

# UNIX基本コマンド

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2020春（準備編）

UNIX・R・NGSの基礎

西出 浩世 (hiroyo@nibb.ac.jp, @piroyon)

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続実行
- 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	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などからsshを使用する
	WSL Windows Subsystem for Linux	Windows10から搭載されたLinuxを実行するための互換レイヤー WSL2(Windows Terminal)からは完全なLinuxカーネルを組み込める。要インストール。

#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メニューを確認
2. 「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

- 画面に表示されている "\$" 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

- 半角英数字および記号のみ

- コマンドの入力は全て半角文字を使用

- 入力文字が全角になる日本語入力はOFFにする

## 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

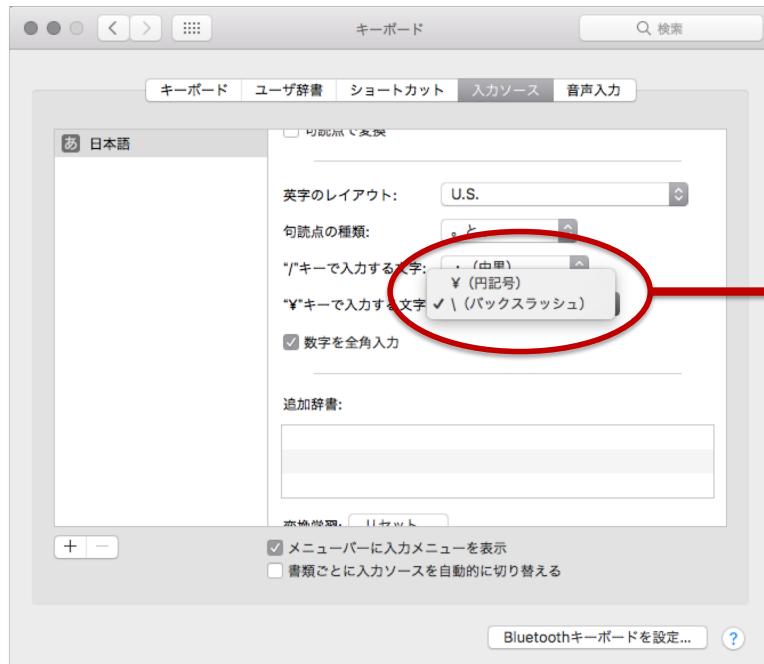
- キーの位置を確認しましょう

" \ " バックスラッシュ(¥キー)  
" | " 縦棒 バーティカルバー



# Macにおけるバックスラッシュ＼の入力

- システム環境設定 - キーボード - 入力ソース
- "¥"キーで入力する文字



¥(円記号)か  
＼(バックスラッシュ)  
かの切替え

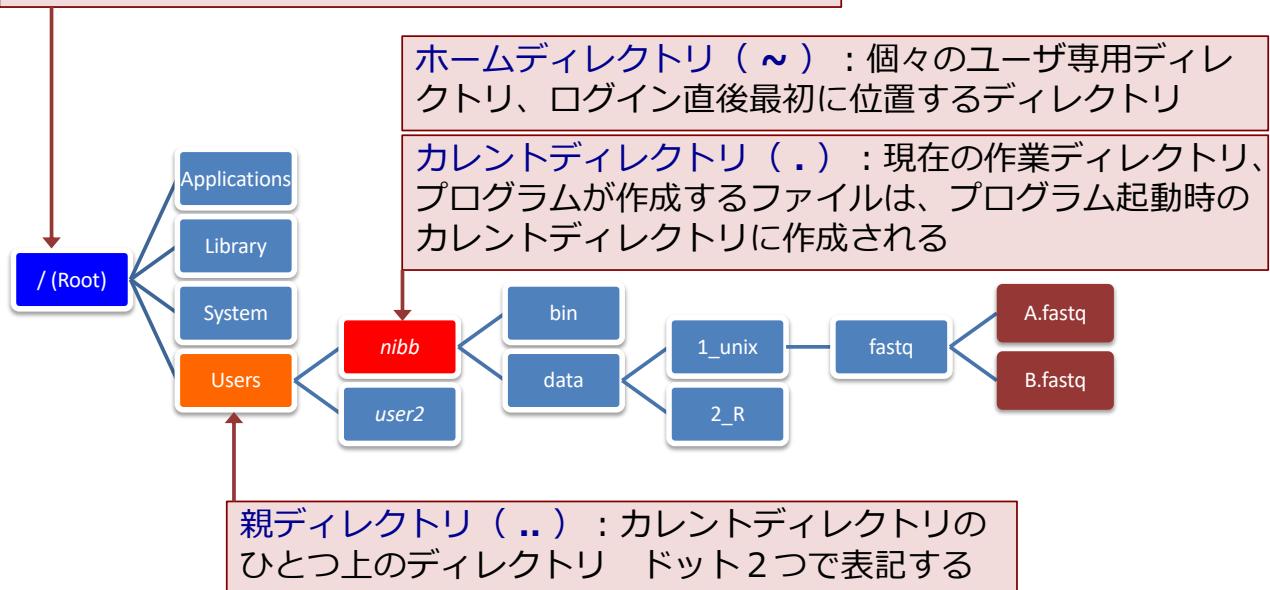


# ファイルシステム

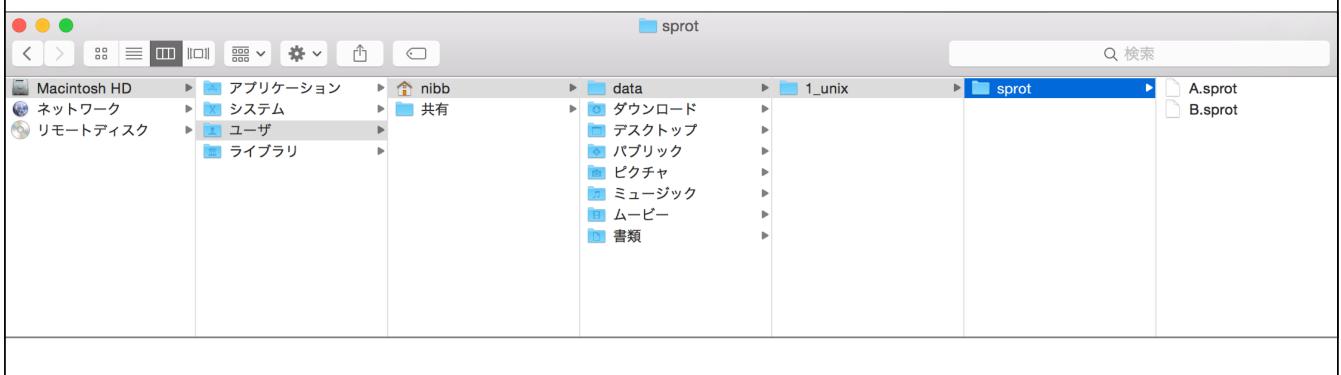
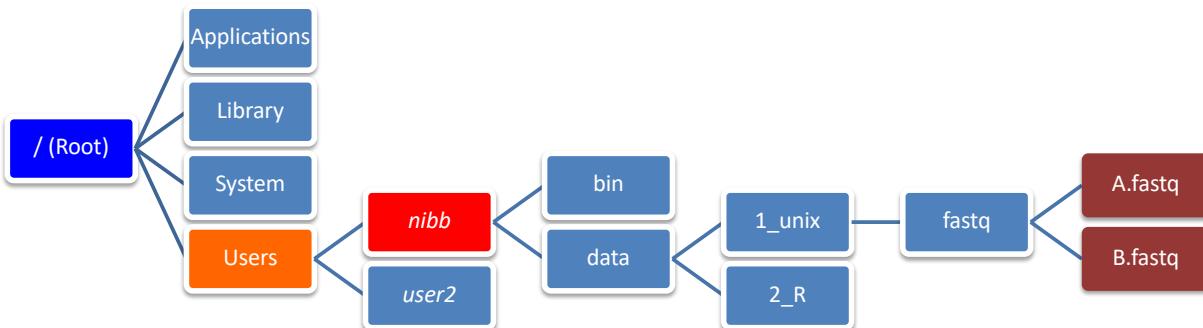
## 階層型ディレクトリ

- UNIXにおけるディレクトリ = PCでのフォルダ
- 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



## 作業ディレクトリ

- 本講習では次のディレクトリを使用します

**~/data/1\_unix**

<b>1_unix</b>	
└── bin	シェルスクリプト用
└── ecoli.gff3	大腸菌遺伝子情報
└── ecoli_genome.fasta	大腸菌ゲノム配列
└── ecoli_protein.fasta	大腸菌遺伝子アミノ酸配列
└── fastq	NGSマッピング結果
└── MG_mpos_1.fastq	
└── MG_mpos_2.fastq	
└── MG_urine_1.fastq.....	
└── exercise	演習問題用ファイル
└── 1433B_HUMAN.fasta	
└── 1433B_MOUSE.fasta	
└── 1433E_HUMAN.fasta.....	

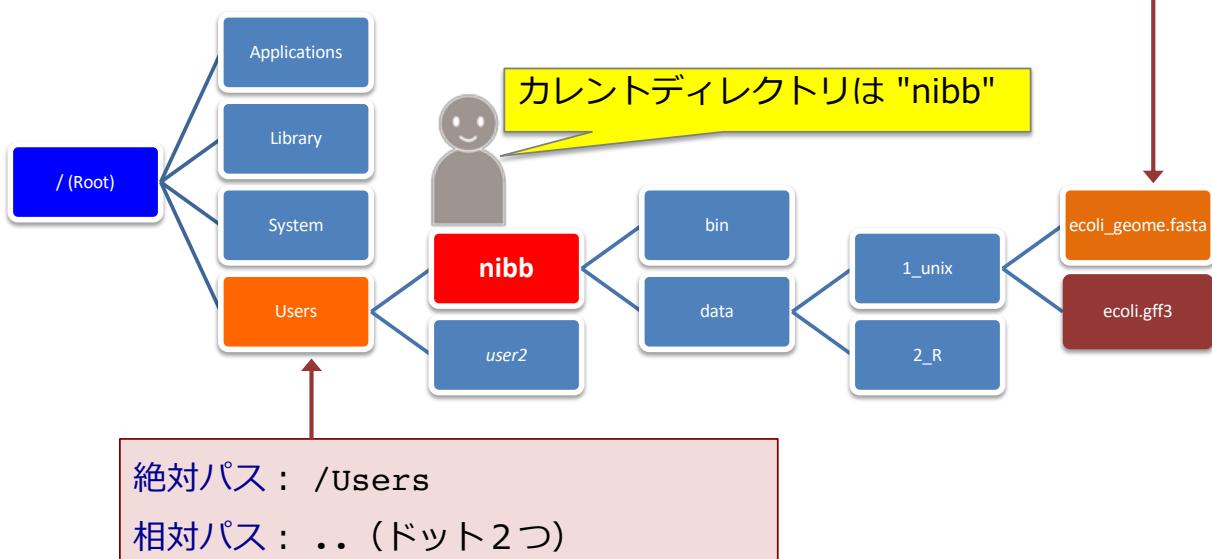
# ファイル/ディレクトリの指定方法

- 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/ecoli_genome.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 (ルート) "/" 以外で始まる
    - 例) `data/1_unix/ecoli_genome.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

ユーザ : nibb がログイン後 :  
   ecoli\_geome.fa ファイルと Users ディレクトリへのパス表記

絶対パス : /Users/nibb/data/1\_unix/ecoli\_genome.fasta  
 相対パス : data/1\_unix/ecoli\_genome.fasta



# 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
.bash_profile
```

ログイン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、tabキーを押す  
"ls da" と入力して、tab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/fa" 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）tabキーを押すと、候補ファイルの一覧を表示する  
"ls data/1\_unix/fastq/MG\_." の状態で2回 tabキーを押す
- 以下のメリットがあるので積極的に活用しましょう
  - 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

1. ホームディレクトリ上で **ls da** と入力後、**<tab>** (tabキー) を押してファイル名の補完機能を試してみる。
2. 続けて **ls data/1\_unix/fastq/MG\_** 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/fa <tab>
$ ls data/1_unix/fastq/ <tab>
$ ls data/1_unix/fastq/MG_ <tab><tab>
.... (候補一覧表示)
```

# 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>s</u> t <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

- 「コントロールキーとPを同時に押す」を何度か繰り返し過去のコマンドを表示する。

\$ **Control + P**

- 「コントロールキーとNを同時に押す」を何度か繰り返す

\$ **Control + N** を何度か入力する。

- 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. `fastq` 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 fastq
```

```
$ pwd
```

```
$ ls
```

# 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 *.fastq
```

```
$ ls *mpos*
```
  - クエスチョン (?) : 任意の1文字
 

```
$ ls MG_mpos_?.fastq
```
  - [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 MG_mpos_[1-3]*.fastq
```
  - {文字列1, 文字列2} : "文字列1" または "文字列2"
 

```
$ ls MG_{mpos,urine}_1.fastq
```

# 実習6

- ワイルドカード

(カレントディレクトリは ~/data/1\_unix/fastq)

1. 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 *.fastq
$ ls *mpos*
$ ls MG_mpos_[1-3].fastq
$ ls MG_{mpos,urine}_1.fastq
$ 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 MG_mpos_1.fastq
```

- ファイル名を複数指定すると内容を連結して表示

```
$ cat MG_mpos_*
```

## 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/fastq)

1. 下記コマンドを実行する。

```
$ cat MG_mpos_1.fastq
```

```
$ cat MG_mpos_*
```

# 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 MG_mpos_1.fastq
```

- 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 MG_mpos_1.fastq
```

- "-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 ecoli_genome.fasta
```

(FASTAファイルの配列のみ表示、配列名は表示しない)

# 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/fastq)

1. 下記コマンドを実行する。

```
$ head MG_mpos_1.fastq
```

```
$ tail -20 MG_mpos_1.fastq
```

2. 一つ上のディレクトリに移動して実行

```
$ cd ..
```

```
$ head ecoli_genome.fasta
```

```
$ tail -n +2 ecoli_genome.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/1\_unix)

1. ecoli.gff3 の内容を less コマンドで見る

**\$ less ecoli.gff3**

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. 文字列 "hemolysin" を検索する。

**/hemolysin**

5. 次にヒットする場所に移動する (next)

**n**

6. lessを終了する (quit)

**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)

1. 実習用のディレクトリ"unixtest" を作成する。

```
$ 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。確認したら1つ上のディレクトリ（ドット2個）に戻る

```
$ ls
```

```
$ cd unixtest
```

```
$ pwd
```

```
$ cd ..
```

# ファイル/ディレクトリのコピー (cp)

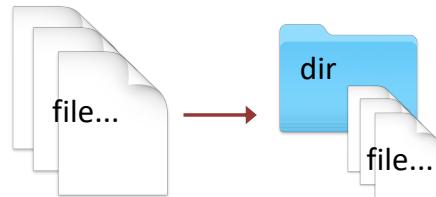
- **cp file1 file2**

- file1のコピーをfile2として作成



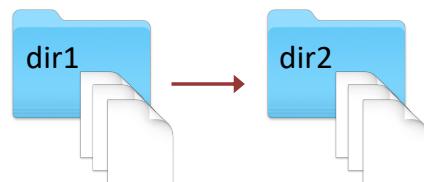
- **cp file1 [file2...] dir**

- 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 ~/data/1\_unix)

1. ecoli\_genome.fasta ファイルを copy.fasta という名前で同じディレクトリ内にコピーする。

```
$ cp ecoli_genome.fasta copy.fasta
```

```
$ ls
```

2. コピーした copy.fasta を同じ名前で unixtest ディレクトリ内にコピーする。

```
$ cp copy.fasta unixtest
```

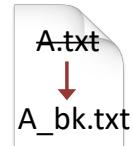
3. コピーされているか確認する。

```
$ ls unixtest
```

## ファイル/ディレクトリ名の変更、移動 (`mv`)

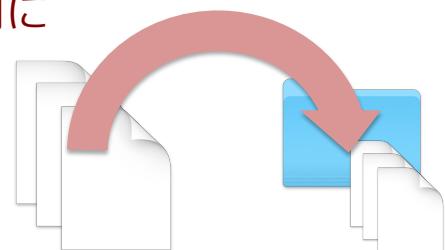
- `mv file1 file2`

- `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 `~/data/1_unix`)

1. `copy.fasta` の名前を `new.fasta` に変更する。

```
$ mv copy.fasta new.fasta
```

2. `new.fasta` を `unixtest` 内に移動する。

```
$ mv new.fasta unixtest
```

3. 移動したか確認

```
$ ls
```

```
$ ls unixtest
```

# 実習13

- cp と mv を利用してファイルを分ける

(カレントディレクトリは ~/data/1\_unix)

1. fastq ディレクトリを unixtest ディレクトリ内に同じ名前でコピーする。

```
$ cp -r fastq unixtest
```

2. unixtest に移動して、中を確認する

```
$ cd unixtest
```

```
$ ls
```

3. ディレクトリを2つ、 MG および URINE という名前で作成する。

```
$ mkdir MG
```

```
$ mkdir URINE
```

4. fastq 下の MG で始まるファイルだけを、 3.で作成した MG ディレクトリにワイルドカードを使用してコピーする。コピーが終わったら確認する。

```
$ cp fastq/MG* MG
```

```
$ ls MG
```

5. MG ディレクトリ内にあるファイルのうち、ファイル名に "urine" を含むもののみを URINE ディレクトリ内に移動する。移動が終わったら確認する

```
$ ls MG/*urine*
```

```
$ mv MG/*urine* URINE
```

```
$ ls URINE
```

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



Memo



# 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orディレクトリの「別名」を作成
- MacOSのエイリアスやWindowsのショートカットに相当

```
$ ln -s ~/data/1_unix/ecoli_genome.fasta .
```

~/data/1\_unix/ecoli\_genome.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

- 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/1_unix/fastq ~
```

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14

- シンボリックリンクの作成

(カレントディレクトリは ~/data/1\_unix/unixtest)

1. ~/data/1\_unix/ecoli\_genome.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 ~/data/1_unix/ecoli_genome.fasta .
```

または

```
$ ln -s ../ecoli_genome.fasta .
```

```
$ ls
```

2. 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 ecoli_genome.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

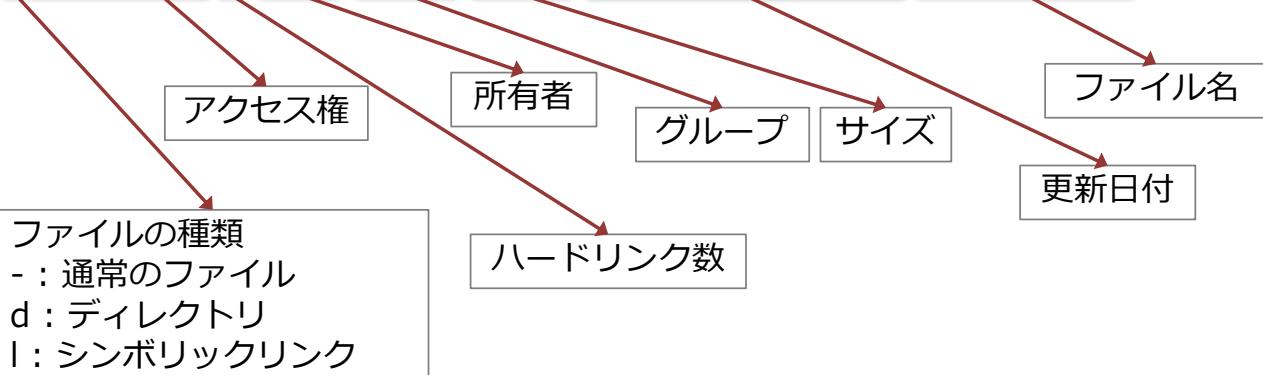
- ファイルの大きさや更新日などの情報をリスト表示

- `ls -lt` ファイル/ディレクトリ名

- 日付順に表示

```
$ ls -l ecoli.gff3
```

```
-rw-r--r-- 1 nibb staff 21675 2019-03-09 09:23 ecoli.gff3
```



## 実習15

- ファイル情報の表示

(カレントディレクトリは `~/data/1_unix/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

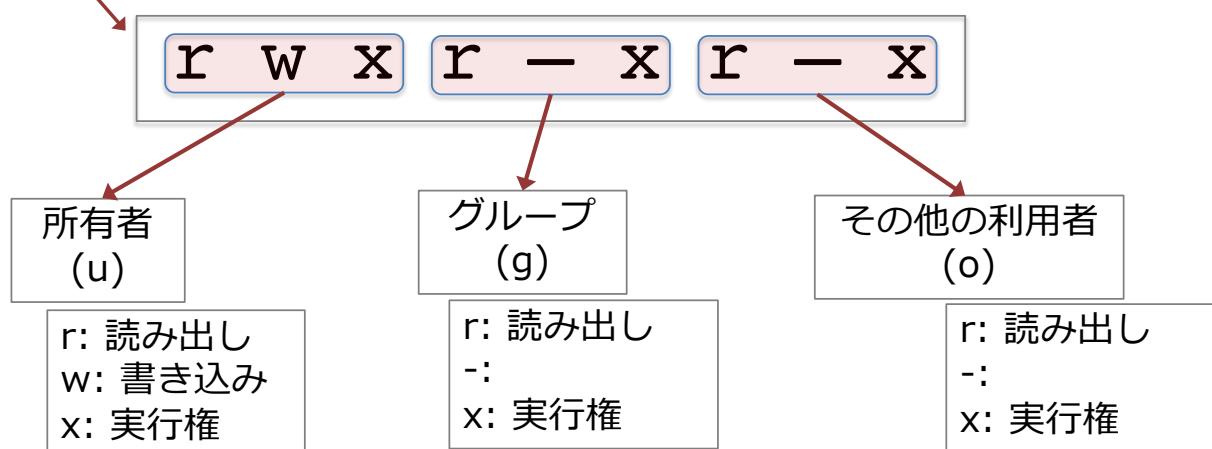
- 全てのファイル／ディレクトリに付けられている
  - MacOSやWindowsでは気にしないが、UNIXでは意外と重要
- 誰に対して、何ができるか
  - 誰に対して：
    - (u) ファイルの所有者
    - (g) 同一グループの利用者
    - (o) その他
  - 何ができるか：
    - 「読む(r)」「書く(w)」「実行する(x)」
  - 「実行する(x)」
    - ファイルをプログラムとして実行できる
    - ディレクトリの場合は「移動が可能」「そのディレクトリへアクセスできる」

## 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 2017-03-09 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16

- アクセス権の確認と変更

(カレントディレクトリは ~/data/1\_unix/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. new.fasta のアクセス権から 自身のRead権限を削除する  
(通常はこのようなことはしませんが)

\$ **chmod u-r new.fasta**

3. lessで new.fasta を読むとエラーになることを確認

\$ **less new.fasta**

4. 再度 new.fasta のアクセス権に自身のRead権限を追加する

\$ **chmod u+r new.fasta**

5. less でエラーなく new.fasta が読めることを確認する

\$ **less new.fasta**

# 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
- UNIXに「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17

- ファイルの削除
 

(カレントディレクトリは ~/data/1\_unix/unixtest)

  1. **new.fasta** を削除する  
**\$ rm new.fasta**
  2. MG ディレクトリ全体を削除する  
**\$ rm -rf MG**

# 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 s 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.
```

- ページ送り、終了の動きは **less** と同じ
- **man -k keyword** で keyword に関するコマンド一覧を表示

## 実習18

- マニュアルの参照

(カレントディレクトリは ~/data/1\_unix/unixtest)

1. ls コマンドのマニュアルを表示する。

```
$ man ls
```

2. 終了

```
$ q
```

# 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 ecoli_genome.fasta
```

66283 66283 4705962 ecoli\_genome.fasta

(66,283行、66,283単語、4,705,962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19

- 行数・単語数のカウント

(カレントディレクトリは ~/data/1\_unix/unixtest)

1. 一つ上のディレクトリに移動する。

```
$ cd ..
```

```
$ pwd
```

2. ecoli\_genome.fasta の単語数をカウントする

```
$ wc ecoli_genome.fasta
```

# 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 rpoB ecoli.gff3
```

ecoli.gff3 から rpoB を含む行を検索

```
$ grep ^@SRR fastq/MG_mpos_1.fastq
```

fastq ディレクトリ内にある MG\_mpos\_1.fastq  
から 「@SRR」で始まる行を検索

("^" は行の先頭を意味する)

- 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20

- パターン検索

(カレントディレクトリは ~/data/1\_unix)

1. ecoli.gff3 の内容をlessで確認し、rpoBを検索する

```
$ less ecoli.gff3
```

2. ecoli.gff3 から "rpoB" が含まれる行を検索して表示する

```
$ grep rpoB ecoli.gff3
```

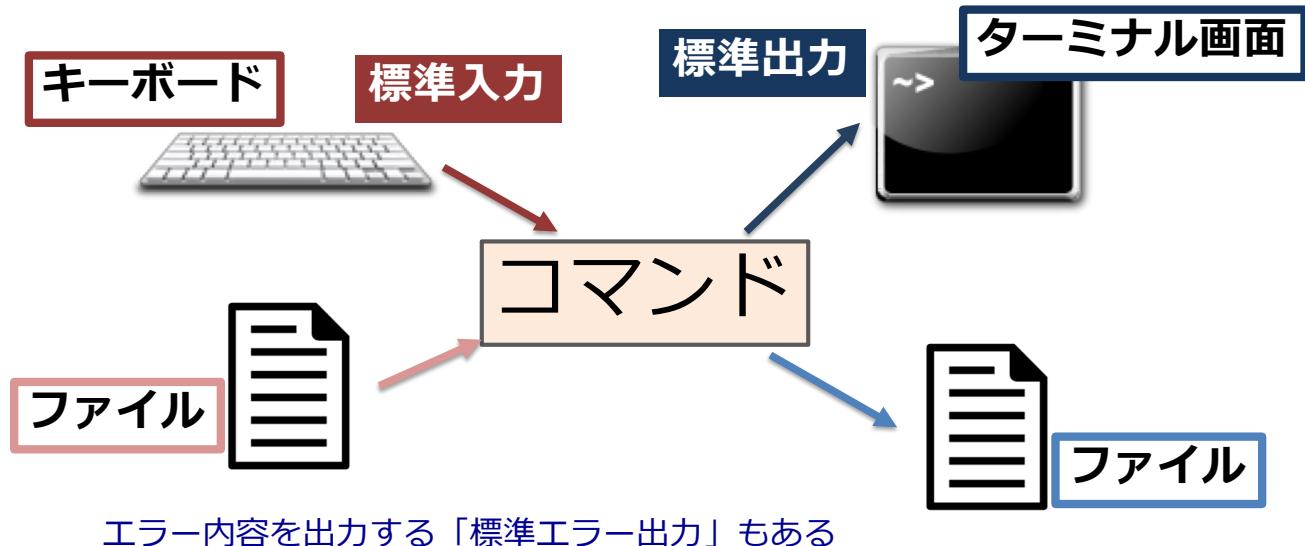
3. fastq ディレクトリ内にある MG\_mpos\_1.fastq から  
「@SRR」で始まる行を検索して表示する

```
$ grep ^@SRR fastq/MG_mpos_1.fastq
```

# 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 ^@SRR MG_mpos_1.fastq
```

- ファイルに保存 (>)

- コマンドの右側に " > filename" を加える
- 結果を fastq\_list というファイルに保存

```
$ grep ^@SRR MG_mpos_1.fastq > fastq_list
(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 (>>)

- コマンドの右側に " >> filename" を加える

```
$ grep ^@SRR MG_mpos_2.fastq >> fastq_list
(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 < fastq_list
(wc fastq_list と同じ)
```

## 実習21

- リダイレクト

(カレントディレクトリは ~/data/1\_unix)

1. fastq ディレクトリ内にある MG\_mpos\_1.fastq から「@SRR」で始まる行を表示して、その結果をリダイレクト（出力）を使用して fastq\_list ファイルに保存

```
$ grep ^@SRR fastq/MG_mpos_1.fastq
(ファイル保存前に確認)
```

```
$ grep ^@SRR fastq/MG_mpos_1.fastq >
fastq_list
```

実際は1行で打つこと！

2. ファイルの中身を less で確認する

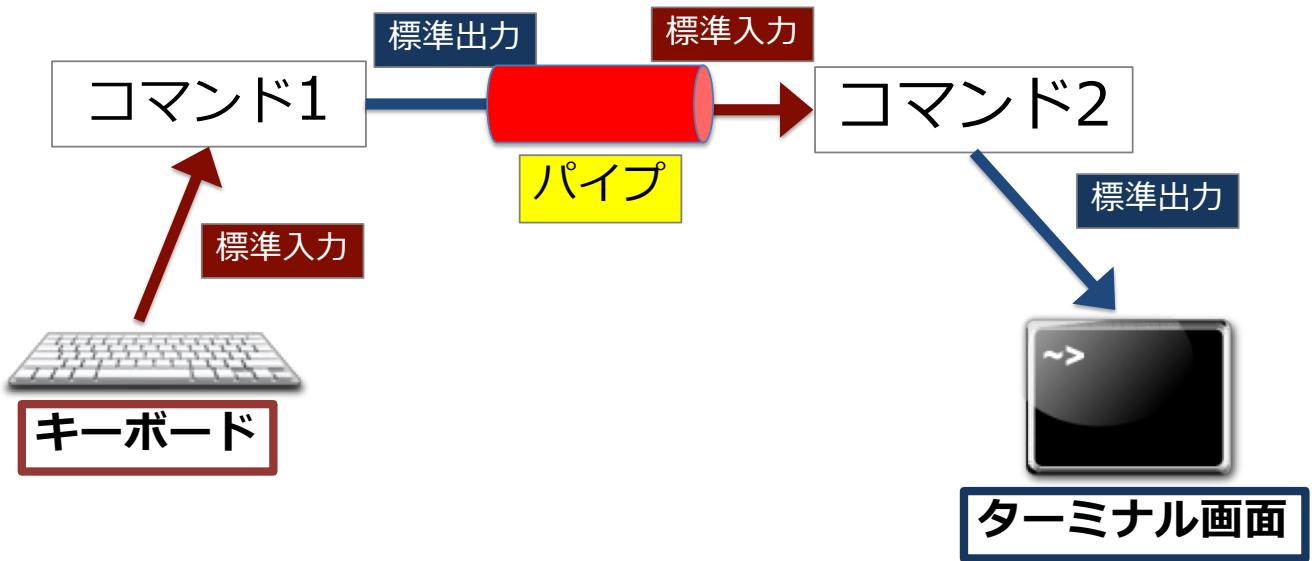
```
$ less fastq_list
```

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 < fastq_list
```

# 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 ^@SRR MG_mpos_1.fastq | wc
```

- grep の結果を less で見る

```
$ grep rpoB ecoli.gff3 | less
```

- grepの結果をさらにgrepして絞り込む

```
$ grep tRNA ecoli.gff3 | grep thr | less
```

- コマンドはいくつでも結合できる

# 実習22

- パイプ

(カレントディレクトリは ~/data/1\_unix)

1. fastq ディレクトリ内にある MG\_mpos\_1.fastq から「@SRR」で始まる行を検索し、結果をパイプを使用してwcで行数・単語数をカウントする。

```
$ grep ^@SRR fastq/MG_mpos_1.fastq | wc
```

2. ecoli.gff3 から "rpoB" が含まれ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 rpoB ecoli.gff3 | less
```

3. ecoli.gff3 から "tRNA" が含まれる行を検索した結果から、更に"thr"を検索して less で表示する。

```
$ grep tRNA ecoli.gff3 | grep thr | less
```

# シェルスクリプト

- 一連のコマンドを実行順に記述したファイル

- 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
```

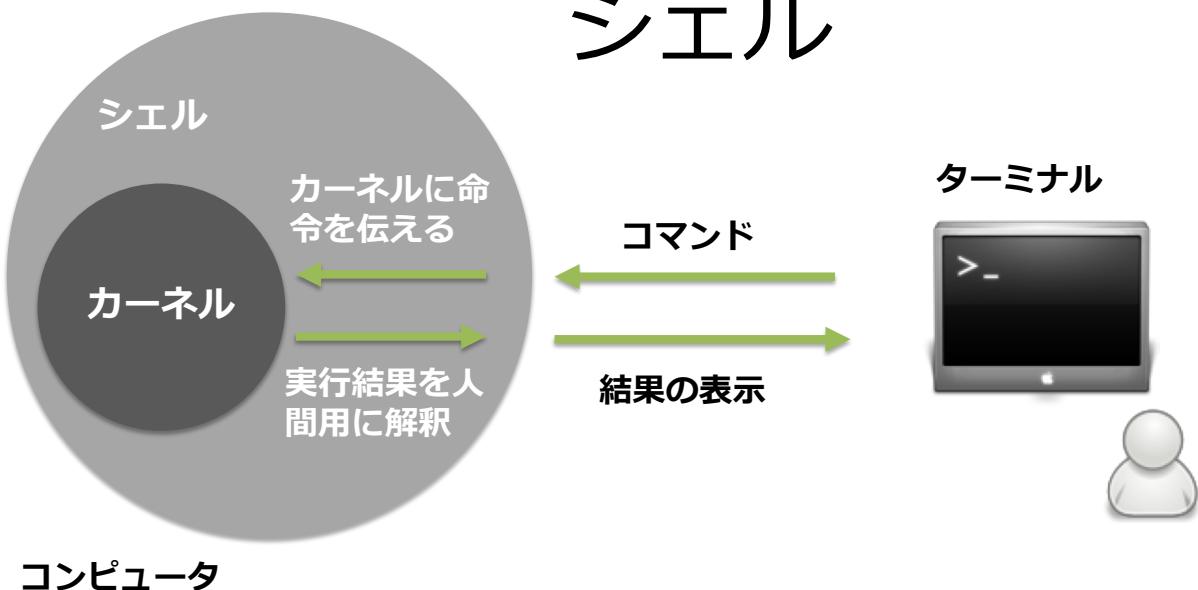
- 実行するときはパスを省略せずに入力

```
$ ./testpg
```

もしくは /Users/nibb/data/1\_unix/bin/testpg

- コマンドサーチパス（後述）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

# シェル



コンピュータ

- シェルによって、UNIXコンピュータとの対話をする
- シェルにも種類がある (bash系、csh系)

## 実習23

### ● シエルスクリプト

(カレントディレクトリは ~/data/1\_unix)

1. bin/testpgをカレントディレクトリにコピーする。

```
$ cp bin/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
  - パスを省略しても動くコマンドは、「コマンドサーチパス」に登録されたディレクトリのリストから探し出される
    - パスなし → "ls"、パスあり → "/bin/ls"
    - 例) ls コマンドのパスは /bin/ls である
- 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" ファイルで設定する
  - 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、最初のパスが優先される
- 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
  - (ログアウト、ターミナルを終了したら無効)
  - PATH変数に新しいコマンドサーチパスを追加する
 

**PATH=\$PATH:~/bin**

(変数を見るときは\$を付けて、変数へ代入するときは付けない)

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24

## ● コマンドサーチパス

(カレントディレクトリは ~/data/1\_unix)

- 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
```

- コマンドサーチパスに登録されているディレクトリの中身を確認する

```
$ ls /bin
```

- ~/binがコマンドサーチパスに登録されていることを確認する

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

自分で作成したコマンドを~/binディレクトリへコピーすることによって、パスの記述なしでコマンドを実行できる

- 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も動作するかを確認する。もし~/binディレクトリが存在しない場合は作成すること

```
$ testpg # エラーになる
```

```
$ cp testpg ~/bin
```

```
$ testpg
```

# コマンドの在処を探す (which)

## ● which コマンド名

- grep がある場所を探す

```
$ which grep
```

```
/usr/bin/grep
```

探したいコマンドが「コマンドサーチパス」のリストにあるディレクトリ内にあれば探すことができる

- which で見つかれば、パスなしで実行できる
- プログラムが正しくインストールされている

# 実習25

- コマンドを探す

(カレントディレクトリは ~/data/1\_unix)

1. grep がある場所を探す

```
$ which grep
```

# 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`
- 設定済みエイリアスの一覧表示  
`$ alias`

# 実習26

- エイリアス

(カレントディレクトリは ~/data/1\_unix)

1. 現在設定されているエイリアスを確認する

```
$ alias
```

2. 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
```

## .bash\_profile

- シェルの動作を変更する設定ファイル

- ユーザの環境変数やエイリアスの設定を記述する。
- .bash\_profileはログイン時に一度だけ実行される。
- 隠しファイルで通常はlsで表示されない。

- 環境変数とシェル変数

- 環境変数はシェルから実行したコマンドにも引き継がれる  
設定例： \$ export HOGE=hogehoge
- シェル変数は現在実行中のシェルだけに有効な変数  
設定例： \$ HOGE=hogehoge

# 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
  - 下記の誤り例ではリダイレクトによりファイルを上書きする  
(誤) `$ grep ^> ecoli_protein.fasta`  
(正) `$ grep '^>' ecoli_protein.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27

### ● メタキャラクタ

(カレントディレクトリは ~/data/1\_unix)

1. `unixtest` に移動する

```
$ cd unixtest
```

2. 一つ上のディレクトリにある `ecoli_protein.fasta` をカレントディレクトリにコピーする

```
$ cp ../ecoli_protein.fasta .
```

3. `ecoli_protein.fasta` から配列名 (">" で始まる行) を検索する  
`$ grep '^>' ecoli_protein.fasta`

4. 同じコマンドの「'」がない状態で実行する

```
$ grep ^> ecoli_protein.fasta
```

5. 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 `ecoli_protein.fasta` ファイルにリダイレクト出力 (>) するという意味になる。

# 実行したコマンドの中止と中断

- 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**
- 実行中のジョブを確認
  - jobs コマンド

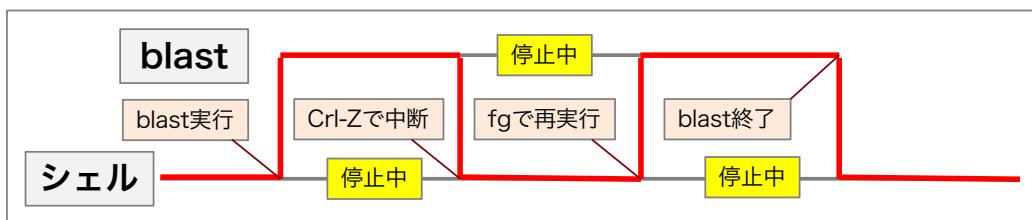
\$ **jobs**

```
[1]- Running yes > /dev/null &
[2]+ Stopped yes > /dev/null
```
- 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）

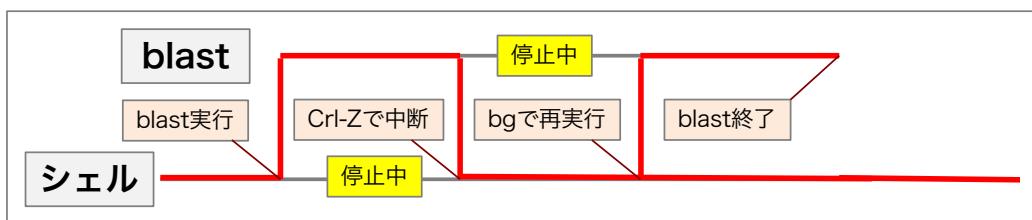
はじめからバックグラウンドでジョブを実行するには、コマンド行末尾に "&"

# 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：プロセス
- シェルが管理する単位：ジョブ



- 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" を押す

# 実習28

## ● プロセスの確認

(カレントディレクトリは ~/data/1\_unix/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
```

# ファイル転送(ftp)

## ● ftp リモートホスト名

- 対話形式によるファイル転送
- 公開FTPサーバ(Anonymous FTP)利用の慣行

ユーザ名:anonymous, パスワード:メールアドレス

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
<b>-o file url</b>	urlのデータをfileに保存
<b>-O</b>	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)
<b>-O url[start-end]</b>	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
<b>-h</b>	ヘルプを表示

# 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**

**gzip -d ファイル名.gz**

- **bzip2 (拡張子 : .bz2)**

圧縮 : **bzip2 ファイル名**

解凍 : **bunzip2 ファイル名.bz2**

**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**ディレクトリを **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・ファイルを作成する
  - **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 ~/data/1\_unix/unixtest)

1. unixtest にある **fastq** ディレクトリの圧縮アーカイブをls、**fastq.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。

```
$ tar zcvf fastq.tar.gz fastq
$ ls
$ \rm -rf ./fastq
```

2. **fastq.tar.gz** の内容を確認する

```
$ tar ztvf fastq.tar.gz
```

3. **fastq.tar.gz** をカレントディレクトリ内に解凍展開する

```
$ tar zxvf fastq.tar.gz
$ ls
```

# プログラムソースからのインストール

- プログラムソース一式の取得
  - 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は限らないので、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
 

<b>\$ ./configure</b>	環境にあったコンパイルの設定を行う
<b>\$ make</b>	プログラムをコンパイル
<b>\$ make install</b>	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

- SAMtools1.10のインストール
 

(カレントディレクトリは ~/data/1\_unix/unixtest)

  - SAMtoolsのダウンロードと解凍
    1. Safariを開いて <http://www.htslib.org> にアクセス
    2. Download → Sourceforge
    3. 1.10 → samtools-1.10.tar.bz2
    4. ~/Downloads にbzip圧縮アーカイブとして保存される

```
$ mv ~/Downloads/samtools-1.10.tar.bz2 .
$ ls
$ tar jxvf samtools-1.10.tar.bz2
$ cd samtools-1.10
$ ls
```

# 実習30 続き

- コンパイルとインストール

```
$ less INSTALL (INSTALLを読んでインストール方法を確認する)
```

Basic Installation のところを見る)

```
$ ./configure --prefix=/Users/nibb
```

(configure というスクリプトファイルを実行)

```
$ make
```

```
$ make install
```

- ✓ ダウンロードするだけで使える「バイナリファイル」が提供されている場合はそれを使う方がよい。自分が使っているOSに合ったバイナリファイルかどうか確認すること。
- ✓ yum, apt, brew など、自動でインストールが行われるコマンドに対応したインストール方法が提供されている場合もある。

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 