Perl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

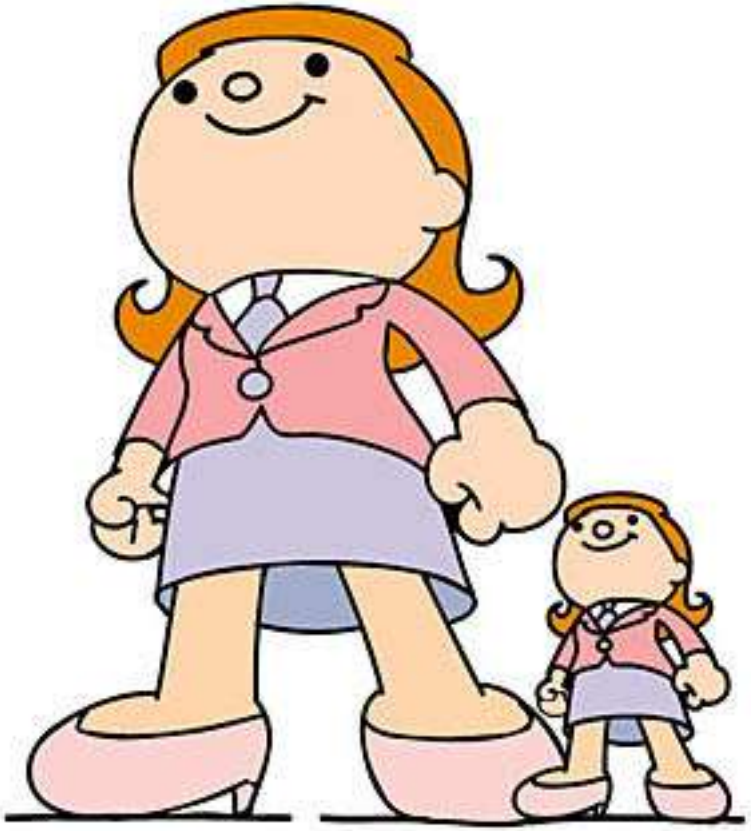
クイズがたくさんあります

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

Perlが導く
明るい未来



Perlが導く明るい未来

あなたは解析担当者です
シェルスクリプトを使いこなして毎日
効率的に解析しています
共同研究者から電話がかかってきました

この間はどうも！
今メールで送ったファイルをXという
ソフトで実行してもらえるかな？

Ｐｅｒｌが導く明るい未来

（送られてきたのは
Excelファイル
じゃないか！）

（ソフトXで
このファイルを
実行するのは
無理だ…）

（ソフトXはExcelファイルを
入力できるように
なっていないのに）

バイオインフォのすばらしいソフトウェアがたくさん公開されています
入出力するファイルのフォーマットが共通化されてきてはいますが
ソフト独自仕様になっていて他との互換性がないことがよくあります

Perlが導く明るい未来

- その時です

諦めないで！

Perlを使えばフォーマットを
変換できるかもしれないよ！

Perlが導く明るい未来

あなたはPerlを使って、受け取った
ファイルをソフトXへ入力できる
フォーマットに変換し、無事Xを
実行することができました

Perlが書けると
データ操作が捗る！

本講義の内容

Perlとは

文法の話

- 変数
- 配列
- コマンドライン引数
- ハッシュ
- 条件付き処理
- 繰り返し処理
- ファイル入出力
- シバン
- 正規表現

Perlとは

オープンソースのプログラミング言語の一つです
高速な処理には向きませんが、比較的手軽に書けることと、「テキスト処理」が得意なところから、バイオインフォマティクス業界でよく使われています

Perlのゆるさ

Perlは同じ処理をいろいろな書き方で書ける言語です

資料中のクイズは弊社社員で手分けして
考えましたのでいろいろな書き方が出てきます

- 解答例は一例です
- 「こう書くともっと良いのでは？」というスクリプトが書けた方は積極的に教えてください
- 様々な解を皆でシェアしましょう

シェルスクリプトとの比較

Perlのほうが複雑な処理に向きます

	シェル スクリプト	Perl
コマンド連続実行	○	○
変数・条件付き処理・繰り返し処理	○	○
ファイル読み込み	○	○
正規表現・複雑な計算・複雑な処理	△	○

Perlを使うとよい場面

ファイルフォーマットの変換

例）「 BAMフォーマットをBEDフォーマットに変換 」

（一般的なフォーマット間であれば大体変換スクリプトがありますが…）

結果ファイルの独自解析

例）「 異なるソフトの出力結果をマージ 」

まずは日常会話から

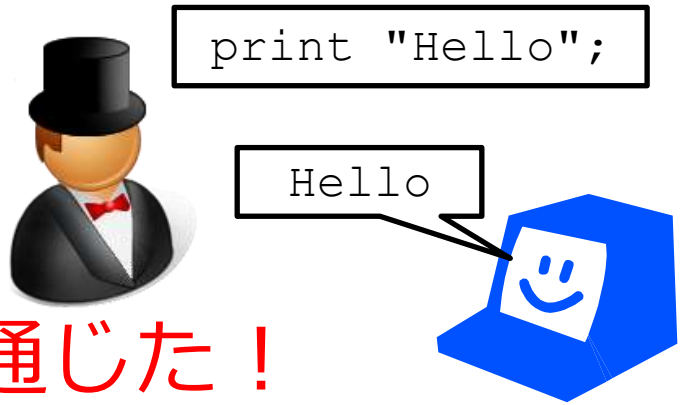
「英語が苦手なのに、**来月海外の学会**に行くことになってしまった！」

–ネイティブに負けないくらいの英語力を身に着けよう　→無謀

–とりあえず学会参加に最低限必要な英語力を身に着けよう　→現実的

まずは日常会話から

Perlも「言語」なのは英語と同じです
解析に必要な「日常会話」をとりあえず
喋って（＝書いて）みましょう



まずは日常会話から

Perlでは複雑なプログラムを書くこともできます…が

本講義ではバイオの解析を行うのに
必要最低限な部分のみを紹介します

こんな方を想定しています

- とりあえずPerlの雰囲気を知りたい
- 人が書いたPerlを読めるようになりたい

Perl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

   Perlスクリプトファイルは拡張子を「.pl」にします

2. perlコマンドで実行

```
$ perl Perl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「perl」ディレクトリを作成します

```
$ cd ~/Desktop
$ mkdir perl
$ cd perl
```

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」から以下の1ファイルを「perl」に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/peptides_longer_headers.fasta .
```

改行を入れずに続けて入力

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

```
>NP_571718.1|DRERSOX9A
MNLLDPYLKMTDEQEKCLSDAPSPSMSEDSAGSPCPSASGSDTENTRPAENSLLAADGTLGDF
KKDEEDKFPVCIREAVSQVLKGYDWTLVPMPVRVNGSSKNKPHVKRPMNAFMVWAQAARRKLA
DQYPHLHNAELSKTLGKLWRLLNEVEKRPFVEEAERLRVQHKKDHPDYKYQPRRRKSVKNGQS
ESEDGSEQTHISPNAIFKALQQADSPASSMGEVHSPSEHSGQSQGPPTPPTTPKTDTQPGKAD
LKREARPLQENTGRPLSINFQDVDIGELSSDVIETFDVNEFDQYLPPNG
                    :
```

ID行

配列行

本講義の達成目標

以下の作業をPerlスクリプトで実行できるようになります

「FastaのID行を変更したり、
アミノ酸の出現頻度を数えたりできる」

Perlの記載方法

- 値を出力するにはprintを実行します
- 文字列はダブルクォートかシングルクォートで囲みます
- 行の末尾に;をつけます

```
print "Hello!";
```

- 全角記号・全角空白は使わないでください

実習１

次のPerlスクリプト・perl1.plを書いて実行してみましょう

Hello!と出力するPerlスクリプトです

```
$ gedit perl1.pl
```

perl1.plにこの1行を書いて保存します

```
print "Hello!";
```

```
$ perl perl1.pl
```

改行

PerlのprintはシェルスクリプトのechoとUnixと違って最後が改行されないんだ！

改行したい場合は明示的に改行コードを書く必要があります

```
print "Hello!¥n";
```

¥n：改行コード

「¥」は**バックスラッシュ** \n です
キーボードの「¥」を打ってください

質問

では、Bye!と出力するには
どう変更すればよいでしょう？

解答

実行内容を変えればいいですね

```
print "Hello!¥n";
```

```
print "Bye!¥n";
```

ここで「変数」を使うとスマートです

変数

シェルスクリプト同様、Perlでも「変数」を使うことができます

- 「my $変数名=値;」と書くと、変数に値を代入できます
- 「$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

変数

「my」の話

とりあえず、最初に変数が出てくるときにはmyをつけると覚えてください

myで定義した変数は、定義したスコープ内でのみ有効です

スコープについては後でご紹介します

実習2

次のPerlスクリプト・perl2.plを書いて実行してみましょう

```
$ cp perl1.pl perl2.pl
$ gedit perl2.pl
```

perl2.plを以下のように変更して保存します

```
my $message="Bye!¥n";
print $message;
```

代入の=の前後に
半角空白が入ってもOKです

```
my $message = "Bye!¥n";
```

```
$ perl perl2.pl
```

難易度：★

クイズ

実行結果は
どうなりますか？

Q1.pl

```
my $str = "Amelieff's blog";
print "$str¥n";
```

```
$ perl Q1.pl
```

A

```
Amelieff's blog
```

C

```
Amelieff s blog
```

B

```
Amelieff
's blog
```

D

```
Amelieff
s blog
```

クイズ

正解は、**A**！！

```
Amelieff's blog
```

難易度：★★

クイズ

実行結果は
どうなりますか？

Q2.pl

```
my $str = "Amelieff's blog";
print '$str¥n';
print "$str¥n";
```

```
$ perl Q2.pl
```

A

```
Amelieff's blogAmelieff's blog
```

B

```
$str¥n
Amelieff's blog
```

C

```
Amelieff's blog
Amelieff's blog
```

D

```
$str¥nAmelieff's blog
```

クイズ

正解は、**D**！！

Q2.pl

```
my $str = "Amelieff's blog";
print '$str¥n';
print "$str¥n";
```

```
$ perl Q2.pl
```

```
$str¥nAmelieff's blog
```

```
print '$str¥n';
```

シングルクォート内の
変数は展開されない

```
print "$str¥n";
```

ダブルクォート内の
変数は展開される

値がたくさんある時

あなたは小学校の先生です
クラス40名のテストの平均点をPerlで
計算してみようと思いましたが…

```
my $seito1 = 65;
my $seito2 = 90;
my $seito3 = 78;
   :
my $seito40 = 70;
```

入力するだけで
大変！

値がたくさんある時

「　配列　」を使うとたくさんの値をまとめて扱うことができます

変数（単体）

配列（複数）

複数の値を
まとめて扱えるので便利

配 列

配列は複数の値を1つの名前でまとめたものです

配列に値を入れるには

　my @配列名＝(値,値,...);

と書きます

```
my @seito = (65,90,78, ..., 70);
```

配 列

配列から値を取り出すには

$配列名[数字] と書きます

↑これを「 添字 」と呼びます

```
my @seito = (65,90,78, ..., 70);
print $seito[1], "¥n";
```

配列

添字は0から始まります

先頭の値＝添字0

2番目の値＝添字1

：

```
my @seito = (65,90,78, ..., 70);
print $seito[1], "¥n";
```

添字1＝2番目の値＝90

実習３

次のPerlスクリプト・perl3.plを書いて実行してみましょう

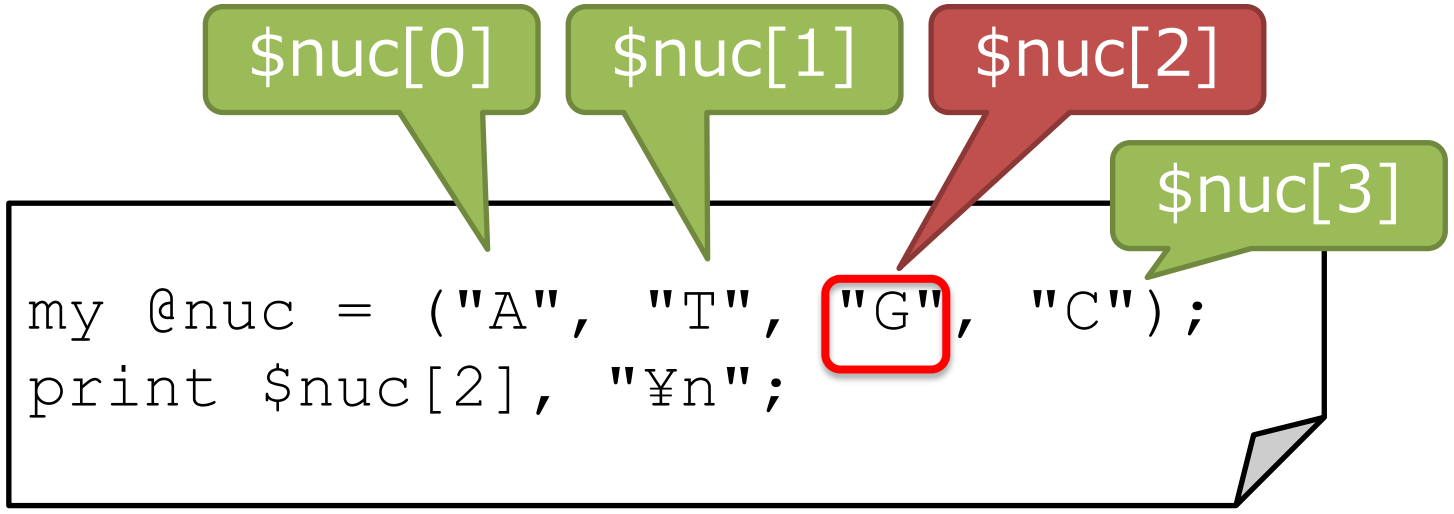
```
$ gedit perl3.pl
```

```
my @nuc = ("A", "T", "G", "C");
print $nuc[2], "¥n";
```

```
$ perl perl3.pl
```

実習３・解答

添字が2の値＝「G」が出力されます



クイズ

実行結果はどうなりますか？

Q3.pl

```
my @pig = ("boo", "foo", "woo");

my $str = $pig[2] . $pig[1] . $pig[0];
 # . は変数を連結して文字列にします

print $str, "¥n";
```

```
$ perl Q3.pl
```

難易度：★

A boowoofoo

B woofooboo

C boofoowoo

D foowooboo

クイズ

正解は、**B**！！

Q3.pl

```
my @pig = ("boo", "foo", "woo");

my $str = $pig[2] . $pig[1] . $pig[0];
 # . は変数を連結して文字列にします

print $str, "¥n";
```

```
$ perl Q3.pl
```

B

```
woofooboo
```

配列に値を追加する

以下のような方法があります

1. 配列[添字]=値
2. pushを使う

```
my @nuc = ("A", "T", "G", "C");
$nuc[4] = "N";
push @nuc, "a";
```

5番目に「N」が
6番目に「a」が
追加されます

配列の要素数を調べる

以下のような方法があります

1. 配列を数値に変換する
2. $#配列 + 1 を計算する

```
my $seito_su_1 = int(@seito);
my $seito_su_2 = $#seito + 1;
```

どちらも値は40になる

配列と文字列を相互に変換する

配列を結合して文字列にする

join（"結合に使う文字", 配列）

```
my @array1  = ('Are', 'you', 'fine?');
my $string1 = join('-', @array1);
print $string1;
```

「Are-you-fine?」
と表示されます

文字列を分割して配列にする

split（/分割に使う文字/, 文字列）

```
my $string2 = 'Tokyo,Japan';
my @arrary2 = split(/,/, $string2);
print $array2[0], "¥n";
```

「Tokyo」
と表示されます

コマンドライン引数

Perlでも外から値を
与えることができるの？

Perlでは、コマンドラインからの引数を@ARGVという配列で受け取ります

- $ARGV[0]に1番目の値が、$ARGV[1]に2番目の値が（以下同様）入ります
- ※空白が値の区切りとみなされます

実習４

次のPerlスクリプト・perl4.plを書いて実行してみましょう

```
$ gedit perl4.pl
```

```
print "Num: ", int(@ARGV), "¥n";
print "3rd: ", $ARGV[2], "¥n";
```

```
$ perl perl4.pl Pink Red Blue
```

値は何でもいいので、値を3つ以上指定して実行してください

実習４・解答

```
print "Num: ", int(@ARGV), "¥n";
print "3rd: ", $ARGV[2], "¥n";
```

```
$ perl perl4.pl Pink Red Blue
Num: 3
3rd: Blue
```

値が3未満だと$ARGV[2]は未定義になります

```
$ perl perl4.pl Pink Red
Num: 2
3rd:
```

値がたくさんあって
各データに名前をつけたい時

ふたたび、あなたは小学校の先生です
クラス40名の誕生日をPerlで管理したい
と思います

Aさんは5月10日、
Bさんは2月28日、...

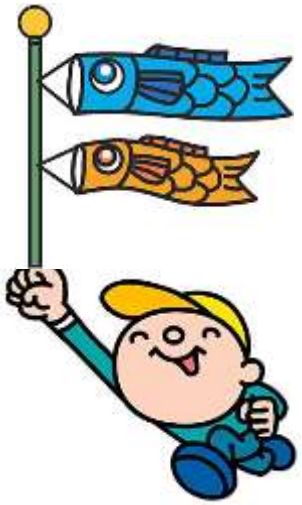
```
my @birth = ("0510", "0228", ... );
```

配列では駄目だ！日付が誰の誕生日かわからない

値がたくさんあって
各データに名前をつけたい時

「ハッシュ」を使うと、各データの名前（キー）と値を対で入力できます

例）Aさん　→　5月5日
　　キー　　値



ハッシュ

ハッシュに値を入れるには、

　my %ハッシュ名=(キー=>値);

と書きます

```
my %birth = ("A"=>"0505", "B"=>"0228",
 ... );
```

Aさんは5月5日、
Bさんは2月28日、...

ハッシュ

ハッシュから値を取り出すには、
$ハッシュ名{キー}　と書きます

```
my %birth = ("A"=>"0505", "B"=>"0228",
  ... );

print $birth{"B"}, "¥n";
```

0228が出力されます

実習5

次のPerlスクリプト・perl5.plを書いて実行してみましょう

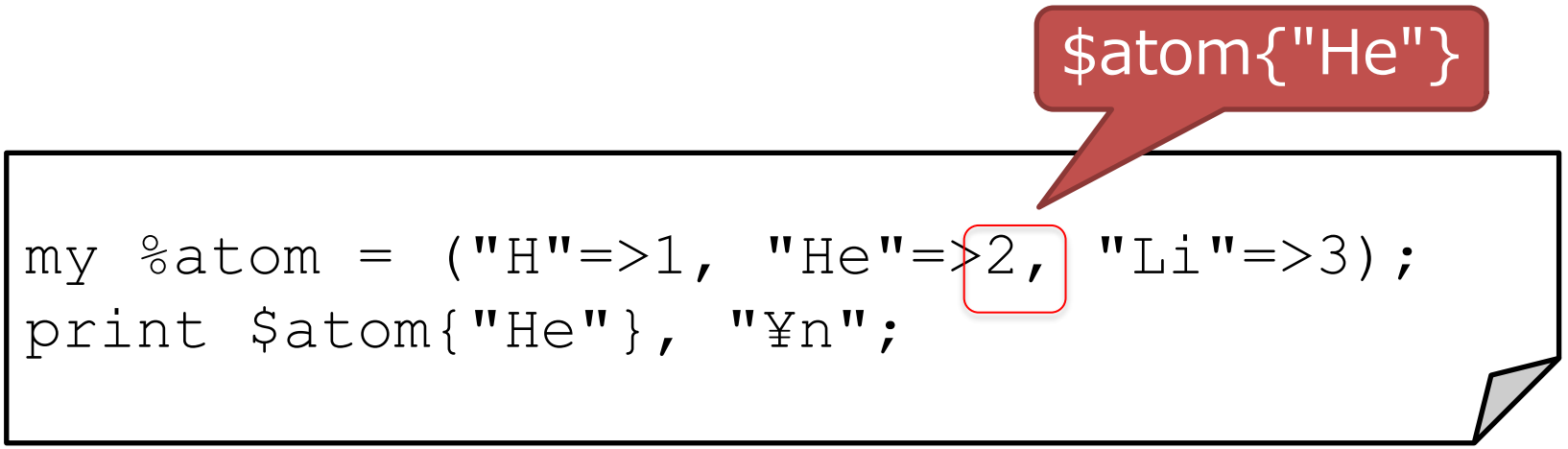
```
$ gedit perl5.pl
```

```
my %atom = ("H"=>1, "He"=>2, "Li"=>3);
print $atom{"He"}, "¥n";
```

```
$ perl perl5.pl
```

実習５・解答

キー「He」の値＝「2」が出力されます



ハッシュにキーと値を追加する

$ハッシュ名{キー}＝値 を実行します

```
my %atom = ("H"=>1, "He"=>2, "Li"=>3);
$atom{"Fe"} = 26;
$atom{"Ca"} = 20;
```

キー「Ca」の値として20、
キー「Fe」の値として26が
入力されます

配列と違ってハッシュは添字でアクセスしないため
最初に入力する順番には意味がありません

配列とハッシュの違い

配列のイメージ

配列は、複数のデータに
端から順にアクセスしたい
場合に向く

ハッシュのイメージ

ハッシュは、該当データに
ピンポイントにアクセス
したい場合に向く

質問

シェルスクリプトのように、
Perlでも条件付き処理や
繰り返し処理が行えるの？

Perlでも条件付き処理や繰り返し処理が可能です

- シェルスクリプトと書き方が似ていますが
  微妙に異なるので混乱しないようにしましょう

条件付き処理

if-elsif-else
構文を使います

```
if(条件1){
  ～条件1を満たした時の処理～
}
elsif(条件2){
  ～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～
}
```

条件付き処理

シェルスクリプトとPerlの違う点を探してみましょう

Perl

```
if(条件1){
 ～条件1を満たした時の処理～
}
elsif(条件2){
 ～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～
}
```

シェルスクリプト

```
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
```

条件付き処理

シェルスクリプトの
比較演算子と混同しない
ようにしましょう

Perlの比較演算子

数値の比較演算子

A == B	A＝Bなら
A != B	A≠Bなら
A < B	A＜Bなら
A <= B	A≦Bなら
A >= B	A≧Bなら
A > B	A＞Bなら

文字列の比較演算子

A eq B	AとBが 同じなら
A ne B	AとBが 異なれば

条件付き処理

複数の条件を指定する場合の書き方

	Perl	【参考】シェルスクリプト
条件1　AND　条件2	条件1 && 条件2	条件1 -a 条件2
条件1　OR　条件2	条件1 || 条件2	条件1 -o 条件2
条件1　でなければ	！条件1	！条件1

実習6

次のPerlスクリプト・perl6.plを書いて
コマンドライン引数にいろいろな数字を
指定して実行してみましょう

```
my $i = $ARGV[0];
if($i >= 10){
  print "$i is equal to or larger than 10¥n";
}
else{
  print "$i is smaller than 10¥n";
}
```

```
$ perl perl6.pl 6
$ perl perl6.pl 11
```

実習６・解答

コマンドライン引数の値により結果が変わります

```
$ perl perl6.pl 6
6 is smaller than 10

$ perl perl6.pl 11
11 is equal to or larger than 10
```

クイズ

実行結果は
どうなりますか？

```
$ perl Q4.pl
```

Q4.pl

```
my $time=13;
if ($time < 12 ){
    print "Good morning¥n";
}
elsif ($time <18 ){
    print "Hello¥n";
}
else {
    print "Good evening¥n";
}
```

難易度：★

A

```
Good morning
```

B

```
Hello
```

C

```
Good evening
```

D

エラーになる

クイズ

正解は、**B**！！

```
Hello
```

Q4.pl

```
my $time=13;
if ($time < 12 ){
    print "Good morning¥n";
}
elsif ($time < 18 ){
    print "Hello¥n";
}
else {
    print "Good evening¥n";
}
```

1. ifの条件を満たさないので、if文は実行されません
2. elsifの条件を満たすのでelsif文が実行されます
3. elsif文が実行されたので、else文は実行されません

繰り返し処理

指定した条件の間、同じ処理を繰り返すことができます

Perlの繰り返し処理にはwhile、for

などの書き方があります

繰り返し処理・while

条件を満たす間繰り返す

while(繰り返し条件){処理内容}

```
my @gene_arr = ("Oct4","Sox2","Kif4","c-Myc");

my $i=0;
while($i<int(@gene_arr)){
   print $gene_arr[$i], "¥n";
   $i = $i + 1;
}
```

この行がないとずっと$iが0のままなので、
実行が終わらなくなります（無限ループ）

出力結果

```
Oct4
Sox2
Kif4
c-Myc
```

繰り返し処理・for

配列の各要素に対して繰り返す

for $変数(@配列){処理内容}

```
my @gene_arr = ("Oct4","Sox2","Kif4","c-Myc");

for my $gene(@gene_arr){
  print $gene, "¥n";
}
```

@gene_arrの先頭から
値を一つずつ取り出して
変数$geneに入れてprint

出力結果

```
Oct4
Sox2
Kif4
c-Myc
```

繰り返し処理・for【別の書き方】

変数の値の変化に応じて繰り返す

for(変数の初期値; 繰り返し条件; 変数増分){処理内容}

```
my @gene_arr = ("Oct4","Sox2","Kif4","c-Myc");

for(my $i=0; $i<int(@gene_arr); $i=$i+1){
  print $gene_arr[$i], "¥n";
}
```

$iが@gene_arrの要素数より小さい間、@gene_arrの各値をprint

出力結果

```
Oct4
Sox2
Kif4
c-Myc
```

実習7

次のPerlスクリプト・perl7.plを書いて実行してみましょう

```
$ cp perl3.pl perl7.pl
$ gedit perl7.pl
```

perl7.plを以下のように変更して保存します

```
my @nuc = ("A", "T", "G", "C");
my $i = int(@nuc);
while($i > 0){
  $i = $i - 1;
  print $nuc[$i], "¥n";
}
```

余裕のある方は
同じ処理をforでも
書いてみてください

実習７・解答

添字3→2→1→0の順に@nucの値がprintされます

```
$ perl perl7.pl
C
G
T
A
```

```
my @nuc = ("A", "T", "G", "C");
my $i = int(@nuc);
while($i > 0){
   $i = $i - 1;
   print $nuc[$i], "¥n";
}
```

実習７・解答

違う書き方もできます

```
my @nuc = ("A", "T", "G", "C");
my $i = int(@nuc);
while($i > 0){
  $i = $i - 1;
  print $nuc[$i], "¥n";
}
```

「 $#nuc + 1 」

$i--;

$i--は$i=$i-1と
$i++は$i=$i+1と
同じ意味になります

実習７・解答

同じ処理をforで書いた場合

```
for(my $i=int(@nuc)-1; $i>=0; $i=$i-1){
  print $nuc[$i], "¥n";
}
```

```
for my $base(reverse @nuc){
  print $base, "¥n";
}
```

reverse:
配列を逆順にする

どちらでも同じ結果になります

繰り返し処理でハッシュにアクセス

forを使う場合

```
my %atom = ("H"=>1, "He"=>2, "Li"=>3);
for my $key(keys %atom){
  $val = $atom{$key};
  print $key, ":", $val, "¥n";
}
```

whileを使う場合

```
my %atom = ("H"=>1, "He"=>2, "Li"=>3);
while(my ($key, $val) = each %atom){
  print $key, ":", $val, "¥n";
}
```

繰り返し処理

繰り返し処理の中で次の要素に
スキップするにはnextを使います

繰り返し処理

繰り返し処理の中で次の要素に
スキップするにはnextを使います

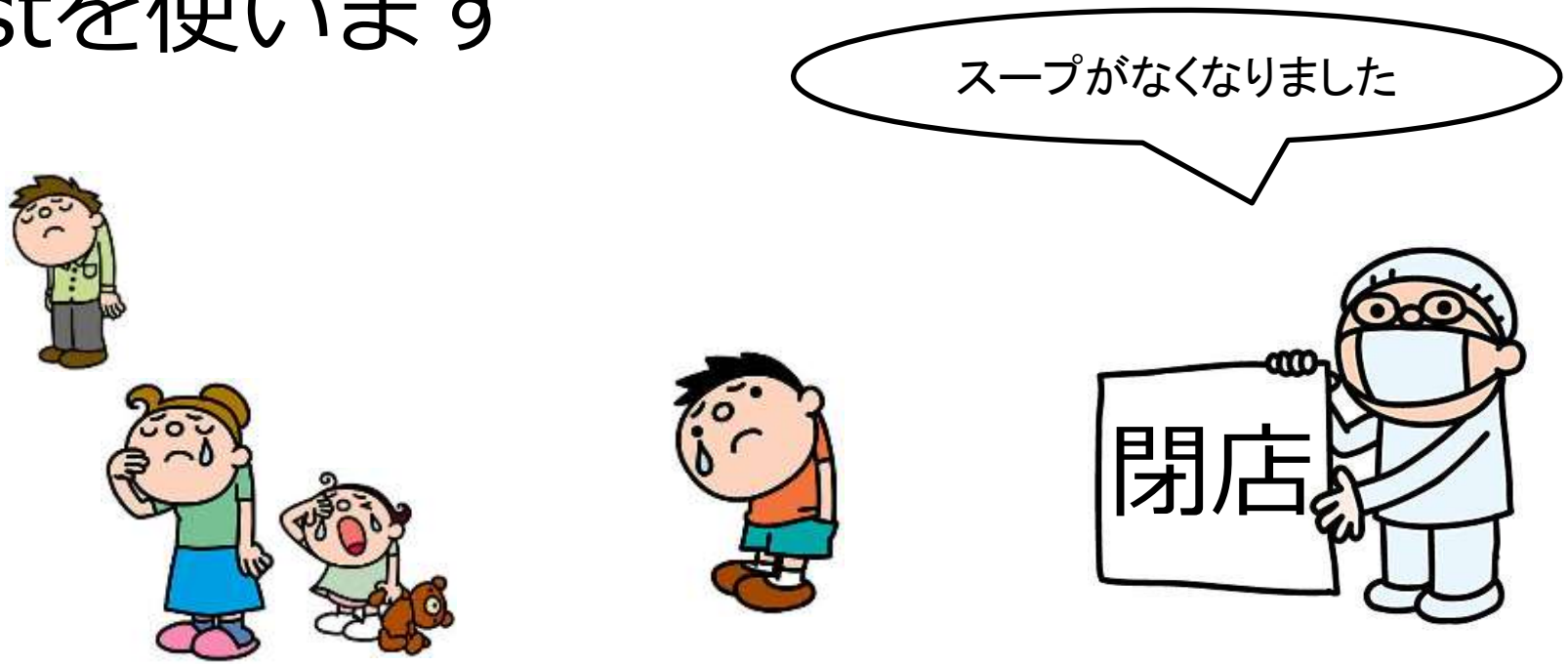
```
for(my $i=0; $i<5; $i++){
   if($i == 2){
      next;
   }
   print $i, "¥n";
}
print "END¥n";
```

出力結果

```
0
1
3
4
END
```

繰り返し処理

繰り返し処理自体を中止するには
lastを使います



繰り返し処理

繰り返し処理自体を中止するには
lastを使います

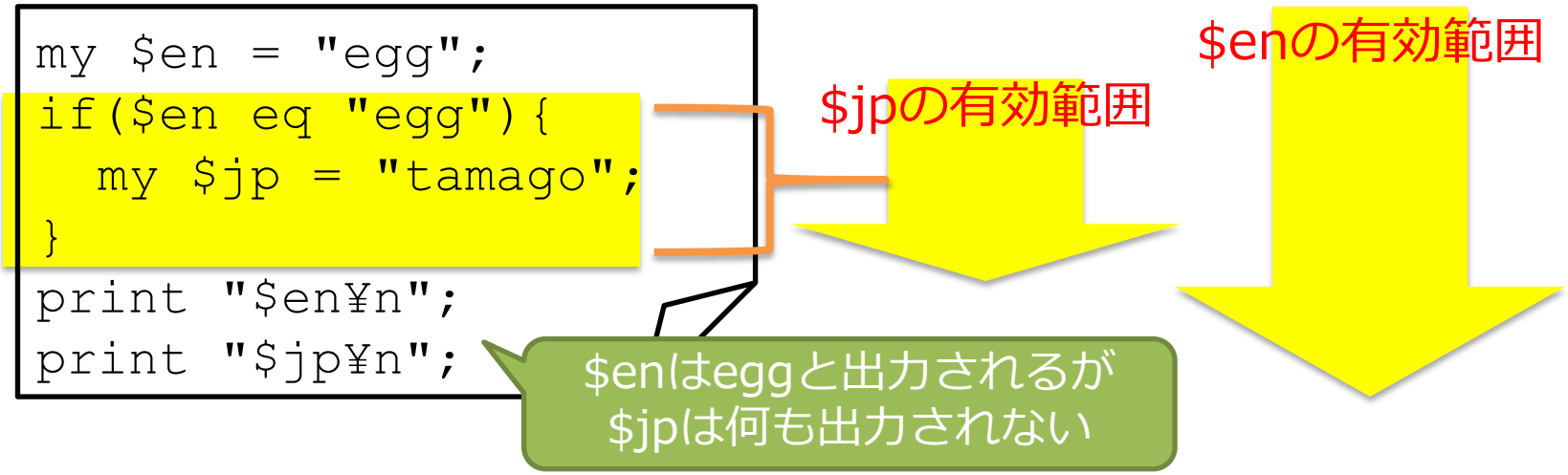
```
for(my $i=0; $i<5; $i++){
  if($i == 2){
    last;
  }
  print $i, "¥n";
}
print "END¥n";
```

出力結果

```
0
1
END
```

スコープ

{}で囲んだ範囲をスコープと呼びます
スコープ内で定義した変数はそのスコープでのみ有効です（ローカル変数と呼びます）



不満

配列やハッシュを使っても
大量のデータを手入力するのは大変！
ファイルから読み込めるといいのに

データをファイルから読み込んだり、
ファイルに書き出したりできます
ファイル入出力には「ファイルハンドル」を
使います

ファイル入出力

ファイルから1行ずつ読み込んで
処理するには次のように書きます

```
my $file = "input.txt";

open my $fh, "<", $file or die;
while(my $line = <$fh>){
  chomp($line);
  if($line eq "abc"){
    print "$line¥n";
  }
}
close $fh;
```

変数fileにファイル名を入力

ファイルハンドル$fhを読込用で開く

$fhから1行ずつ読み込んで変数$lineに入れる

$line末尾の改行コードを除去する

$lineが"abc"なら改行コードを付与して出力

ファイルハンドル$fhを閉じる

ファイル入出力

ファイルから1行ずつ読み込んで
処理するには次のように書きます

```
my $file = "input.txt";

open my $fh, "<", $file or die;
while(<$fh>){
  chomp($_);
  if($_ eq "abc"){
    print "$_¥n";
  }
}
close  $fh;
```

同じ意味

```
chomp;
```

ファイル読み込み結果を
代入する変数を省略すると
$_という特殊な変数に入ります

さらに、
chompの引数を
省略すると
$_が処理されます

ファイル入出力

ファイルに書き出すには次のように書きます

```
my $file = "output.txt";

open my $fh, ">", $file or die;
print $fh "test¥n";
close  $fh;
```

変数fileにファイル名を入力

$fhを書出用で開く

$fhに文字列"test"を改行をつけて出力

$fhを閉じる

バグを見つけやすくする

Perlは制約が緩い言語のため、
バグ（プログラムの誤り）を
見つけにくいことがあります
以下を記述すると、プログラム実行時に
文法や変数の定義をチェックできます

```
use strict;
use warnings;
```

実習８　１/２

次のPerlスクリプト・perl8-1と
perl8-2.plを書いて実行してみましょう

```
$ cp perl2.pl perl8-1.pl
$ gedit perl8-1.pl
```

perl8-1.plを以下のように変更して保存します

```
$message="Bye!¥n";
print $message;
```

1行目先頭のmyを削除

```
$ perl perl8-1.pl
```


次ページにつづく

実習８　２/２

次のPerlスクリプト・perl8-1と
perl8-2.plを書いて実行してみましょう

```
$ cp perl8-1.pl perl8-2.pl
$ gedit perl8-2.pl
```

perl8-2.plを以下のように変更して保存します

```
use strict;
use warnings;
$message="Bye!¥n";
print $message;
```

use ~ を追記

```
$ perl perl8-2.pl
```

エラーが出るようになります

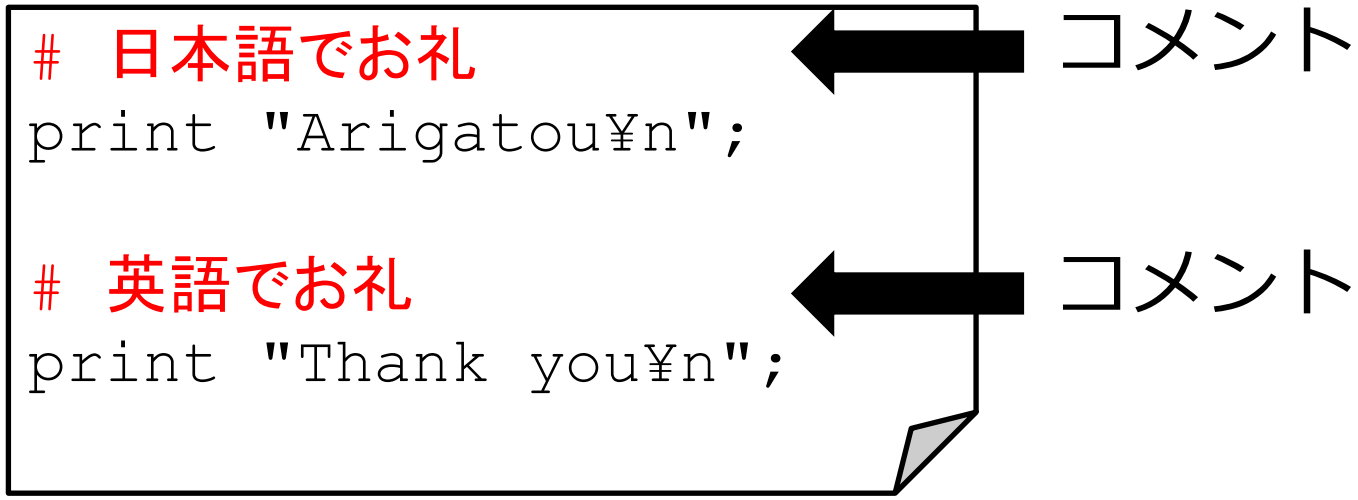
実習８・解答

perl8-1

```
$ perl perl8-1.pl
Bye!
$ perl perl8-2.pl
Global symbol "$message" requires explicit package
name at perl8.pl line 3.
Global symbol "$message" requires explicit package
name at perl8.pl line 4.
Execution of perl8.pl aborted due to compilation
errors.
```

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Perlスクリプトである
ことが明示的になります

```
#!/usr/bin/perl
```

これにより、perlコマンドをつけずファイル
単体で実行できるようになります

```
$ chmod a+x perl9.pl
$ ./perl9.pl
```

chmod a+x：実行権限をつける

正規表現

文字列のパターンを表現する方法

Perlでは、正規表現を//で定義します

```
my $str = "bioinfo";
if($str =~ /info/){
 print "match¥n";
}
else{
  print "not match¥n ";
}
```

変数$strに「info」という文字列が含まれていれば「match」と出力、そうでなければ「not match」と出力

メタ文字の例

正規表現では以下のような記号を使えます

正規表現	意味
[atgc]	a, t, g, cのいずれか
¥w	英数字
¥W	英数字以外
¥d	数字
¥D	数字以外
¥s	空白・タブ・改行
¥S	空白・タブ・改行以外
.	任意の1文字
^	行頭
$	行末

いろいろなマッチングパターン

正規表現	文字列	マッチしたか？
/bioinfo/	bioinformatics	Yes
/bioinfo/	informatics	No
/^bioinfo/	bioinformatics	Yes
/bioinfo$/	bioinformatics	No
/[atgc]/	bioinformatics	Yes
/[a-z]/	bioinformatics	Yes
/bio./	bioinfo	Yes
/bio./	bio	No
/bi*o/	biiio, bio, bo	Yes
/bi*o/	aio	No
/bi+o/	biiio,bio	Yes
/bi+o/	bo, dio	No

a*　aの0回以上の繰返し
a+　aの1回以上の繰返し

マッチした箇所の取り出し

正規表現パターン中の()で囲った箇所を$1, $2, ...で取り出すことができます

```
my $str = "Amelieff";

if($str =~ /(Ame*)li(ef+)/)
  print $1, "¥n";
  print $2, "¥n";
}
```

```
Ame
eff
```

1番目の()にマッチした文字列が$1に
2番目の()にマッチした文字列が$2に
...（以下同じ）入る

正規表現を用いた置換

~s/正規表現パターン/置換文字列/オプション

```
my $str1 = "genome,proteome";
$str1 =~ s/ome/omics/;
print "$str1¥n";
```

```
genomics,proteome
```

```
my $str2 = "genome,proteome";
$str2 =~ s/ome/omics/g;
print "$str2¥n";
```

```
genomics,proteomics
```

変数$str1中に最初に登場した「ome」という文字列が「omics」に置き換わる

変数$str2中に登場した全ての「ome」という文字列が「omics」に置き換わる

gオプションをつけるとマッチしたすべてについて置換

正規表現を用いた置換

~s/正規表現パターン/置換文字列/オプション

```
my $str3 = "I sing a song.";
$str3 =~ s/I/You/;
print "$str3¥n";
```

```
You sing a song.
```

変数$str3中に登場した全ての「I」という文字列が「You」に置き換わる

```
my $str4 = "I sing a song.";
$str4 =~ s/I/You/i;
print "$str4¥n";
```

```
You sYoung a song.
```

変数$str4中に登場した全ての「IまたはI」という文字列が「You」に置き換わる

iオプションをつけると
大文字個別を区別せず置換

難易度：★

クイズ

実行結果はどうなりますか？

```
$ perl Q5.pl
```

Q5.pl

```
my $org = "Gallus gallus gallus";
$org =~ s/gallus/gorilla/g;
print $org, "¥n";
```

A

```
Gallus gallus gallus
```

C

```
Gorilla gorilla gorilla
```

B

```
Gallus gorilla gorilla
```

D

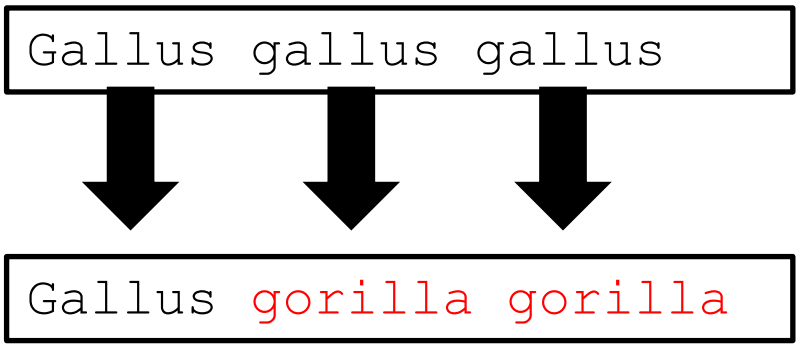
```
gorilla gorilla gorilla
```

クイズ

正解は、**B**！！

```
Gallus gorilla gorilla
```

- iがついていないので、大文字小文字が区別されます（gallus→マッチ、Gallus→マッチしない）
- gがついているので、全てのgallusがgorillaに置換されます



クイズ

実行結果はどうなりますか？

```
$ perl Q6.pl
```

Q6.pl

```
my $gene = "hg19;chr12;KRAS";
$gene =~ s/*;//g;
print $gene, "¥n";
```

A

```
chr12;KRAS
```

B

```
KRAS
```

C

```
chr12hg19KRAS
```

D

エラーになる

クイズ

正解は、**D**！！

エラーになる

```
Quantifier follows nothing in
regex; marked by <-- HERE in m/*
<-- HERE ;/ at quiz.pl line 2.
```

Perlの正規表現では「*」は「直前の文字の0回以上の繰り返し」を意味するため、「*」の前には何らかの文字が必要です

実習 9

必要スキル
- 変数
- 配列
- 条件付き処理
- 繰り返し処理
- ファイル入出力
- 正規表現

次のPerlスクリプト・perl9.plを書いて実行してみましょう

1. peptides_longer_headers.fastaを読込用で開いて1行ずつ読み込んで改行コードを削除する
2. 読み込んだ行がID行（「>xxx|yyy」形式）なら、「xxx|yyy」に改行コード（¥n）を付けて出力
3. それ以外の行なら、内容を変えずに改行コード（¥n）を付けて出力

実習 9

```
#!/usr/bin/perl

use strict;
use warnings;
use autodie;  # or dieを勝手にやってくれます

my $file = "peptides_longer_headers.fasta";

open my $fh, "<", $file;
while(<$fh>){
  chomp;
  if($_ =~ /^>(.+)$/){
    print ">$1¥n";
  }
  else {
    print $_, "¥n";
  }
}
close $fh;
```

最終課題

必要スキル
- 変数
- 配列
- 条件付き処理
- 繰り返し処理
- ファイル入出力
- 正規表現
- コマンドライン引数
- ハッシュ

次のPerlスクリプト・
perl10.plを書いて実行してみましょう

```
$ cp perl9.pl perl10.pl
$ gedit perl10.pl
```

1. コマンドライン引数で指定したファイルを読込用で開いて、1行ずつ読み込んで改行コードを削除する
2. 読み込んだ行がID行以外なら、一文字ずつ区切って各アミノ酸の出現頻度をハッシュでカウントする
3. カウント結果を出力
4. コマンドライン引数にpeptides_longer_headers.fastaを与えて実行

最終課題

```
:
(シバン及びuse~)
:

my $file = $ARGV[0];
my %aaCount;     ←ハッシュの定義をしている

open my $fh, "<", $file;
while(<$fh>){
  chomp;
  if($_ !~ /^>/){
    my @aaArr = split(//, $_);
       for my $aa(@aaArr){
         $aaCount{$aa} ++;
       }
  }
}
close $fh;

while(my ($aa, $count) = each %aaCount){
  print $aa, ":", $count, "¥n";
}
```

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

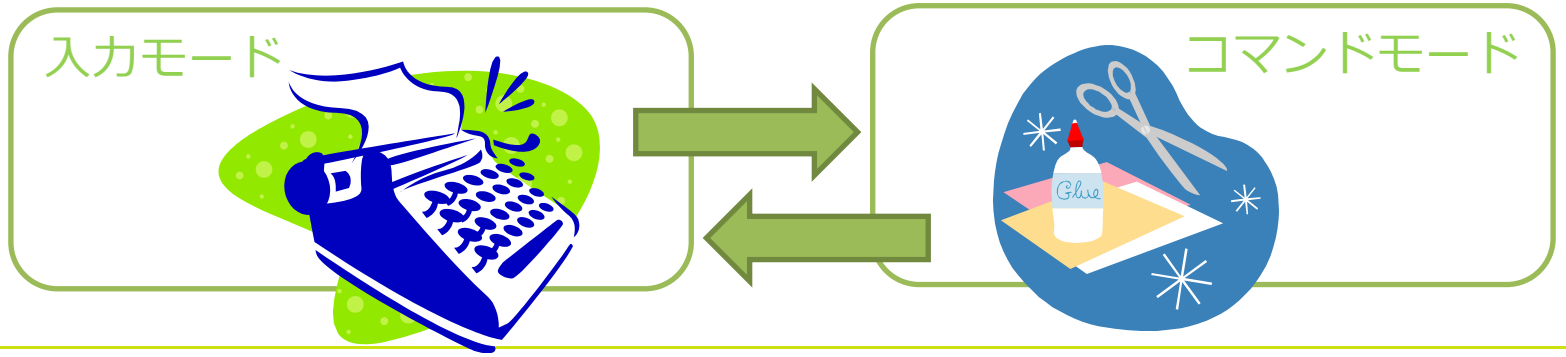
CentOSにはデフォルトでviというCUIエディタが入っています

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）