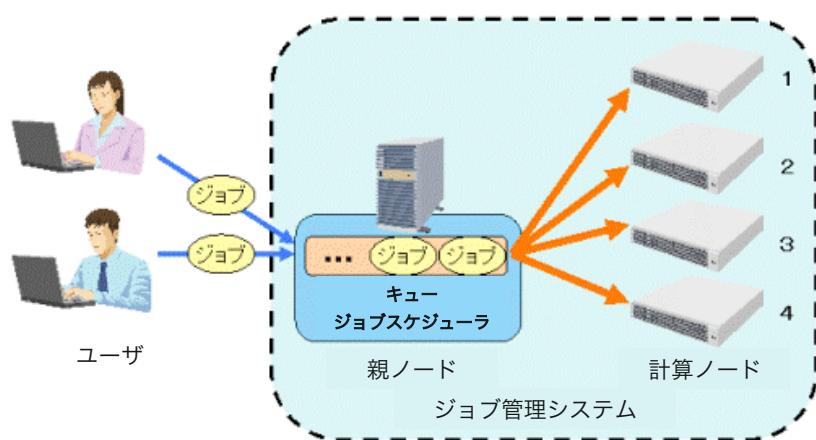
生物情報解析システム

SunGridEngine 利用方法

大型計算機を有効に使うには

複数の人間が同じ計算機群を使いたい...
どの計算機/CPUが空いてるか?
平等に使うには?

- ・ ユーザのジョブを
 - 実行された順番に
 - 空いている計算機に割り振ってくれる

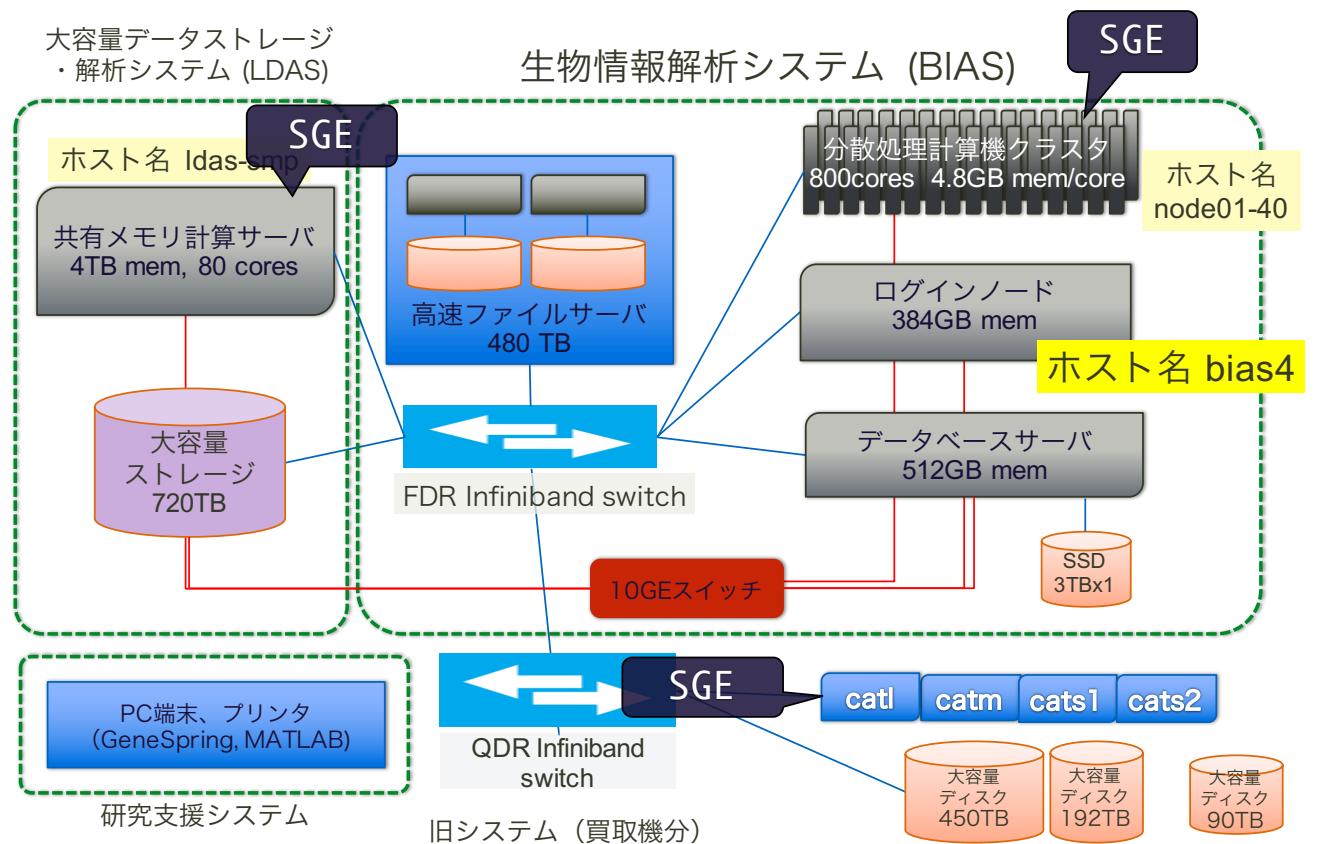


- ・ ジョブ管理システム

ジョブ管理システム

- ・親ノードが、複数ある計算機から資源の割り当てを自動で行い、効率を上げる
- ・ユーザは親ノードにジョブを投げるだけ（親ノードの名前すら知らないてもよい）
- ・生物情報解析システム上でのデータ解析は基本ジョブ管理システムを使うこと
 - bias4.nibb.ac.jp はパワーがないので、皆がbias4上で解析を行うとすぐ倒れます
- ・ **SunGridEngine (SGE)**

生物情報解析システムの構成



作業ディレクトリ

bias4.nibb.ac.jp 上での ~/6_sge

```
$ cd ~/6_sge  
$ ls
```

ジョブ管理システム:SGEの利用

- ・ 実行したいコマンドをシェルスクリプト内に記述し、**qsub**コマンド（後述）を用いてジョブ管理システムに実行させる
- ・ シェルスクリプト ex.sh の中身を確認

```
$ less ex.sh
```

```
#!/bin/sh  
#$ -cwd  
#$ -q small  
  
bowtie2 -x ecoli_genome -U ecoli.1.fastq -S ecoli.1.sam
```

qsub 実行！

- ・ シェルスクリプトをジョブ管理システム(SGE)に投入 : qsub

```
$ qsub scriptfile
```

- ・ ex.sh を qsub コマンドで実行

```
$ qsub ex.sh
```

```
Your job 814953 ("ex.sh") has been submitted
```

- ・ 投入されたジョブはログインノードから分散処理計算機クラスタ内のノードに送られて実行される
- ・ 標準出力と標準エラー出力のログが ホームディレクトリ にファイルとして作られる
- ・ 投入したジョブの状況を見るには qstat コマンドを使う

```
$ qstat
```

qsub (bowtie2) 結果の確認

```
$ ls
```

結果ファイルの確認

```
$ ls ex.sh.*
```

標準出力・標準エラー出力ファイル

```
ex.sh.e814953 ex.sh.o814953
```

```
$ less ex.sh.e814953
```

標準エラー出力ファイルの中身を確認

```
330118 reads; of these:
```

```
  330118 (100.00%) were unpaired; of these:
```

```
    3364 (1.02%) aligned 0 times
```

```
    229054 (69.39%) aligned exactly 1 time
```

```
    97700 (29.60%) aligned >1 times
```

```
98.98% overall alignment rate
```

SGEの主なコマンド

qsub <i>script_file</i>	<i>script_file</i> ジョブを投入
qstat	自分のジョブの状態を表示
qstat -u '*'	全ユーザのジョブ状態を表示
qdel <i>job-ID</i>	<i>job-ID</i> のジョブを削除

```
$ qstat
```

```
job-ID  prior  name    user   state submit/start at  queue slots ja-task-ID
-----  
814953  0.00000 ex.sh    hiroyo  r    01/08/2015 14:14:54 small@node04  1  
814954  0.00000 job.sh   hiroyo  qw   01/08/2015 14:14:54 small@node05  1  
814955  0.00000 job.sh   hiroyo  qw   01/08/2015 14:14:54 small@node05  1
```

```
$ qdel 814953
```

```
hiroyo has deleted job 814953
```

qsub のオプション

- qsub には様々なオプションがあり、シェルスクリプト内で「#\$」に続けて書いておくことで機能を加えることができる

```
$ less ex.sh
```

```
#!/bin/sh
#$ -q small          small キューを指定
#$ -cwd              qsub したディレクトリに移動してジョブを実行
bowtie2 -x ecoli_genome -U ecoli.1.fastq -S ecoli.1.sam
```

- cwd を指定しているので、ファイルのパスを付けていない

qsub のオプション

オプション	説明
<code>#\$ -o filename</code>	標準出力の結果を指定したファイルに保存
<code>#\$ -e filename</code>	標準エラー出力の結果を指定したファイルに保存
<code>#\$ -q queue_name</code>	キューを指定してジョブを実行
<code>#\$ -cwd</code>	qsubした時のディレクトリに移動してジョブを実行
<code>#\$ -v 環境変数=値</code>	環境変数をジョブに渡す
<code>#\$ -N job_name</code>	ジョブ名を指定する
<code>#\$ -s shell_name</code>	ジョブスクリプトを指定したシェルで実行
<code>#\$ -a MMDDhhmm</code>	ジョブの開始日時を指定
<code>#\$ -l resource_name 値</code>	ジョブが使うリソース量を指定する
<code>#\$ -pe PE_name プロセス数</code>	並列ジョブを実行する場合の環境と並列数の指定
<code>#\$ -t 開始番号-終了番号</code>	アレイジョブを実行

キュー（ジョブの待ち行列）構成

	分散並列処理型			共有メモリ型			
	分散処理計算機クラスタ			共有メモリ型計算サーバ		cat群	
キュー名	small	medium	large	smpls	smpm	smpl	cat
ジョブの特徴	短時間・ 並列多	中規模	長時間	中メモリ	大メモリ	最大メモリ	denovoアセ ンブリ用
利用ノード	node01-40	node01-40	node01-40	ldas-smp	ldas-smp	ldas-smp	cat1, catm cat1, cat2
最大実行時間 /job	6時間	72時間	no limit	no limit	no limit	no limit	no limit
最大ジョブ数 /キュー	580	200	20	8	4	1	112
最大使用メモリ /ジョブ	4GB	4GB	4GB	500GB	1TB	4TB	512GB/1TB/ 96GB/96GB
利用できるPE	smp, mpi128, mpi256, make	smp, mpi128, mpi256, make	smp, mpi128, make	smp	smp	smp	smp, make

- ・キューを指定しない場合、デフォルトでは「small」で実行される
- ・ユーザあたりジョブ同時実行数は最大 400

for文を使って並列化



bowtie2



node-01



bowtie2



node-01



bowtie2



node-14



node-01~node40
分散処理計算機クラスタ

- 1台で順次実行していくは時間がかかる計算も、分割して複数台、複数CPUに仕事をさせれば数倍の速度で終わる

for文を使って並列化：実際の例

```
$ less ex2.sh
```

```
#!/bin/sh
#$ -cwd

num=${1}
bowtie2 -x ecoli_genome -U ecoli.${num}.fastq
-S ecoli.${num}.sam
```

- qsub に渡すシェルスクリプト

for文を使って並列化：例2

```
$ less ex3.sh
```

```
#!/bin/sh
for i in 1 2 3
do
    qsub ex2.sh ${i}
done
```

- qsubを実行するためのシェルスクリプト
- for文を使って以下のコマンドを順番にqsubしている
 - ex2.sh 1
 - ex2.sh 2
 - ex2.sh 3

- 実行

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

for文を使って並列化：例3

- 投入したジョブを確認
- 3つのbowtie2が異なるマシンで実行されている

```
$ qstat
```

7769444	0.50500	ex3.sh	hiroyo	r	02/16/2016 10:19:38	small@node26	1
7769445	0.50500	ex3.sh	hiroyo	r	02/16/2016 10:19:38	small@node11	1
7769446	0.50500	ex3.sh	hiroyo	r	02/16/2016 10:19:38	small@node19	1

主なUNIXコマンド早見表

コマンド	由来等	説明	実行例
ls	LiSt	ディレクトリの内容をリスト表示	ls directory
pwd	Print Working Directory	現在のディレクトリを確認	pwd
cd	Change Directory	ディレクトリを移動	cd directory
cp	CoPy	ファイルコピー	cp file1 file2
mv	MoVe	ファイル移動	mv file1 file2
rm	ReMove	ファイル削除	rm file
chmod	Change MODE	ファイルの権限を変更	chmod u+x file
ln -s	LiNk	シンボリックリンクを作成	ln -s file directory
cat	ConcATinate	ファイルの内容を一括表示	cat file
head	HEAD	ファイルの先頭10行を表示	head file
tail	TAIL	ファイルの末尾10行を表示	tail file
less	antonym of more (*1)	ファイルの内容を表示	less file
wc	Word Count	ファイルの行数・単語数をカウント	wc file
grep	Global Regular Expression Print	ファイル内のパターン検索	grep pattern file
mkdir	MaKe DIRectory	ディレクトリを作成	mkdir directory
rmdir	ReMove DIRectory	ディレクトリを削除	rmdir directory
man	MANual	コマンドのマニュアルを表示	man command
alias	ALIAS	コマンドの別名を設定	alias name='command'
echo	ECHO	引数の文字列を表示	echo string
jobs	JOBS	実行中のジョブをID付きで表示	jobs
fg	ForeGround	ジョブをフォアグラウンドで実行	fg %jobid
bg	BackGround	ジョブをバックグラウンドへ移行	bg %jobid
ps	ProceSs	プロセスを表示	ps
kill	KILL	プロセスを停止	kill process_id
top	TOP	マシンの稼働状況をリアルタイム表示	top
ssh	Secure SHell	暗号化された通信経路で別のマシンにログイン	ssh username@hostname
scp	Secure CoPy	暗号化された通信経路で別のマシンにファイルをコピー	scp file1 username@hostname:file2
sftp/ftp	(Secure) File Transfer Protocol	別のマシンにファイルを転送	ftp hostname
curl	Client for URLs	ウェブサーバからファイルを取得	curl url
rsync	Remote Sync	ディレクトリ間の同期	rsync directory directory
gzip		ファイルの圧縮および解凍	gzip file / gzip -d file.gz
bzip2		ファイルの圧縮および解凍	bzip2 file / bzip2 -d file.bz2
tar	Tape ARchive	ファイルをまとめる	tar cvf file.tar / tar xvf file.tar
vi	VIvisual editor	テキストエディタ(vi)を起動	vi

*1 「more」というファイル表示コマンドも存在する。moreにはできない逆スクロールができるページャとして「less」と名付けられた。圧縮ファイルの表示なども可。