

第1講(13:00-13:50)

CUI/GUIとは

- UI(User Interface): 人がコンピュータを操作する/コンピュータが人に情報を見せる仕組み
- GUI: Graphical UI, デスクトップ上に表示されているボタンやアイコンをマウスでクリックして操作するイメージ
- CUI: Character UI, ターミナル(端末)と呼ばれるソフトウェア上でキーボードからcommandを入力して操作

command 書式

```
$ command [option] [引数]
```

- 行頭の"\$"はプロンプトといい、コマンド入力可能であることを表示している。
- []は省略可能
- 行頭, 行末およびcommand, option, 引数の間にはいくつ空白を置いてもよい。
- option: 機能切替え
- 引数: commandの処理対象

第2講(14:00-14:50): FILE/DIRECTORYの操作

- file・directoryは階層構造で配置されている
- 特定のfile・directoryの位置は絶対パス又は 相対パスで指定する
- 相対パスではcurrent directory (=現在の作業位置)を起点とする
- login直後のcurrent directoryはuserごとのhome directoryである

pwdコマンド

Print Working Directoryの意味. current directoryの確認に使用.

```
$ pwd
```

出力例

```
/home/tom
```

mkdirコマンド

make directoryの意味. 指定名のdirectoryを作る.

```
$ mkdir HIGO          # "HIGO"という名前のdirectoryを/home/tom上に作製 (/home/tom/HIGO). これは相対パス指定の例
                        # mkdir /home/tom/HIGO というように絶対パス指定でもOK
                        # mkdir ~/HIGO とも書ける, "~"はhome directoryに置換される
$ mkdir HIGO/class2    # 因みに"#"以下は行末までコメント(コマンドの一部でない)として扱われる
                        # "HIGO" directory配下に"class2" directoryを作製
                        # mkdir -p HIGO/class2 という風に-pオプションを使えば
                        # HIGO directoryが存在しなくても両方のdirectoryを一度
                        # に作製することができる.
```

cdコマンド

Change Directoryの意味. Current directoryの移動に使用.

```
$ cd HIGO/class2
$ pwd
```

```
/home/tom/HIGO/class2
```

```
$ cd ..                # ".."は一つ上のdirectoryの意味
$ pwd
```

```
/home/tom/HIGO
```

```
$ cd                   # 引数が無指定の場合はhome directoryに移動
$ pwd
```

```
/home/tom
```

```
$ cd -          # 直前のcurrent directoryに移動する
$ pwd
```

```
/home/tom/HIG0
```

```
$ cd ~/HIG0/class2    # 当然、絶対パスでも指定可能
$ pwd
```

```
/home/tom/HIG0/class2
```

## lsコマンド

listの意味. あるdirectory内のfileやdirectoryの一覧を表示する.

```
$ ls ~          # 引数で指定したdirectory(=ここではhome directory)のfile/directoryの一覧表示

$ ls -a ~       # ~から始まるfile・directory名のもの(隠しファイル)も表示
$ ls -l ~       # 詳細表示
$ ls -al ~      # -a オプションと -l オプション の両方を指定した場合
                # ls -a -l ~ と書いても同じ
                # ls -l -a ~ と書いても同じ
                # ls -la ~ と書いても同じ
$ ls            # Current directoryのfile・directory一覧表示
```

## lessコマンド

file内容を表示する (別画面に遷移する/qとタイプすると終了)

```
$ less ~/.bashrc
```

## catコマンド

file内容を表示する

```
$ cat ~/.bashrc
```

file内容を連結(Catenation)表示する

```
$ echo aaa > file1      # "aaa"という内容のファイルfile1を作製
$ echo bbb > file2      # "bbb"という内容のファイルfile2を作製
$ cat file1 file2 > file3  # file1 file2の順でファイル内容を連結した
                        # file3を作製

$ cat file3
```

```
aaa
bbb
aaa
```

## cpコマンド

Copyの意味. file・directoryを複製する.

```
$ cp file3 file4      # file3をfile4という名前で複製
$ cat file4
```

```
aaa
bbb
aaa
```

```
$ cp -r ../class2_copy  # directoryを複製するときは-r オプションが必要
                        # "."はこのdirectoryの意味
$ ls ../class2_copy
```

```
file1 file2 file3 file4
```

## mvコマンド

moveの意味. fileを別のdirectory配下に移動.

```
$ cp file4 file5
$ ls
```

```
file1 file2 file3 file4 file5
```

```
$ mv file5 ../class2_copy
$ ls
```

```
file1 file2 file3 file4
```

```
$ ls ../class2_copy
```

```
file1 file2 file3 file4 file5
```

ファイル名を変更する

```
$ mv file4 file_4
$ ls
```

```
file1 file2 file3 file_4
```

```
$ mv file1 file_4      # 移動先をすでに存在するファイル名を指定した場合
$ cat file_4
```

```
aaa
```

rmコマンド

removeの意味. file・directoryの削除.

```
$ rm file_4
$ ls
```

```
file2 file3
```

```
$ rm -r ../class2_copy    # directoryを削除する際は-r オプションをつける.
$ ls ..
```

```
class2
```

---

## 第3講(15:00–15:50)

まずは準備

```
$ mkdir ~/HIGO/class3
$ cd ~/HIGO/class3
```

commandの入出力

- 標準入力: キーボード (default)
- 標準出力: 端末画面 (default)
- 標準エラー出力: 端末画面 (default)

入力元・出力先の切替え (=リダイレクト)

command実行結果の出力先を標準出力からFILEにリダイレクト

```
$ command > FILE
```

実行例

```
$ seq 5          # 実行例を示す前に
$ seq 1 5        # seqコマンドの挙動を確認
$ seq 5 10       #
$ seq -w 10      #
```

```
$ seq 3 > file
$ cat file
```

```
1
2
3
```

```
$ seq 4 6 > file
$ cat file
```

```
4
5
6
```

”>”だと元々のファイル内容は一旦空になることに注意.  
ファイル末尾にデータを新たに追加したい場合は追記リダイレクト”>>”を使う

```
$ seq 3 > file
$ seq 4 6 >> file
$ cat file
```

```
1
2
3
4
5
6
```

command実行時の標準エラー出力をFILEにリダイレクト

```
$ command 2> FILE      # 2と>の間に空白がないように
```

実行例

```
$ cat file1           # 例えば存在しないファイルfile1を指定した場合
```

```
ls: 'file1'にアクセスできません: そのようなファイルやディレクトリはありません
```

```
$ cat file1 2> file
# cat file
```

```
ls: 'file1'にアクセスできません: そのようなファイルやディレクトリはありません
```

入力元・出力先の切替えによるcommand同士の連携 (=パイプ)

```
$ command1 | command2    # command1の標準出力をcommand2の標準入力に接続
```

実行例

```
$ seq 3 > file
$ tac file                # tacコマンドはファイル内容を逆順に出力する.
```

```
3
2
1
```

```
$ seq 3 | tac
```

```
3
2
1
```

変数

```
$ original=atgc          # “atgc”という文字列を変数originalに代入.”=”の前後に空白を含めよう!!
$ echo $original          # 変数originalの内容を参照
```

```
atgc
```

コマンド置換: “\$(command)”

```
$ echo $original | rev          # revコマンドは行内を逆順出力
```

```
cgta
```

```
$ reverse=$(echo $original | rev)
$ echo ORIGINAL_SEQ is $original
$ echo REVERSE_SEQ is $reverse
```

```
ORIGINAL_SEQ is atgc
REVERSE_SEQ is cgta
```

## TEXT editor

```
$ vim hoge          # “hoge”という名前のファイルをvimで編集する.
```

今回は**VIM**というTEXT editorを用いてファイルを編集する.  
text editorはMS wordのように文書(プログラム)を書くものだが、  
MS wordのように文書を装飾する機能(太字、斜体、フォント変更など)はない.  
プログラムを書くために便利な機能が多くある.

VIMにはノーマルモードとインサートモードがある(他のモードも多くあるがここでは省略).

- **i** (ノーマルモード時): インサートモードに移行.
- **Esc** (インサートモード時): インサートモードに移行.
- **dd** (ノーマルモード時): 行削除(delete).
- **yy** (ノーマルモード時): 行コピー(yank).
- **p** (ノーマルモード時): 貼り付け(paste).
- **u** (ノーマルモード時): 元に戻す(undo).
- **:w** (ノーマルモード時): ファイルを(上書き)保存.
- **:wq** (ノーマルモード時): ファイルを(上書き)保存して終了.
- **:q!** (ノーマルモード時): ファイルに変更点を保存せずに終了.

## script作製

一連の作業内容をファイルに保存し、一つのコマンドとして再利用したい場合など.

1. scriptファイルを作製する.
2. そのファイルに実行権限を付与する.
3. そのファイルをPATHの通ったdirectory下に配置する.

### 1. script作製

```
$ vim reverse_seq
```

~/HIG0/class3/reverse\_seq

```
#!/bin/sh

seqname=$1
original=$2
reverse=$(echo $original | rev)

echo $original $reverse
```

文頭の“#!/bin/sh”をシバンという. 一種のおまじないとして覚えておくように.  
\$1や\$2は位置パラメータという.  
reverse\_seq seq1 atgc としてこのコマンドを使用した場合、\$1=seq, \$2=atgcとなる.

### 2. 実行権限を付与する.

```
$ chmod +x reverse_seq      # “chmod +x”によりコマンドとして実行可能なファイルとなる.
```

### 3. PATHを通す.

```
$ echo $PATH                # PATHは組込変数
                             # ":"区切り. コマンド用directoryへのパスの一覧
                             # これらのdirectory内のファイル名はそのままコマンド名として認識される.
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
```

```
$ vim ~/.profile            # PATH変数は通常~/.profile内で定義されている.
```

ファイル内のPATHの記載部分を  
PATH=~/**bin**:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin などのように書き換えて  
自分の作ったコマンドを入れるdirectory(~/**bin**)をPATHの先頭に追加.

```
$ . ~/.profile              # ". "コマンドを用いて変更内容を反映させる.
$ echo $PATH
```

```
/home/tom/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
```

```
$ mkdir ~/bin               # 存在しないなら作製.
$ mv reverse_seq ~/bin      # scriptをPATHの通った~/binに移動.
                             # これでコマンドとして使用できるはずだから...
$ reverse_seq seq_id_1 atacgtag  # 確認!!
```

```
seq_id_1 atacgtag gatgcata
```

練習問題

上と同様の引数を与えたときに以下を出力するコマンド”reverse\_seq\_2”を作製せよ.  
seq\_id\_1\_original atacgtag  
seq\_id\_1\_reverse gatgcata

第4講(16:00-16:50)： FILTER

trコマンド

wcコマンド

grepコマンド

sedコマンド

sortコマンド

headコマンド

tailコマンド

while read文

20221019

第5講(13:00-13:50)

gzip

tar

curl

wget

第6講(14:00-14:50)： AWK

```
$ cat employees

# 全従業員の月額給料データ
# 1: 名前
# 2: 時給 (円/hr)
```

```
# 3: 労働時間 (hr/month)
```

```
Sato      1300 225
Hirai     900 120
Nakamura  800  0
Ito       2500 205
Kato      1250 275
Suzuki    1200 180
```

問題

[1] ゼロ時間より多く働いた従業員の名前と賃金は？

[2] 働かなかった従業員の名前は？

[3]

ANSWER

[1]

```
$ cat data | awk '$3>0{print $1, $2*$3}'
```

[2]

```
$ cat data | awk '$3==0{print $1}'
```

---

第7講(15:00–15:50)： SCRIPTの作製

tm

reverse complement

---

第8講(16:00–16:50)： RNA-seq