

## Genbank file format

---

.faa  
.fna  
.ffn  
.ptt  
.gbk  
.asn

Look at the manual under **manual** on our course home page.

## About dot files (It is not about Perl)

---

```
$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
UNAME='uname';
echo ---- I am using $UNAME ---;

# make sure at least one blank in the 'if test [ ]'; I hate it!
if [ $UNAME = AIX ]; then
    PATH=$PATH:$HOME/bin:/bioinfo/spbin:.
else
    PATH=$PATH:$HOME/bin:/bioinfo/solarbin:.
fi
```

## Getopt::Command Option Handling

---

1. Read documentation using a command `perldoc Getopt::Long`
2. Three types are supported (see the example below)
  - `s` for string
  - `i` for integer
  - `f` for real number

```
$ cat extract_subseqs.pl
use Getopt::Long;
GetOptions ('ranges=s' => \$Rfnm,
            'seqs=s' => \$Sfnm,
            'seqids' => \$Seqidtype,
            'dna' => \$DNAseq,
            'rc' => \$RC,
            'protein' => \$Proteinseq
);
```

## Variable Syntax

---

Scalar	<code>\$var</code> \\
Array	<code>@var;</code> <code>\$var[0]</code> , <code>\$var[1]</code> , ... \\
Hash	<code>%var;</code> <code>\$var{"key"}</code> \\
Subroutine	<code>&amp;subroutine;</code> <code>&amp;subroutine(argument)</code> \\
Typeglob	<code>*name;</code> <code>everything</code> \\

## Comparison operators

---

Different comparison operators should be used for numeric and string variables.

==	eq
!=	ne
<	lt
>	gt
<=	le
>=	ge
<=>	cmp

# Example1: below is wrong.

```
$str1 = "this";
```

```
$str2 = "that";
```

```
print "The two are same.\n" if ($str1 == $str2);
```

```
@nvars = (1, 12, 9, 8, 100);
```

```
@sorted = sort {$a <=> $b} @nvars;
```

```
print "@sorted \n"; # print an array var?
```

```
# in an desceding order, string comparisons?
```

## Special Variables

---

`$1 $2 etc` : texts matched by reg-expressions

`$a $b` : a pair of values to be compared; `sort { $a <=> $b } @an_array`

`$_` : the default input and pattern search space.

`@_ @ARGV` : the argument list passed to the subroutine.

`@ARGV` :

`%ENV` : current environment vars; `$ENV{PATH} = "/mybin:". $ENV{PATH};`

`%INC @INC` : where to look at by `do FILE`, `require`, or `use`.

`autoflush HANDLE EXPR, $AUTOFLUSH, $|` : forces a buffer flush after every print

`$$, $PID, $PROCESS_ID`: a process id

`$0, $PROGRAM_NAME`: the name of the program

## List and Array

---

A list of values, enclosed by `()`, can be assigned to an array or a hash variable, but *not* to a scalar variable.

A list of values enclosed by `[]` should be assigned to a scalar variable, *reference to an array*, but *not* to an array.

```
$a_ref_to_an_array = [1,2,3,4,5];
@an_array = (1,2,3,4,5);
$a_ref_to_an_array = \@an_array;
```

```
$length = @an_array;
print "The length of this array is $length.\n";
print "The subscript of the last entry is $#an_array.\n";
```

```
$a_ref_to_an_array = (1,2,3,4,5); # wrong
@an_array = [1,2,3,4,5]; # wrong
```

```
%hash = (one, 1, two, 2, three, 3); # correct but not recommended
%hash = {one=>1, two=>2, three=>3}; # the same as the above but bet
```

## Pointers, lvalue, and rvalue

---

```
@a = (1,2,3,4,5);
```

```
$ptr = \@a;
```

```
print "The fourth element is $ptr->[3]. \n";
```

```
lvalue = rvalue;
```

## Loops and Control

---

```
my %ahash = (1 => one, 3 => three, 2 => two);
```

```
@values = (values %ahash);  
print "Values: @values \n";
```

```
@keys = sort (keys %ahash);  
print "KEYS: @keys \n";
```

```
while (($k, $v) = each %ahash ) {  
    print "($k, $v) \n";  
}
```

## Loops and Control

---

for, foreach, while, unless  
last [label], next [label], redo [label]

```
@we = qw/memo memo skip dennis terminate bob/;
foreach $me (@we) {
    next if ($me eq "skip");
    if ($me eq "terminate") { last; }

    print "$me \n";
    # redo if ($me eq "sun");
}
```

```
for ($i=0; $i<100; $i++) {
    if ($we[$i] == "sun") { # something wrong!
        $i --; # never modify the index!
    }
    print "$we[$i] \n";
}
```

## File Handle

---

```
$Fname = "anyname";
open (FH, "<$Fname") or die "open $Fname failed.";
open (FH, ">$Fname") or die "open $Fname failed.";
open (FH, ">>$Fname") or die "open $Fname failed.";
open (FH, "|$Command") or die "open $Command failed.";
open (FH, "$Command|") or die "open $Command failed.";

open (FH, $Fname) or die "open $Fname failed.";
<FH>;          # $_ reads in a line from the file.
$aline = <FH>; # $aline reads in a line from the file.
while(<FH>) {  # $_ reads in a line from the file.
while($aline=<FH>) { # $aline reads in a line from the file.

@alllines = <FH>; # The entire content is read in to @alllines.

close FH;      # Always, make sure to close FH.
```

## File Testing and Directory Browsing

---

```
$path = "/home/sunkim/courses/L519/lecture";
opendir (THISDIR, $path) or die "open dir $path failed";
@regular_files = grep { !/^\.\/ } readdir THISDIR;
print @regular_files, " \n"; # context is wrong!
print "@regular_files \n";
closedir THISDIR;
```

```
foreach $file (@regular_files) {
    if (-r $file) {
        print "Yes, $file is readable.\n";
    } else {
        print "No, $file is not readable.\n";
    }
}
```

```
# -r, -w, -x, -e, -d, -l, .....
```

## Subroutines and Parameter Passing

---

```
# argument passing
thisfunc($hi, $lo);

sub
thisfunc {
    ($arg1, $arg2) = @_;
}

# reference
@a = (1, 6, 9);
$all = sum (\@a );
print "Total is $all.\n";

sub
sum {
    my ($aref) = @_; my ($total) = 0;
    foreach (@$aref) { $total += $_ };
    return $total;
}
```

```
# file handle passing
open (F1, "<$fnm") or die "open $fnm failed";
collect_gene_family(\*F1);

sub
collect_gene_family
{
    $fh = $_[0];
    while(<$fh>) { #do something.
    }
}
```

## Regular Expressions

---

```
$var =~ /regular-expression/modifiers  
$var =~ s/pattern1/pattern2/modifiers  
$var =~ tr/pattern1/pattern2/modifiers
```

modifiers: i (case insensitive), g (global), e (expression), x, o

Character classes: \d \D \s \S \w \W

```
# a sample program  
$seq1 = "ATGGACGAT";  
$seq1 =~ reverse tr/ATGC/TACG/;  
print "$seq1 \n";  
  
$seq2 = "GGACGGCA";  
$seq3 = "GGAAACGGCAT";  
@seqs = ($seq1, $seq2, $seq3);  
foreach $this (@seqs) {  
    if ($this =~ /^GG(A{1,3})[~T]+T/) {  
        print "seq '$this' has the pattern.\n";  
    }  
}
```

## A Sample Program for DB File

---

```
use DB_File;
use Getopt::Long;

GetOptions('f=s' => \$File,
          'p=s' => \$Pat);

#$filename = "dbfile";

tie (%hash, "DB_File", $File) or die "cannot open $filename";

$hash{"key1"} = "value1";
$hash{"key2"} = "value2";
$hash{"key3"} = "value3";
$v = $hash{$Pat};
untie %hash;

print "$v \n";
```

## A Sample Program for DB File (Btree)

---

```
use DB_File;
use Getopt::Long;

GetOptions('f=s' => \$File,
           'p=s' => \$Pat);

#$filename = "dbfile";

tie (%hash, "DB_File", "${File}.idx", O_RDWR|O_CREAT, 0666, $DB_BTREE)
    or die "cannot open ${File}.idx";

$hash{"seq1"} = "aaaatggctagagtagagcg";
$hash{"seq2"} = "gggatagaggatag";
$hash{"seq3"} = "cccgtgagagatagagga";
$hash{"seq4"} = "ttttgatagagtaga";

# print key,value in-order traversal.
while( ($k, $v) = each %hash) {
    print "$k => $v \n";
}
untie %hash;
```

## Hint for Homework2

---

```
use Getopt::Long;
GetOptions('seqfile=s' => \$Sfile, 'range=s' => \$Rfile);

my %Sranges;
open (PFH, "$Rfile") or die "$Rfile open failed";
while(<PFH>) {
    chomp;
    s/^\s+//g;
    ($seqid, $start, $end) = split(/\s+/, $_);

    unless (defined($Sranges{$seqid})) {
        my @newa;
        $Sranges{$seqid} = \@newa;
    }
    push(@{$Sranges{$seqid}}, $start, $end);
}
close PFH;
pr_ranges("s1"); pr_ranges("s2"); pr_ranges("s3");

sub pr_ranges
{
    $seqid = shift;
    print "\mRanges for $seqid\n";
    foreach $r (@{$Sranges{$seqid}}) {
        print $r, "\n";
    }
}
```

## A Sample Pattern Searching Program

---

```
use Getopt::Long;

GetOptions('f=s' => \$File,
          'seq=s' => \$Seqid,
          'seqidtype=s' => \$Seqidtype,
          'p=s' => \$Pat);

pr_usage("pattern required: -p \"pattern\"") unless (defined(\$Pat));
pr_usage("seq-file required: -f") unless (defined(\$File));
pr_usage("seq-id required: -seq") unless (defined(\$Seqid));

$Seqid = "$Seqidtype\\|$Seqid" if (defined(\$Seqidtype));

my $thisseq = getseq(\$File, \$Seqid);

#print "$thisseq \n";

# The regexp is progressive matching and case insensitive.
# The () in the regexp is necessary to store the regexp
# match to $1, a special variable in Perl.
# The while loop iterates until a match found.
while($thisseq =~ /($Pat)/gi) {
    printf "found %s at %d \n", $1, pos($thisseq) -1;
}
```

```
open (FH, "<$file") or die "$File open failed";
while(<FH>) { # Alternatively, 'while($_=<FH>) {'
    if ($reading == 1) {
        $_ =~ s/\s+//g;
        $seq .= $_;
    }

    if (/^>/) {
        if (/^\S+$seqid/) {
            # The flag '$reading' indicates that the seqid is found.
            $reading = 1
        } else {
            last if ($reading == 1);
        }
    }
}
close(FH); # be sure to close FH.

return $seq;
}

sub
pr_usage
{
    print STDERR "ERROR: $_[0]\n";
    exit;
}
```