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

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

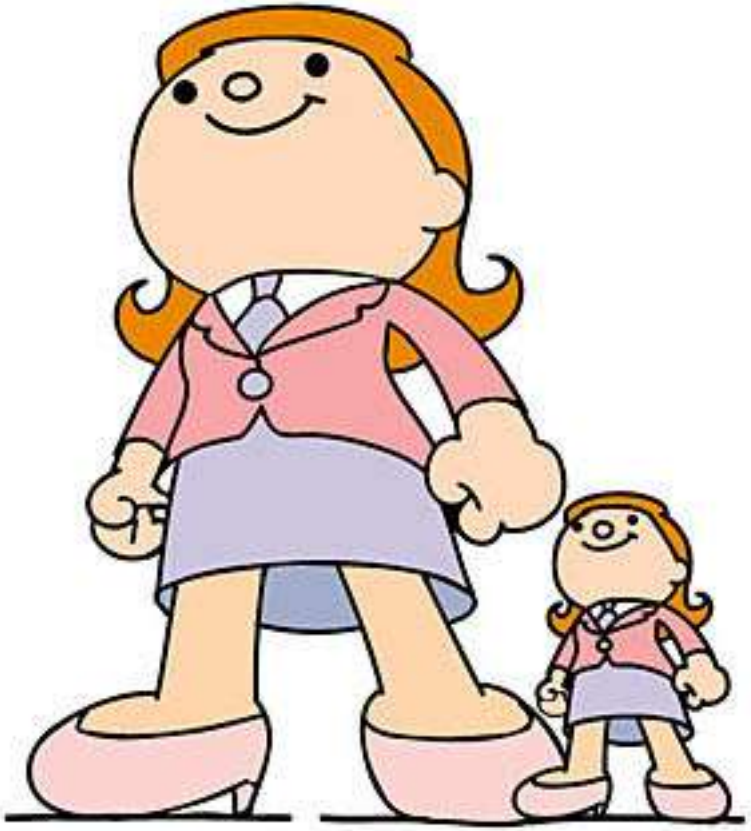
クイズがたくさんあります

めざせ全問正解！

実習がたくさんあります

とにかく書いてみるのが理解の早道です

シェルスクリプトが導く明るい未来



シェルスクリプトが導く明るい未来

- あなたは解析担当者です
- 今は朝の10時です
- 共同研究者から一本の電話がかかってきました

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？

シェルスクリプトが導く明るい未来

(無理だ...)

シェルスクリプトが導く明るい未来

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...

シェルスクリプトが導く明るい未来

今日に限って会議が2つも入っています

10	11	12	13	14	15	16	17	18	19
		会議					会議		飲み会

A実行
(2時間)

B実行
(3時間)

C実行
(2.5時間)

期限

(18時までに終えるのは
無理だ…)

(締め切りを20時まで伸ばしてもらえるかな？)

(飲み会は諦めよう)

シェルスクリプトが導く明るい未来

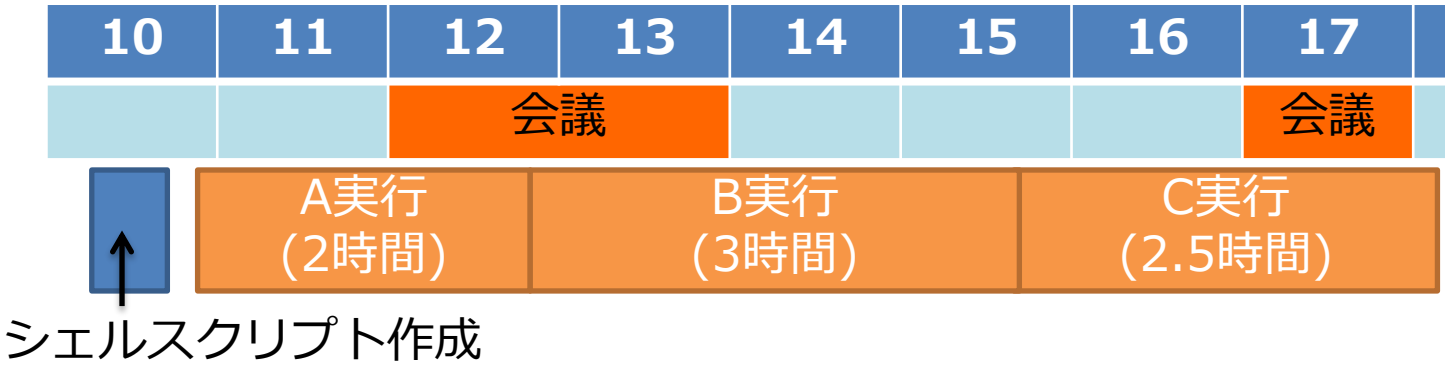
その時です

諦めないで！

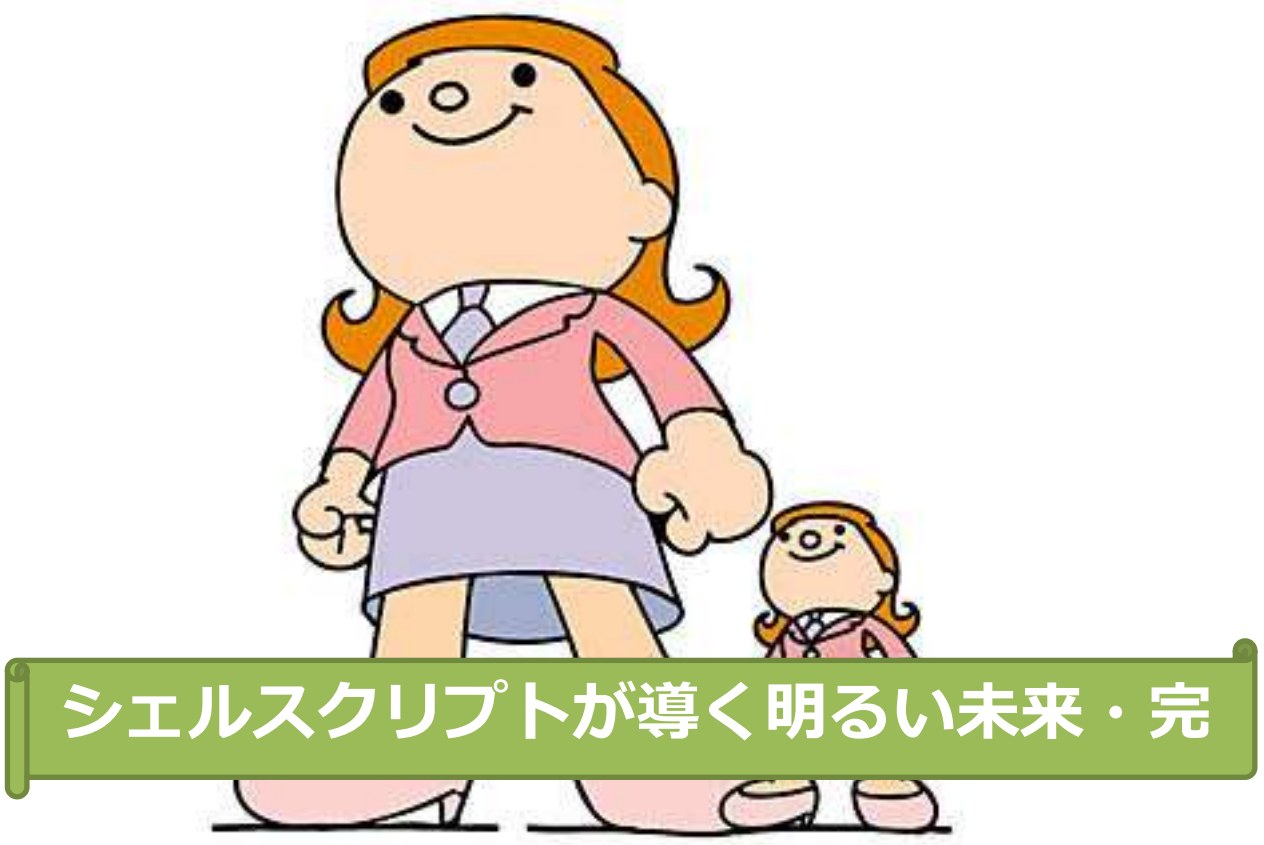
シェルスクリプトを使えば
18時までに終わるよ！

シェルスクリプトが導く明るい未来

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
飲み会にも間に合いました



シェルスクリプトを使うと時間を効率的に使える！



シェルスクリプトが導く明るい未来・完

本講義の内容

シェルスクリプトとは

文法の話
- 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

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

手入力で1つずつコマンドを
実行していたのを…

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日進月歩

しかし

Linuxコマンドや、シェルスクリプトの
文法はこの10年大きくは変わっていない

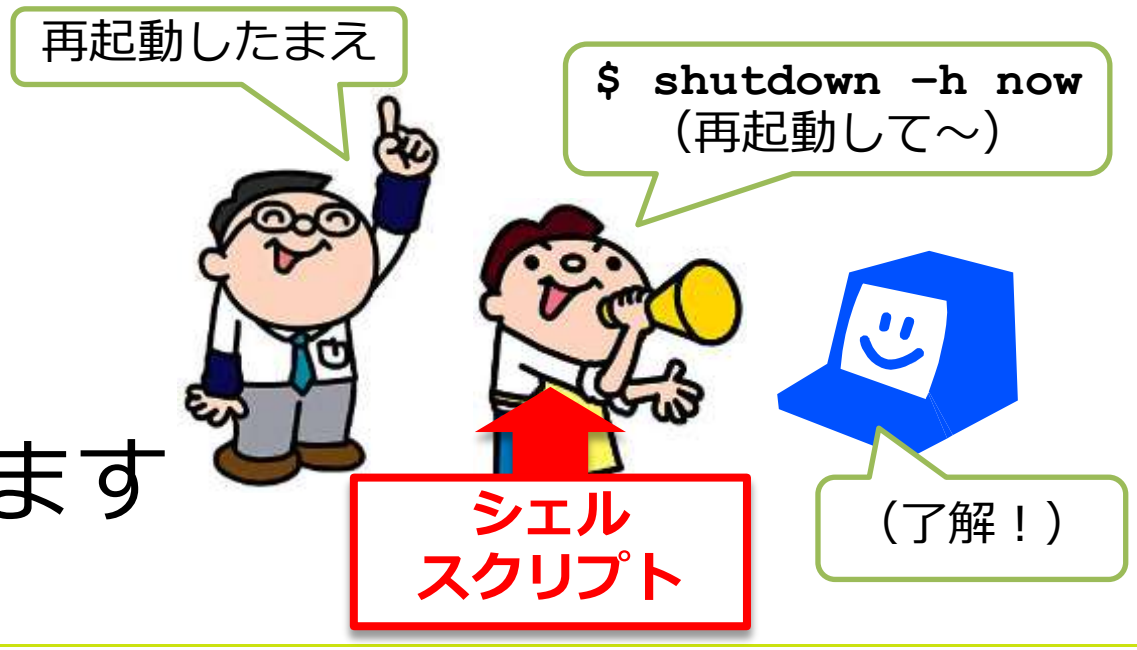
つまり

一度身につければ長く使える！

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
zsh
などの
種類があります



シェルの種類

本テキストはzshをベースとした記述になっていますがbashでもほぼ同じ挙動になります

- BioLinuxはデフォルトがzshです

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

   テキストエディタの使いかたは資料末尾をご覧ください

   シェルスクリプトファイルは拡張子を「.sh」にします

2. shコマンドで実行

```
$ sh シェルスクリプトファイル名
```

実習環境

1. 仮想環境を起動します
2. デスクトップに「sh」ディレクトリを作成します

```
$ cd ~/Desktop
$ mkdir sh
$ cd sh
```

本日の実習はすべてこの中で行います

実習環境

テストデータ

デスクトップの「Sample Data」から「sh」に
以下の2ファイルをコピーしてください

「../S」だけ入力してTabキーを押すと「Sample¥ Data」まで入ります

```
$ cp ../Sample¥ Data/peptide_seqs/p
eptides_longer_headers.fasta .
$ cp ../Sample¥ Data/peptide_seqs/p
eptides_short_headers.fasta .
```

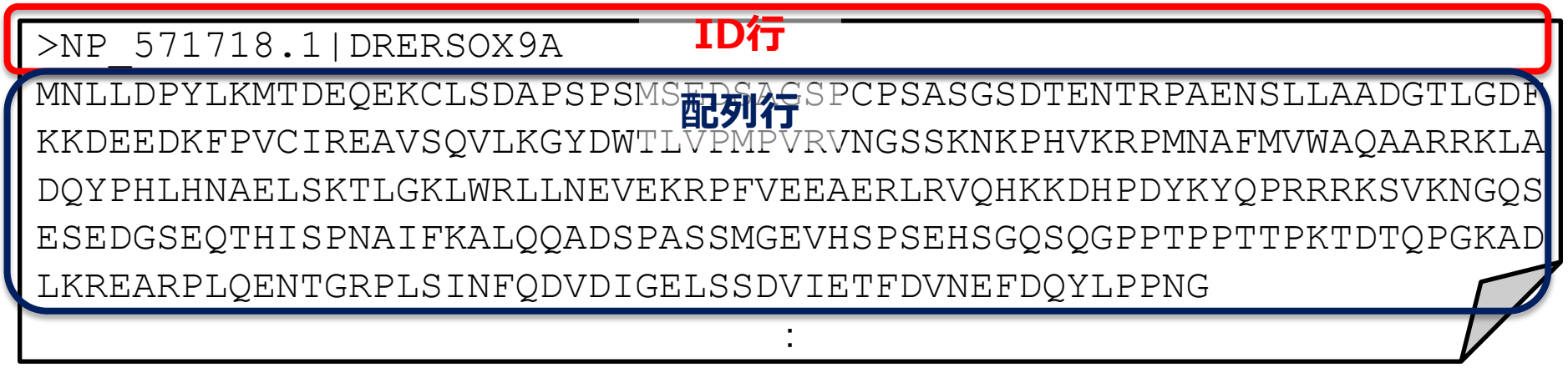
改行を入れずに続けて入力

改行を入れずに続けて入力

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



本講義の達成目標

以下の作業をシェルスクリプトで
実行できるようになります

「Fastaファイルから指定した遺伝子の
配列だけを取り出す」

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 *k* ファイル：
ファイルの先頭*k*行を出力する

```
$ head -n 4 peptides_short_headers.fasta
>DRERSOX9A
MNLLDPYLKMTDEQEKCLSDAPSPSMSEDSAGSPCPSASGSDTENTRPAENS ..
FPVCIREAVSQVLKGYDWTLVPMPVRVNGSSKNKPHVKRPMNAFMVWAQAAR ..
LGKLWRLLNEVEKRPFVEEAERLRVQHKKDHPDYKYQPRRRKSVKNGQSESE ..
```

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

peptides_short_headers.fastaファイルの先頭4行を表示するスクリプト

```
$ gedit test1.sh
```

test1.shにこの1行を書いて保存します

```
head -n 4 peptides_short_headers.fasta
```

```
$ sh test1.sh
```

実行

質問

では、もう一つのファイル
peptides_**longer**_headers.fasta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 peptides_short_headers.fasta
```

```
head -n 8 peptides_longer_headers.fasta
```

こんなときはどうする

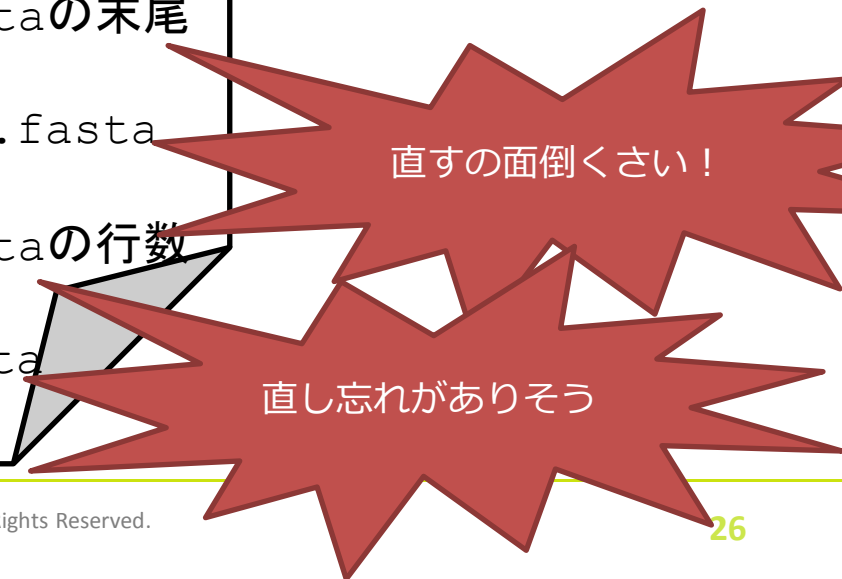
ファイル名が何回も記載されていた場合は？

```
echo "peptides_short_headers.fastaの先頭4行は？"
head -n 4 peptides_short_headers.fasta

echo "peptides_short_headers.fastaの末尾4行は？"
tail -n 4 peptides_short_headers.fasta

echo "peptides_short_headers.fastaの行数は？"
wc -l peptides_short_headers.fasta
                    :
```

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="peptides_short_headers.fasta"
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（文字列はダブルクオート(“)で囲みます）
- 「**$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="peptides_short_headers.fasta"
num=4
head -n $num $file
```

```
$ sh test2.sh
```

実習２・解答

実習1と同じ挙動になります

```
file="peptides_short_headers.fasta"
num=4
head -n $num $file
```

1. 変数「file」にファイル名を入れる
2. 変数「num」に表示したい行数を入れる
3. 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じゃなく
B.fastqで
実行しよう！

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じゃなく
B.fastqで
実行しよう！

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）「　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

peptides_longer_headers.fasta

```
file=peptides_short_headers.fasta
num=4
head -n $num $file
```

「 引数 」を使うと実行するファイル名
などを外から与えられるようになります

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（空白区切りで複数入力可）です

```
$ sh test3.sh peptides_short_headers.fasta 4 ...
```

引数

引数の値は専用の変数に入ります

変数$1に1番目の値が、$2に2番目の値が（$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
```

引数にファイル名と行数を指定して実行

```
$ sh test3.sh peptides_short_headers.fasta 4
```

実習３・解答

実習1・２と同じ挙動になります

```
$ sh test3.sh peptides_short_headers.fasta 4
```

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

1番目の値（peptides...）が変数$1に
2番目の値（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

```
$ sh test3.sh peptides_longer_headers.fasta 10
```

クイズ

実行結果は
どうなりますか？

難易度：★

Q1.sh

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

echo $v2
```

★：基本です
★★：理解できています
★★★：解けたらすごい

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

A

```
I love bioinformatics
```

C

```
love
```

B

```
I
```

D

```
bioinformatics
```

クイズ

正解は、**C**！！

```
love
```

Q1.sh

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

echo $v2
```

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

1つ目の値
→$1に入る

2つ目の値
→$2に入る

3つ目の値
→$3に入る

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
シェルスクリプトです

```
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

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

sunとmoonができていれば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

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

警告が出るはずです

実習４・解答

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

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

同じコマンドを再実行すると警告が出ます

```
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
警告を出さなくするには
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するようにします

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（インデント）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

同じコマンドを繰り返し実行しても
警告が出ないことを確認します

実習５・解答

1回目の実行

```
$ sh 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

```
$ sh 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

→警告が出ない

RNAが存在するので
ifの中が実行されない

→警告が出ない

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
```

TEMPERATUREが30以上だったら

TEMPERATUREが10以下だったら

TEMPERATUREがそれ以外だったら

クイズ

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

難易度：★★

Q2.sh

```
mkdir "dir3"
cd "dir3"
if [ ! -f "foo.txt" ]
then
  touch "foo.txt"
else
  echo "It already exists."
fi
```

```
$ sh Q2.sh
```

A

「It already exists.」と
出力される

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

正解は、**B**！！

dir3と
foo.txtが作成される

Q2.sh

```
mkdir "dir3"
cd "dir3"
if [ ! -f "foo.txt" ]
then
  touch "foo.txt"
else
  echo "It already exists."
fi
```

```
$ sh Q2.sh
```

Q2.sh別解

```
mkdir "dir3"
cd "dir3"
if [ ! -f "foo.txt" ];then
  touch "foo.txt"
else
  echo "It already exists."
fi
```

右のようにIf文でセミコロン（;）
を使うと1行に書くことができます

実習５を再度見てみましょう

```
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でも1回だけ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.txt」「2.txt」...「100.txt」という名前のファイルをtouchコマンドで作成する

```
for FILE in `seq 1 100`
do
  touch $FILE".txt"
done
```

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

実行結果は
どうなりますか？

難易度：★★

Q3.sh

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

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

```
$ sh Q3.sh File1 File2 Out
```

A

File1の先頭2行がOutに出力される

B

File1の先頭2行とFile2の先頭2行がOutに出力される

C

File2の先頭2行がOutに出力される

D

エラーになる

クイズ

正解は、**C**！！

File2の先頭2行がOutに出力される

Q3.sh

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

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

Q3.sh修正版

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

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

【参考】ファイルに追記するには
>を>>にすると
ファイル書き出しが追記になり
File1の先頭2行の次に、
File2のf2の先頭2行が
出力されます

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

もっと便利にする

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"
```

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

もっと便利にする

exitで処理を終了できます

```
for i in `seq 1 10`;do
  echo $i
   if [ $i -eq 3 ];then
   echo 'Duh!'
     exit
   fi
done
```

```
$ sh duh.sh
1
2
3
Duh!
```

標準出力と標準エラー出力

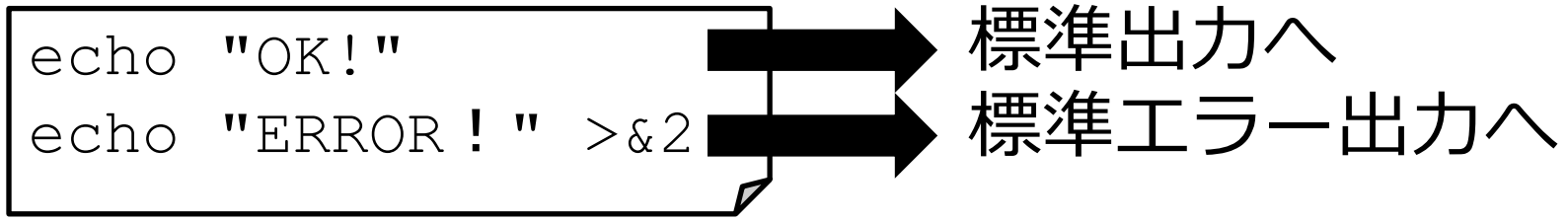
正常時の出力と、エラー時の出力を
区別して出すことができます

```
$ sh 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



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 fine."
echo "Something wrong." >&2
```

実行結果の違いを確認します

```
$ sh test7.sh
```

どちらも画面に出力する

```
$ sh 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

```
$ sh test7.sh >logall 2>&1
```

どちらもファイルlogallへ出力する

不満

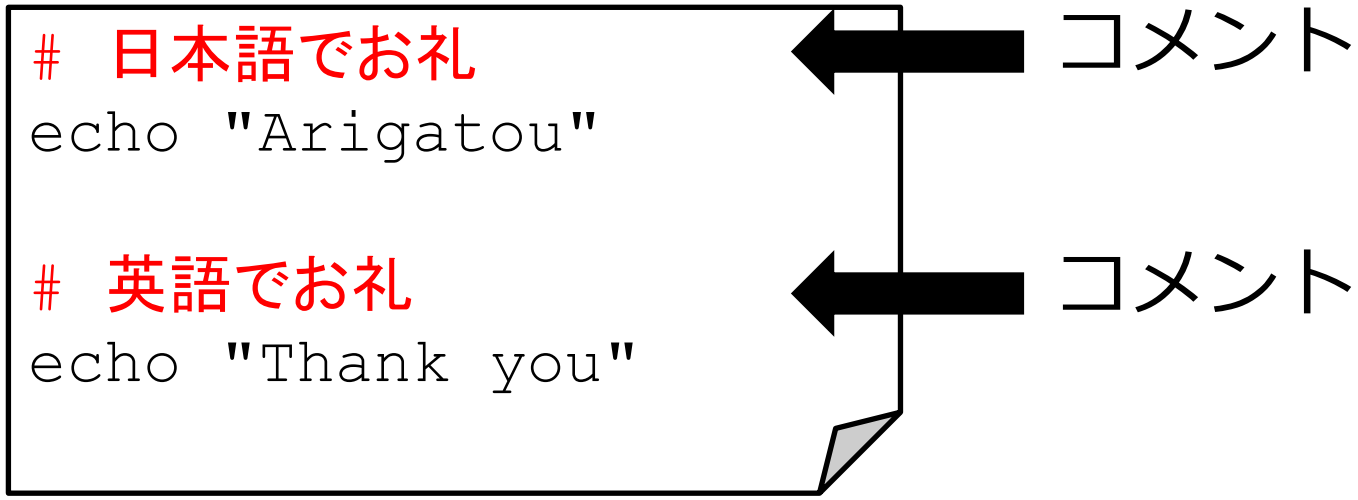
他人のスクリプトはもちろん、
自分で書いたスクリプトでも
後で読み返すと何をやっているのか
わからなくなるよ

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/sh
```

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、shコマンドなしでも実行できるようになります

```
$ chmod a+x test7.sh
$ ./test7.sh
```

chmod a+x：実行権限をつける

クイズ

実行結果はどうなるでしょう？

Q4.sh

難易度：★ ★

```
#!/bin/sh

echo "Humpty "
echo "Dumpty " >&2
echo "sat on "
exit
echo "a wall" >&2
```

```
$ chmod a+x Q4.sh
$ ./Q4.sh 2>egg.txt
```

A

egg.txtに以下が出力される

Humpty sat on

B

egg.txtに以下が出力される

Dumpty

C

egg.txtに以下が出力される

Humpty Dumpty sat on a wall

D

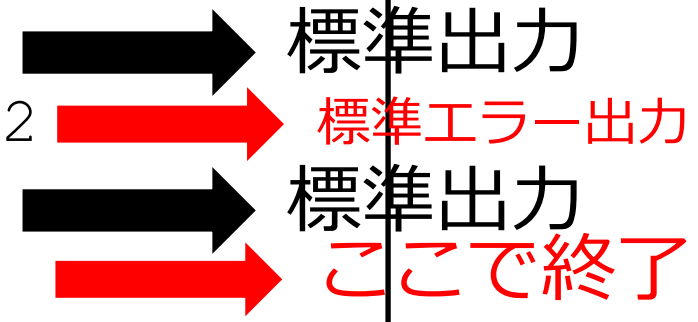
エラーになる

解答

正解は、**B**！

```
#!/bin/sh

echo "Humpty "
echo "Dumpty " >&2
echo "sat on "
exit
echo "a wall" >&2
```



cdbtools

Fastaファイルを操作するソフトウェアです
https://umbc.rnet.missouri.edu/resources/How2RunCdbtools.html

以下の2コマンドからなります
1. cdbfasta：Fastaにインデックスをつける（前準備）
2. cdbyank：Fastaから指定した配列を取り出す

実行例（peptides_short_headers.fastaからDRERSOX9Aの配列を取り出す）：

```
$ cdbfasta peptides_short_headers.fasta
$ cdbyank -a 'DRERSOX9A' peptides_short_headers.fasta.cidx
```

最終課題（1/2）

「peptides_short_headers.fastaから、
DRERSOX9A遺伝子の配列だけを抜き出す」

```
$ grep '>' peptides_short_headers.fasta
>DRERSOX9A
>CAURSOX9A
>OMYKSOX9
>OLATSOX9B
>TNIGUnk
>XTRPSOX9
>RRUGSOX9A
>DRERSOX9B
>CCARSOX9B
>CAURSOX9B
>TNIGUnm
>ASTUSOX9
>HSAPSOX9
```

peptides_short_headers.fastaに
書かれている遺伝子名の一覧

```
>DRERSOX9A
MNLLDPYLKMTDEQEKCLSDAPSPSMSEDSAGSPCPSASGSDTENTRPAENSLLAADGTLGDFKKDEEDK
FPVCIREAVSQVLKGYDWTLVPMPVRVNGSSKNKPHVKRPMNAFMVWAQAARRKLADQYPHLHNAELSKT
LGKLWRLLNEVEKRPFVEEAERLRVQHKKDHPDYKYQPRRRKSVKNGQSESEDGSEQTHISPNAIFKALQ
QADSPASSMGEVHSPSEHSGQSQGPPTPPTTPKTDTQPGKADLKREARPLQENTGRPLSINFQDVDIGEL
SSDVIETFDVNEFDQYLPPNGHQNAPYAGGYAAWMTKPQNGSPQSSQLTPLNPAEPDQPRTTHIKTEQLS
PSHYNEQQGSPQHISYGSFNVQHLQHYSTSFPSITRAQYDYSDSHQGGASSYYTHAGGQSSGLYSTFSYM
SSSQRPMYTPIADSTGVPSIPQSNHSPQHWDQQPVYTQLSRP
```

最終課題（2/2）

次のシェルスクリプト・test8.shを書いて実行します

1. Fastaファイル名をコマンドラインから引数で受け取り変数FASTAに入れる
   FASTA=$1
2. $FASTAの値をechoし、指定した値が入っていることを確認する
   echo $FASTA
   ↑まずはここまでやってみましょう
3. $FASTAに対し、以下のコマンドでインデックスを作成する
   cdbfasta $FASTA
4. 変数CIDX、変数OUTに以下の文字列を入れ、echoで確認
   CIDX="${FASTA}.cidx"
   OUT="${FASTA}.sub.fasta"
   echo $CIDX, $OUT
5. $CIDXに対し、以下のコマンドで配列を取り出す
   cdbyank -a 'DRERSOX9A' -o $OUT $CIDX
6. 3と5のコマンド文を実行前にechoする

```
$ sh test8.sh peptides_short_headers.fasta
```

＜応用編＞
- シバンをつける
- FASTAの存在確認を行い、存在しない場合は標準エラー出力にエラーメッセージを出して終了する
- 遺伝子名も引数で受け取るようにし、別の遺伝子に変えて実行する

```
$ sh test8.sh peptides_short_headers.fasta DRERSOX9A >log 2>err
```

現在いる場所を確認する【pwd】

現在Linuxのどのディレクトリにいるか確認するには次のコマンドを実行します

```
$ pwd
```

コマンドを入力した後、Enterキーを押すとコマンドが実行されます

ディレクトリ内を確認する【ls】

現在いる場所にどのようなファイル・ディレクトリがあるか確認するには次のコマンドを実行します

```
$ ls -l
```

-lをつけて実行するとlsだけを実行するより詳しい結果が表示されます（アクセス権限など）
-lを「オプション」と呼びます

他のディレクトリに移動する【cd】

他のディレクトリに移動するには次のコマンドを実行します

```
$ cd 移動先ディレクトリ
```

コマンドとオプションの間、コマンドと値の間には半角空白を1つ以上入れます

ディレクトリを作成する【mkdir】

```
$ mkdir 移動先ディレクトリ
```

ファイルを作成する【touch】

```
$ touch 作成するファイル名
```

ファイルを閲覧するにはlessやmore、
ファイルを編集するにはgeditやviを使います

ファイルを編集する【gedit】

```
$ gedit 編集するファイル名
```

ファイルが存在しない場合は新規作成されます
GUI環境がない場合はviを使います

ファイルまたはディレクトリをコピーする【cp】

```
$ cp ファイル名|ディレクトリ名 コピー先名
```

ファイルまたはディレクトリを移動する【mv】

```
$ mv ファイル名|ディレクトリ名 コピー先名
```

アクセス権限を変更する【chmod】

```
$ chmod 付与する権限 ファイル名|ディレクトリ名
```

権限の例）755：全員に読み書き実行を許可、700：所有者のみに読み書き実行を許可

主な解凍コマンド

拡張子	圧縮形式	コマンド
.tar.gz	gzip	$ tar zxvf *ファイル名*
.tar.bz2	bzip2	$ tar jxvf *ファイル名*
.gz	gzip	$ gunzip *ファイル名*
		$ gzip -d *ファイル名*
.bz2	bzip2	$ bunzip2 *ファイル名*
		$ bzip2 -d *ファイル名*
.zip	zip	$ unzip*ファイル名*
.tar	tar	$ tar xvf *ファイル名*

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

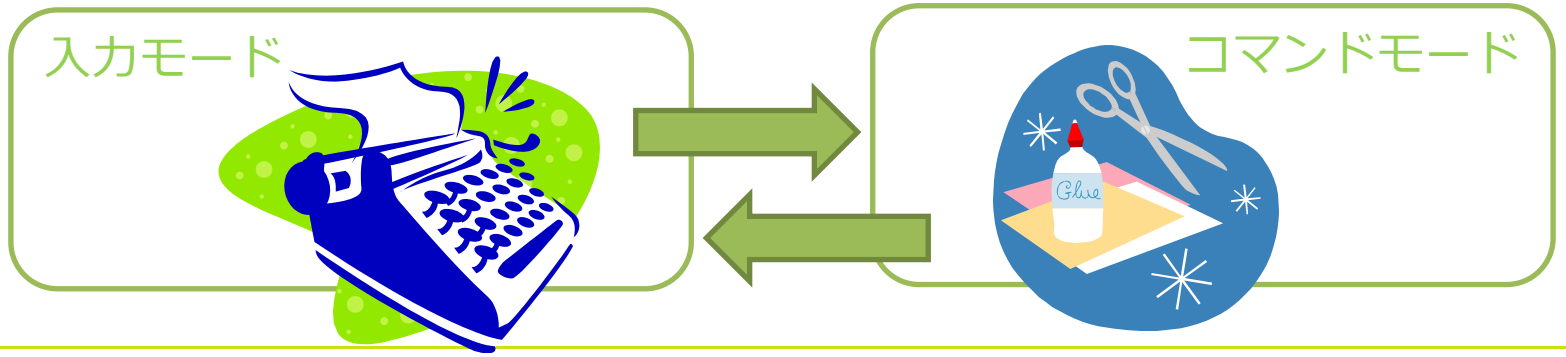
CentOSにはデフォルトでviというCUIエディタが入っています

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）