シェルスクリプト入門

ITのチカラで研究を支援

amelieff

アメリエフ株式会社

本講義にあたって

- 講師の服部について
  - アメリエフ株式会社でバイオインフォマティクスの解析を行っています
  - LinuxやRやソフトウェアの使い方のトレーニングも担当しています
  - 新潟県出身でお酒は飲めますがスキーは下手です
  - 大学でショウジョウバエの進化を研究していました
  - 社会人になってからはずっとSEをやっています
  - 趣味は落語と三味線です

本講義にあたって

- テキストが穴埋めになっています
  - 埋めて完成させてください
- クイズがたくさんあります
  - めざせ全問正解！（賞品は特にありませんが…）
- 実習もたくさんあります
  - とにかく書いてみるのが理解の早道です
  - どうしても難しい場合は「sh_answer」ディレクトリに解答例がありますのでコピーして実行してください

本講義にあたって

- クイズの解答など、お手元の資料には入っていないページがあります

ページ右上に
 がついているスライドは
配布資料にはありません

本講義の内容

- プロローグ
- シェルスクリプトとは
- 変数
- 引数
- 条件付き処理
- 繰り返し処理
- 標準出力と標準エラー出力
- シバン
- エピローグ

プロローグ

- あなたは解析担当者です
- 今は朝の10時です
- 突然、一本の電話がかかってきました

共同研究者の○△だけど
例の解析結果が急に必要になったので
今日の18時までに送ってもらえる？
よろしく！

プロローグ

（無理だ…）あなたは頭を抱えました

- その解析はA,B,Cという3つのソフトを順番に実行する必要があるのですが…

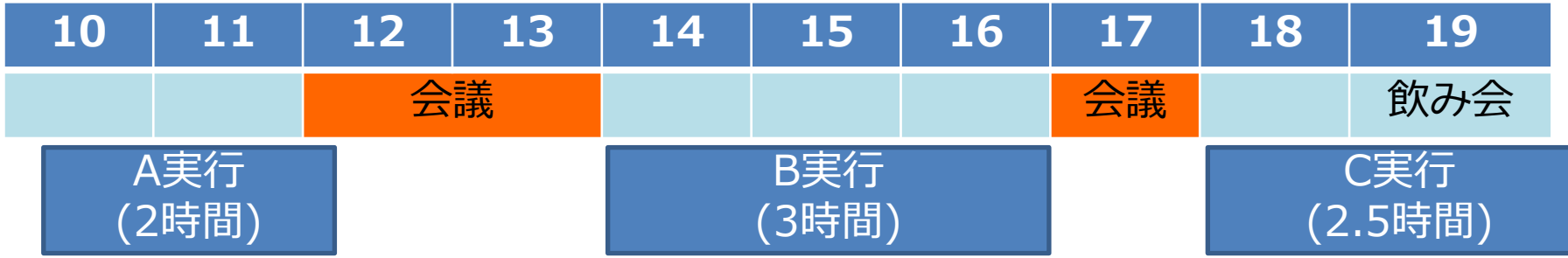
プロローグ

- 今日に限って会議が2つも入っています

（18時までに終えるのは無理だ...）

（締切を20時まで延ばしてもらえるかな？）

（飲み会は諦めよう...）



プロローグ

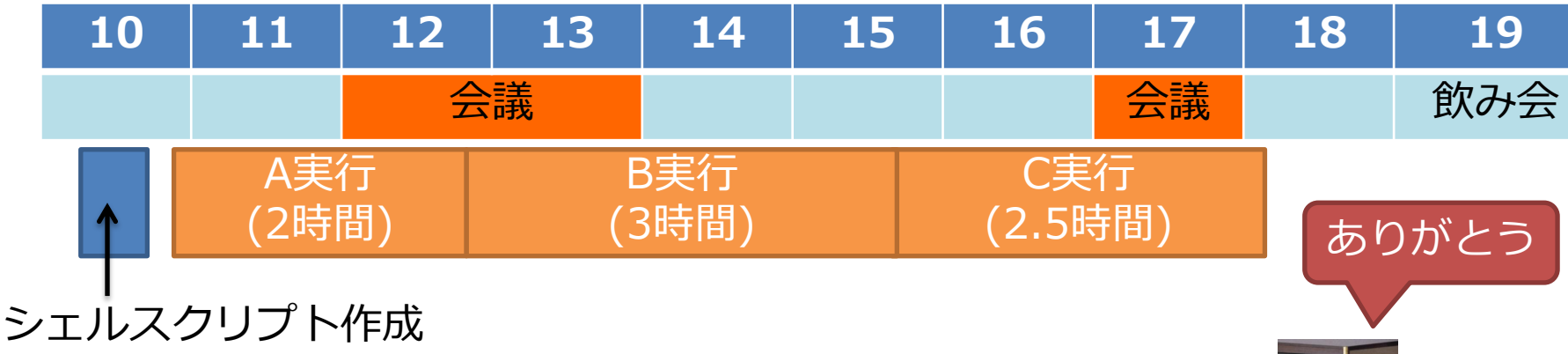
- その時です

諦めないで！

シェルスクリプトを使えば
18時までに終わるよ！

プロローグ

- シェルスクリプトのおかげであなたは共同研究者の要望に応え飲み会にも行くことができました



シェルスクリプトとは

- 「Linuxコマンド」をファイルに書いたもの
  - –書かれた内容をLinuxが自動実行
  - –変数・条件付き処理・繰り返し処理などのプログラミングが可能

シェルスクリプトのメリット

- 効率的に解析できる
  - 指定通り自動で実行されるので、解析の待ち時間が減らせる
  - 同じ処理を別のデータや異なる条件で繰り返し実行しやすい
  - 実行ログを残しやすい

「cd」実行

「pwd」実行

「ls」実行

手入力で1つずつコマンドを
実行していたのを…

まとめて実行

シェルスクリプトなら
まとめて実行できる

シェルスクリプトの強み

- バイオインフォのソフトは毎年変わる

しかし

- Linuxコマンドや、シェルスクリプトの文法はこの10年**ほとんど変わっていない**

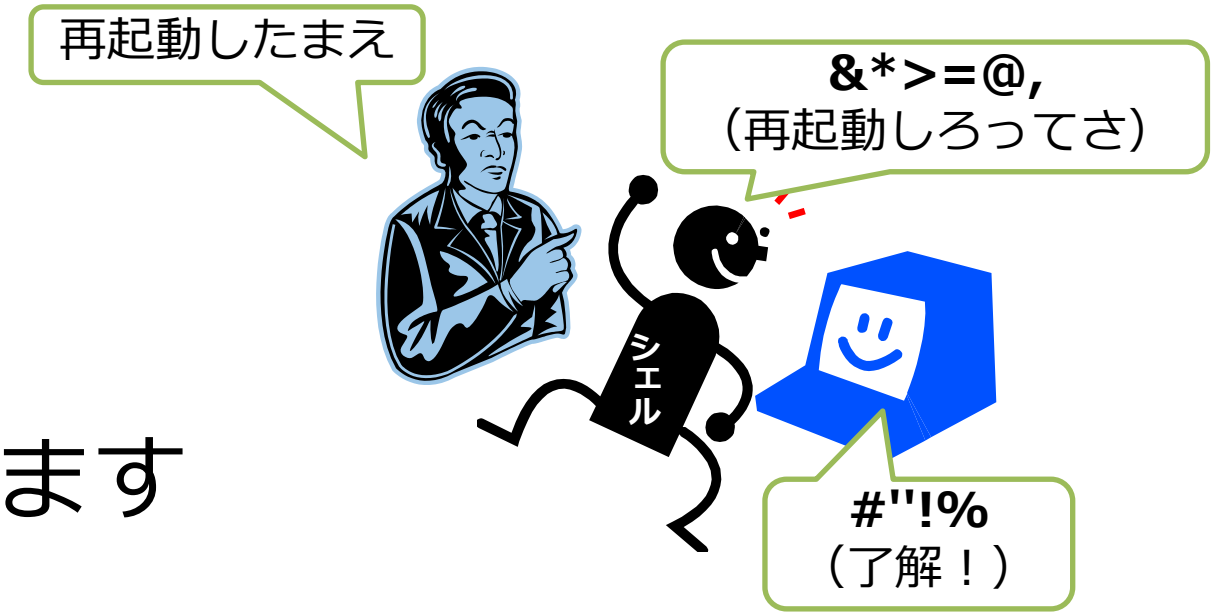
つまり

- 今身につければ（おそらく）あと10年は使える

シェルとは

- ユーザが入力したコマンドをコンピュータに伝えるプログラムです
  - –bash
  - –tcsh
  - –zsh

  などがあります



シェルの種類

- 本テキストはbashをベースとした記述になっています
- BioLinuxはデフォルトがzshです
- 基本的には大きな差はありませんが特にzshで挙動が異なるところには注釈を入れています

シェルスクリプトの作成と実行

1. テキストエディタ（vi, gedit等）で実行内容をファイルに書いて保存

   本資料末尾
   テキストエディタの使いかたは別紙をご覧ください

   シェルスクリプトファイルは拡張子を「.sh」にします

2. bashコマンドで実行

```
$ bash シェルスクリプトファイル名
```

実習環境

- 仮想環境を起動します
- ホームディレクトリの下のamelieff/shディレクトリに移動
- 実習はすべてここで行います

```
$ cd
$ cd amelieff/sh
```

実習環境

- テストデータ
  - mirbaseからダウンロードしたmiRNA配列
  - hairpin.fa（miRNA前駆体配列）
    - ftp://mirbase.org/pub/mirbase/CURRENT/hairpin.fa.zip
  - mature.fa（成熟miRNA配列）
    - ftp://mirbase.org/pub/mirbase/CURRENT/mature.fa.zip

どちらもFastaフォーマットのファイルです

Fastaフォーマット

- ＞で始まるID行と配列行（塩基またはアミノ酸）から成るフォーマットです
- ゲノムや遺伝子の配列を表すのによく使われます

```
>cel-let-7 MI0000001 Caenorhabditis elegans let-7 stem-loop
UACACUGUGGAUCCGGUGAGGUAGUAGGUUGUAUAGUUUGGAAUAUUACCACCGGUGAAC
UAUGCAAUUUUCUACCUUACCGGAGACAGAACUCUUCGA
>cel-lin-4 MI0000002 Caenorhabditis elegans lin-4 stem-loop
AUGCUUCCGGCCUGUUCCCUGAGACCUCAAGUGUGAGUGUACUAUUGAUGCUUCACACCU
GGGCUCUCCGGGUACCAGGACGGUUUGAGCAGAU
                                   :
```

ID行

配列行

達成目標

- 以下の作業をシェルスクリプトで自動実行できるようになりましょう
  - mature.faの一部の配列を切り出して別のファイルに書き出す
  - 書き出したファイルに対してweblogoというソフトウェアを実行してモチーフ図を描く

ファイルの先頭を表示する

- Linuxのheadコマンドを実行すると、指定したファイルの先頭数行が表示されます

```
$ head -n 4 mature.fa
>cel-let-7-5p MIMAT0000001 ...
UGAGGUAGUAGGUUGUAUAGUU
>cel-let-7-3p MIMAT0015091 ...
CUAUGCAAUUUUCUACCUUACC
```

head –n *k* ファイル：
ファイルの先頭*k*行を出力する

mature.faファイルの
先頭4行が表示される

実習１

- 次のシェルスクリプト・test1.shを書いて実行してみましょう
  - mature.faファイルの先頭4行を表示するシェルスクリプトです

```
$ gedit test1.sh
```

test1.shにこの1行を書いて保存します

```
head -n 4 mature.fa
```

```
$ bash test1.sh
```

実行

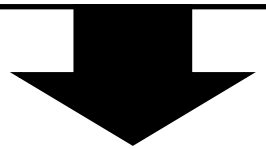
質問

- では、別のファイル・hairpin.faの先頭8行を表示するようにするにはスクリプトをどう変更すればよいでしょう？

解答

- 実行内容を以下のように変えます

```
head -n 4 mature.fa
```



```
head -n 8 hairpin.fa
```

解答

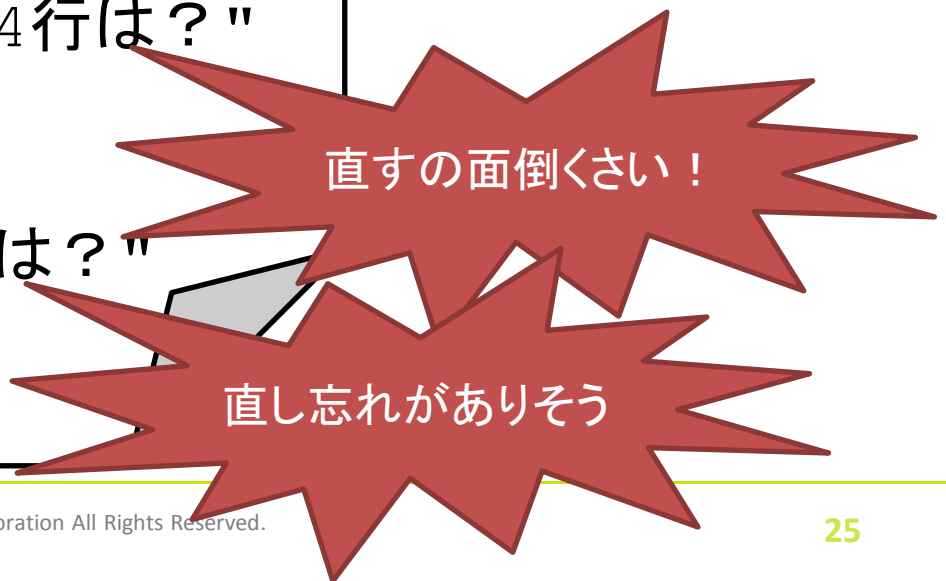
- でも、何力所もあったら？

```
echo "mature.fa の先頭4行は？"
head -n 4 mature.fa

echo "mature.fa の末尾4行は？"
tail -n 4 mature.fa

echo "mature.fa の行数は？"
wc -l mature.fa
                    :
```

echo：値を出力する
tail –n *k*：
　末尾*k*行を出力する
wc –l：行数を出力する



変数

- 「変数」を使うと値を一元管理できます

```
FILE="mature.fa"
echo "$FILE の先頭4行は？"
head -n 4 $FILE

echo "$FILE の末尾4行は？"
tail -n 4 $FILE

echo "$FILE の行数は？"
wc -l $FILE
                :
```

変数「FILE」に
ファイル名を入れる

それ以降は
「$FILE」と書くと
設定した値が
自動で入る！

変数

- 「変数」は値を格納するものです
  - 入れた値は変更することができます
  - 「変数名=値」と書くと、変数に値を代入できます
  - 「$変数」と書くと、変数に入っている値を呼び出すことができます

実習 2

- 次のシェルスクリプト・test2.shを書いて実行してみましょう

```
$ cp test1.sh test2.sh
$ gedit test2.sh
```

cp *FileA FileB*:
*FileA*を*FileB*という名前で複製

test2.shを以下のように変更して保存します

```
file="mature.fa"
num=4
head -n $num $file
```

```
$ bash test2.sh
```

実習２・解答

- 実習1と同じ挙動になります

```
file="mature.fa"
num=4
head -n $num $file
```

- 変数「file」にファイル名を入れる
- 変数「num」に表示したい行数を入れる
- headコマンドを$num, $fileを使って実行する

変数のありがたみがわかる例

- 変数を使わずに書いたスクリプトの例

```
echo "入力ファイルは A.fastq です"
echo "A.fastq のマッピング開始"
bwa mem genome A.fastq >out.sam
echo "A.fastq のマッピング終了"
                :
```

やっぱり
A.fastqはやめて
B.fastqで
実行しよう！

スクリプトを
書いた人

何箇所も直さない
といけない…

時間がかかる上に
直し忘れたりする

変数のありがたみがわかる例

- 変数を使って書いたスクリプトの例

```
file="A.fastq"
echo "入力ファイルは $file です"
echo "$file のマッピング開始"
bwa mem genome $file >out.sam
echo "$file のマッピング終了"
                :
```

ここだけ直せばよい

やっぱりA.fastqはやめてB.fastqで実行しよう！

スクリプトを書いた人

10秒で直せます！

変数にしたほうがいいもの

- 実行のたびに変わる可能性のある値
  - 例）「　入力ファイル名　」
- スクリプト内に何度も登場する値
  - 例）「 リファレンスゲノム配列のファイル名 」
- なるべく変数にしておくと後で修正がしやすい

不満

対象ファイルが変わるたびに
スクリプトファイル内の
ファイル名の値を書き換えないと
いけないのは面倒だなあ

hairpin.fa

```
file=mature.fa
num=4
head -n $num $file
```

- 「 引数 」を使うとスクリプトの中身を
  いちいち書き換えなくてよくなります

引数

- 「引数（ひきすう）」は実行時にスクリプト名以降に入力された値（複数可能）です

```
$ bash test3.sh mature.fa 4 ...
```

引数

- 引数は自動で専用の変数に入ります
  - 変数$1に1番目の引数の値が、$2に2番目の引数の値が（$3以下同様）入ります

実習 3

- 次のシェルスクリプト・test3.shを書いて実行してみましょう

```
$ cp test2.sh test3.sh
$ gedit test3.sh
```

以下のように変更して保存

```
file=$1
num=$2
head -n $num $file
```

```
$ bash test3.sh mature.fa 4
```

応用：
hairpin.faの
先頭10行を
表示するには
どう実行したら
いいでしょう？

実習３・解答

- 実習1・２と同じ挙動になります

```
$ bash test3.sh mature.fa 4
```

```
file=$1
num=$2
head -n $num $file
```

1番目の引数（mature.fa）が変数$1に
2番目の引数（4）が変数$2に入ります

–引数を変えると実行内容が変わります

```
$ bash test3.sh hairpin.fa 10
```

クイズ

- スクリプトの実行結果はどうなりますか？

Q1.sh

```
v1=$1
v2=$2
v3=$3


echo $v2
```

難易度：★

★：基本です
★★：理解できています
★★★：解けたらすごい

```
$ bash Q1.sh I love bioinformatics
```

A

```
I love bioinformatics
```

C

```
love
```

B

```
I
```

D

```
bioinformatics
```

クイズ

- スクリプトの実行結果はどうなりますか？

Q1.sh

```
v1=$1
v2=$2
v3=$3


echo $v2
```

```
$ bash Q1.sh I love bioinformatics
```

正解は、**C**！！

```
love
```

引数は空白で区切って与えます

ディレクトリを作成する

- 以下は「sun」「moon」という名前の2つのディレクトリを作成するスクリプトです

```
mkdir "sun" "moon"
```

mkdir：
ディレクトリを作成する

実習 4

- 次のシェルスクリプト・test4.shを書いて実行してみましょう
  - "sun"と"moon"というディレクトリを作成する
  - ディレクトリ名は引数で指定する

```
$ bash test4.sh sun moon
$ ls
```

sun, moonがあればOK

ls：ファイルと ディレクトリの一覧を表示

- もう一度実行してみましょう

```
$ bash test4.sh sun
moon
```

エラーが出るはずです

実習４・解答

```
dir1=$1
dir2=$2
mkdir $dir1
mkdir $dir2
```

```
$ bash test4.sh sun moon
$ ls
```

- 同じ引数で再実行するとエラー

```
mkdir: ディレクトリ 'sun' を作成できません: ファイルが存在します
mkdir: ディレクトリ 'moon' を作成できません: ファイルが存在します
```

質問

ディレクトリが存在する場合に
エラーが出なくなるには
どうしたらいいの？

- 「　条件付き処理　」を用います
- 作ろうとする名前のディレクトリが存在しない時のみmkdirするようにしましょう

条件付き処理

- 条件を満たした時だけ処理を実行させることができます

```
if␣ [␣条件␣]
then
   ～処理～
fi
```

␣:半角スペースを入れる
（実際は表示されない）

ここに空白がないと
エラーになるので注意

処理文は少し行頭を下げると見やすい

条件付き処理

ファイル*FIL*が存在して
かつ通常ファイルなら

```
if [ -f FIL ]
```

- ファイル・ディレクトリの存在確認

ファイル*FIL*が存在すれば

```
if [ -e FIL ]
```

ファイル*FIL*が存在して
かつサイズが0でなければ

```
if [ -s FIL ]
```

ディレクトリ*DIR*が存在すれば

```
if [ -d DIR ]
```

ディレクトリ*DIR*が存在しなければ

```
if [ ! -d DIR ]
```

実習 5

- 次のシェルスクリプト・test5.shを書いて実行してみましょう
  - 2つのお好きな名前のディレクトリを作成する
  - ディレクトリ名は引数で受け取る
  - ディレクトリが存在しない場合のみmkdirする
- 同じ引数で2回実行してエラーが出ないことを確認します

実習５・解答

1回目の実行

```
$ bash test5.sh DNA RNA
```

```
dir1=$1
dir2=$2
if [ ! -d $dir1 ]
then
 mkdir $dir1
fi
if [ ! -d $dir2 ]
then
 mkdir $dir2
fi
```

DNAが存在しないので
ifの中が実行される

→DNAができる

RNAが存在しないので
ifの中が実行される

→RNAができる

実習５・解答

2回目の実行

```
$ bash test5.sh DNA RNA
```

```
dir1=$1
dir2=$2
if [ ! -d $dir1 ]
then
 mkdir $dir1
fi
if [ ! -d $dir2 ]
then
 mkdir $dir2
fi
```

DNAが存在するので
ifの中が実行されない

→エラーが出ない

RNAが存在するので
ifの中が実行されない

→エラーが出ない

質問

条件付き処理では他に
どんな条件が指定できるの？

- 変数の値に応じた処理などが可能です
  - –例）変数Aが100より大きければ
  - –例）変数Bが"cancer"でなければ

条件付き処理

- 値の比較には「比較演算子」を使います
  - 数値の比較演算子

A -eq B	A＝Bなら
A -ne B	A≠Bなら
A -lt B	A＜Bなら
A -le B	A≦Bなら
A -ge B	A≧Bなら
A -gt B	A＞Bなら

  - 文字列の比較演算子

A = B	AとBが 同じなら
A != B	AとBが 異なれば

条件付き処理

- 変数を使った条件付き処理

```
TEMPERATURE=$1
if [ $TEMPERATURE -ge 30 ]
then
  echo "Is it hot today?"
fi
```

変数TEMPERATUREが
30以上だったら

「Is it hot today?」と出力

条件付き処理

- 複数の条件を指定することもできます

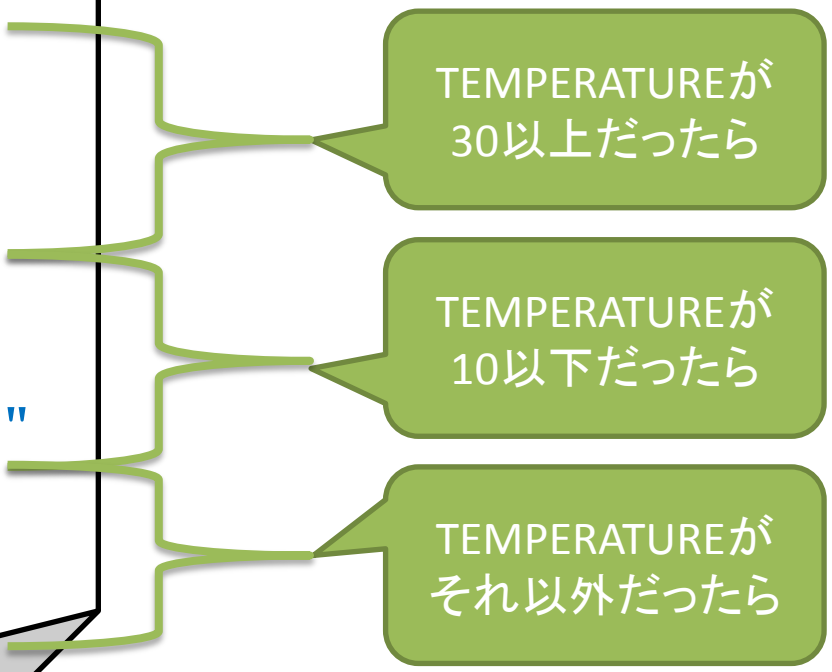
```
if   [ 条件1 ]
then
   ～条件1を満たした時の処理～
elif   [ 条件2 ]
then
   ～条件1は満たさなかったが、
     条件2を満たした時の処理～
else
   ～どの条件も満たさなかった
     時の処理～
fi
```

elif は何回でも記述可能

条件付き処理

- 複数の条件付き処理の例

```
TEMPERATURE=$1
if [ $TEMPERATURE -ge 30 ]
then
  echo "Hot enough for you?"
elif [ $TEMPERATURE -le 10 ]
then
  echo "Cold enough for you?"
else
  echo "It's a nice day
today."
fi
```



クイズ

難易度：★★

スクリプトの実行結果はどうなりますか？

実行開始時点でdir3は存在しないものとします

Q2.sh

```
mkdir dir3
cd dir3
if [ ! -f foo.txt ]
then
  touch "foo.txt"
else
  echo "既に存在します"
fi
```

```
$ bash Q2.sh
```

A

「既に存在します」と出力

B

dir3と
foo.txtが作成される

C

dir3のみ作成される

D

エラーになる

クイズ

- スクリプトの実行結果はどうなりますか？

正解は、**B**！！

dir3と
foo.txtが作成される

右のようにIf文でセミコロン（;）を使うと使用する行数を抑えることができます

Q2.sh

```
mkdir dir3
cd dir3
if [ ! -f foo.txt ]
then
  touch "foo.txt"
else
  echo "既に存在します"
fi
```

```
$ bash Q2.sh
```

Q2.sh別解

```
mkdir dir3
cd dir3
if [ ! -f foo.txt ];then
  touch "foo.txt"
else
  echo "既に存在します"
fi
```

実習５の改善点

```
dir1=$1
dir2=$2
if [ !-d $dir1 ]
then
 mkdir $dir1
fi
if [ !-d $dir2 ]
then
 mkdir $dir2
fi
```

なんとなく冗長な
感じがしませんか？

不満

ディレクトリを100個作る場合は

```
if [ !-d $dir1 ]
then
 mkdir $dir1
fi
```

を100回

書かないといけなくて大変だ！

- 「繰り返し処理」を用いれば、何度も実行する処理を1回書くだけでよくなります

繰り返し処理

- 繰り返し処理の構文

```
for␣変数␣in␣値1 値2 値3 …
do
   ～処理～
done
```

繰り返し処理

- 繰り返し処理の例
  - 「1」「2」...「100」という名前のファイルをtouchコマンドで作成するスクリプト

```
for FILE in ~~1:100~~
                `seq 1 100`
do
  touch $FILE
done
```

~~m:n mからnまでの1刻みの数~~
touch ファイルを作成する

`seq n m`
nからmまで1刻みの数

クイズ

- スクリプトの実行結果はどうなりますか？

Q3.sh

難易度：★★

```
f1=$1
f2=$2
out=$3

for f in $f1 $f2
do
 head -n 2 $f > $out
done
```

```
$ bash Q3.sh hairpin.fa mature.fa Out
```

A

hairpin.faの先頭2行がOutに出力される

B

hairpin.faの先頭2行とmature.faの先頭2行がOutに出力される

C

mature.faの先頭2行がOutに出力される

D

エラーになる

クイズ

- スクリプトの実行結果はどうなりますか？

Q3.sh

```
f1=$1
f2=$2
out=$3

for f in $f1 $f2
do
 head -n 2 $f > $out
done
```

```
$ bash Q3.sh hairpin.fa mature.fa Out
```

正解は、**C**！！

ちなみに、>を>>にすると
ファイル書き出しが追記になり
hairpin.faの先頭2行の次に、
mature.faのf2の先頭2行が
出力されます

Q3.sh修正版

```
f1=$1
f2=$2
out=$3

for f in $f1 $f2
do
 head -n 2 $f >> $out
done
```

実習 6

- 次のシェルスクリプト・test5.shを書いて実行してみましょう
  - 3つのお好きな名前のディレクトリを作成する
  - ディレクトリ名は引数で受け取る
  - ディレクトリが存在しない場合のみmkdirする
  - ディレクトリを作成する手順はfor文を使って1回だけ記述する

実習６・解答例

```
dir1=$1
dir2=$2
dir3=$3
for dir in $dir1 $dir2 $dir3
do
  if [ ! -d $dir ]
  then
   mkdir $dir
  fi
done
```

何度も実行する処理だが
書くのは一回だけなので楽

処理内容に変更があっても
ここだけ変更すればよい

不満

どのコマンドが実行されたか
実行結果が正しく終わったのか
わかりづらいよ

- 実行コマンドをechoで出力すると結果がわかりやすくなります

```
file=$1
echo "$file のマッピング開始"
bwa mem genome $file >out.sam
echo "$file のマッピング終了"
```

```
$ bash bwa.sh B.fastq
B.fastqのマッピング開始
B.fastqのマッピング終了
```

標準出力と標準エラー出力

- 正常時の出力と、エラー時の出力を区別して出すことができます

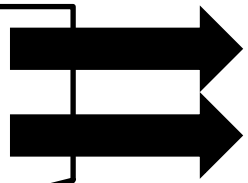
```
$ bash miso_soup.sh
ネギを切りました
豆腐を切りました
お湯が沸きました
ネギと豆腐を投入しました
エラー！味噌が見つかりません
終了します
```

エラー時の出力は区別できるようにしたい

標準出力と標準エラー出力

- 通常のechoの結果は「標準出力」へ、末尾に「>&2」をつけてechoした結果は「標準エラー出力」へ出力されます

```
echo "OK!"
echo "ERROR！" >&2
```



標準出力へ
標準エラー出力へ

実習 7

- 次のシェルスクリプト・test7.shを書いて実行してみましょう

```
echo "I'm file."
echo "Something wrong." >&2
```

- 実行結果の違いを確認します

```
$ bash test7.sh
```

どちらも画面に出力する

```
$ bash test7.sh 1>log 2>err
```

標準出力はファイルlogへ、標準エラー出力はファイルerrへ出力する

```
$ bash test7.sh >log 2>&1
```

どちらもファイルlogへ出力する

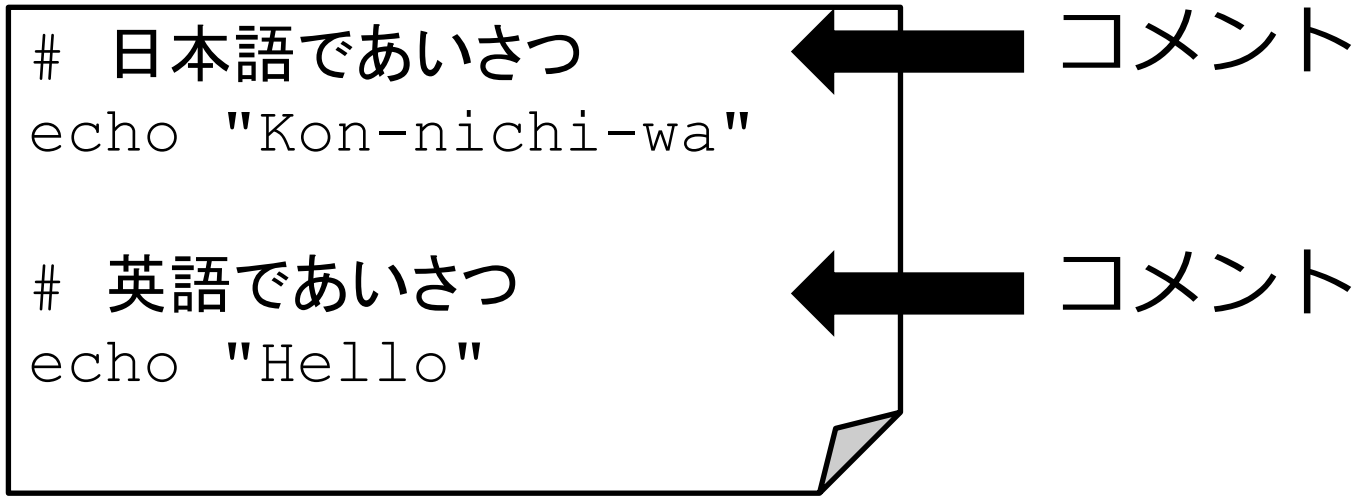
不満

他人のスクリプトはもちろん、
自分で書いたスクリプトでも
後で読み返すと何をやっているのか
わからなくなるよ

- 何をやっているかわかりやすくするため
  スクリプトに「コメント」を入れましょう

コメント

- #で始まる行はコメント扱いとなり、処理に影響しません



シバン

- スクリプトの1行目に以下を記述するとこのファイルがシェルスクリプトであることが明示的になります

```
#!/bin/bash
```

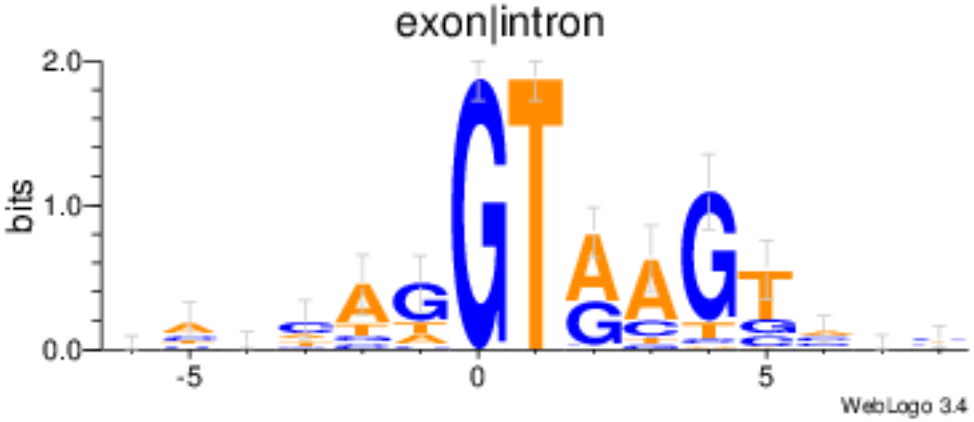
スクリプトの1行目に書く何で実行するかの指定をシバンと言います

- これにより、bashコマンドなしでも実行できるようになります

```
$ chmod a+x test8.sh
$ ./test8.sh
```

chmod a+x：実行権限をつける

weblogo



- 複数の配列を入力すると上のような組成割合の図を描くソフトウェアです
  - http://weblogo.threeplusone.com
- 以下のように実行します
  - 入力する配列は長さが揃っている必要があります

```
$ weblogo -F jpeg < fastaファイル > 結果.jpg
```

最終課題

- 次のシェルスクリプト・test8.shを書いて実行します

【基本】mkdir, headを実行する

- inとoutというディレクトリがあるか確認し、無ければ作る
- mature.faから先頭2行のみを抜き出し、in/mature_hsa.faに書き出す
  - ヒント：head -n 2 mature.fa > in/mature_hsa.fa
- ディレクトリ名やファイル名は変数を使わずに直書きで結構です

余裕のある方は次のページへ

最終課題

- test8.shを以下のように書き換えます

【やや難】mkdir, headを実行する

  - inとoutというディレクトリがあるか確認し、無ければ作る
  - mature.faから先頭2行のみを抜き出し、in/mature_hsa.faに書き出す
    - ヒント : head -n 2 mature.fa > in/mature_hsa.fa
  - ディレクトリ名やファイル名を引数で指定できるようにしましょう

余裕のある方は次のページへ

最終課題

- test8.shを以下のように書き換えます

【やや難】mkdir, grepを実行する

  - inとoutというディレクトリがあるか確認し、無ければ作る
  - mature.faから各生物種の「miR390-5p」のIDと配列のみを抜き出し、in/mature_hsa.faに書き出す
    - ヒント：grep -A 1 'miR390-5p' mature.fa | grep -v '^--$' - > in/mature_hsa.fa

余裕のある方は次のページへ

最終課題

- test8.shを以下のように書き換えます

【より難】mkdir, grep, weblogoを実行する

- inとoutというディレクトリがあるか確認し、無ければ作る
- mature.faから各生物種の「miR390-5p」のIDと配列のみを抜き出し、in/mature_hsa.faに書き出す
  - ヒント：grep -A 1 miR390-5p mature.fa | grep -v '^--$' - > in/mature_hsa.fa
- in/mature_hsa.faをweblogoに入力して結果をout/mature_hsa.jpgに出力する

余裕のある方は次のページへ

最終課題

- test8.shについて

【難】hairpin.faに対して実行するにはどう変更したらよいか考えてみる

エピローグ

- hairpin.faでは塩基配列が複数行になっている場合があるため、mature.faのスクリプトがそのまま使えません
- このような場合には、Perlが必要です

Perl編につづく

クイズ

- スクリプトの実行結果はどうなりますか？実行開始時点でdir1は存在しないものとします

Q4.sh

```
mkdir dir1

cd dir1

touch a.txt b.txt c.txt

ls -l *.txt > tmp.txt

wc -l tmp.txt
```

難易度：★ ★ ★

touch:ファイルのアクセス時刻と修正時刻を変更する（ファイルが存在しない場合は新規作成する）
wc -l：ファイルの行数を返す

```
$ bash Q4.sh
```

A
作成した全てのテキストファイルとディレクトリの数が返ってくる

B
dir1内に作成したテキストファイルの数が返ってくる

C
dir1に移動する

D
tmp.txtに記載された内容が確認できる

解答

正解は、**B**！！

dir1内に作成したテキストファイルの数が返ってくる

Q4.sh

```
mkdir dir1

cd dir1

touch a.txt b.txt c.txt

ls -l *.txt > tmp.txt

wc -l tmp.txt
```

① mkdir でdir1ディレクトリを作成
② cd で作成したディレクトリへ移動
③ touch でテキストファイルを作成
④ ls -lで拡張子が「.txt」のファイルを全てリスト表示し、その結果を「>」で tmp.txtへ記載
⑤ wc -lで tmp.txtに記載された行数をカウント

tmp.txtの記述を確認する場合 `$ less tmp.txt`

クイズ

- f2.txtには標準エラー出力が何行記載されるでしょう？

実行開始時点でdir2は
存在しないものとします

Q5.sh

難易度：★ ★ ★

```
mkdir dir2

cd dir2

echo "ok" > f1.txt 2> f2.txt

head f3.txt > f1.txt 2> f2.txt
```

```
$ bash Q5.sh
```

A　B　C　D

0　1　2　3

解答

正解は、**B**！

1

Q5.sh

```
mkdir dir2

cd dir2

echo "ok" > f1.txt 2> f2.txt

head f3.txt > f1.txt 2> f2.txt
```

1は標準出力、2は標準エラー出力を表します

1と2の結果をそれぞれf1.txtとf2.txtに振り分けたいときは、
$ コマンド 1>f1.txt 2>f2.txt　と入力します

① mkdir でdir2ディレクトリを作成
② cd で作成したディレクトリへ移動
③ echo "ok"の実行結果をf1.txtへ、エラーをf2.txtへ出力します
　エラーは出ないので、この時点でf1.txtは「ok」1行、f2.txtは空行です
④ head f3.txtを実行すると、f3.txtが存在しないのでエラーがf2.txtに
　書き出されます

Linuxのテキストエディタ

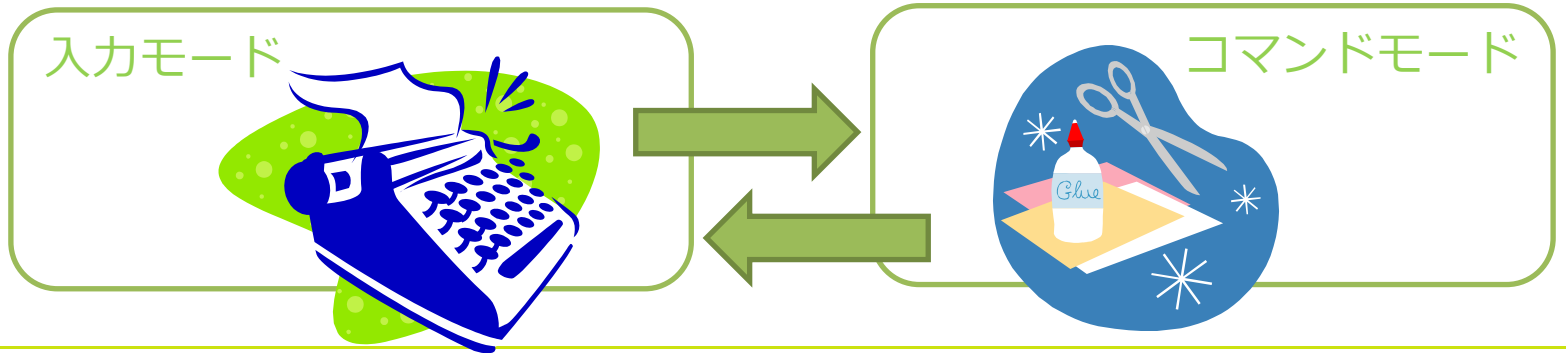
- GUIのエディタとCUIのエディタがあります
  - GUI：Windows/Macソフトのように、マウスで操作する
    - 長所：Linux初心者にも操作が容易
    - 短所：GUIがない環境では使えない
  - CUI：キーボードからコマンドで操作する
    - 長所：GUIがない環境でも使える
    - 短所：操作コマンドを覚える必要がある

gedit

- CentOSにはデフォルトでgeditというGUIエディタが入っています
- geditを起動するには `$ gedit` コマンドを実行します

vi

- CentOSにはデフォルトでviというCUIエディタが入っています
- viを起動するには `$ vi` コマンドを実行します
- viには2つのモードがあり、モードを切り替えながら操作します
  - 入力モード：文字を入力する
  - コマンドモード：編集する（切り貼り、ファイルの保存など）



vi

- 入力モードのコマンド

Escキー	コマンドモードに移行

- コマンドモードのコマンド

a	入力モードに移行（カーソルの右から入力）
o	入力モードに移行（次の行の行頭から入力）
x	1文字カット
dd	今いる行をカット
yy	1行コピー
p	カットした行をペースト
[数字]g	[数字]行に移動
G	最終行に移動
:%s/*foo*/*bar*/	文字列置換（*foo*を*bar*に置換）