PROGRAM NAME: # fullgffcoords.pl

AUTHOR: **Josep F. Abril** .... jabril@imim.es

LICENSE: **GNU General Public License (GNU-GPL)**

LAST UPDATE: **October 15, 2001**

DESCRIPTION: Given a GFF file in which coords are mapping genomic coords, we want to retrive CDS and amino-acid positions for each group. As a side product, we can obtain the group start and end coords in genomic ones.

Genome Informatics Research Lab
Grup de Recerca en Infomàtica Biomèdica
Institut Municipal d'Investigació Mèdica
Universitat Pompeu Fabra

Contents

List of Tables

List of Figures

`< Id: deploy.nw,v 1.12 2001/10/05 18:30:33 jabril Exp >`

1 Introduction

1.1 Program description

TO DO

- A full description of the program and schemmes of some tricky parts.
- Include examples of input/output files and append a testing section.

1.2 Input

It reads GFF files from genomic annotations (not alignments) and uses grouping field to determine each gene boundaries.

1.3 Output

Basic output is a GFF file similar to input one, to which we have appended, to the end of the grouping field and for every record, the CDS and protein coords for that GFF record.

```
    Field_1 ... Field_8  Grouping;  CDS ori end;  AA ori end;
```

1.4 To Do

▷ [*Section* 1.1, *page* 1]
A full description of the program and schemmes of some tricky parts.

▷ [*Section* **??**, *page* **??**]
Include examples of input/output files and append a testing section.

▷ [*Section* 3.1.1, *page* 4]
Not implemented yet an option to switch on/off reporting execution to STDERR (now by default send logs there but cannot be dissabled).

2 Implementation

2a ⟨*Program Info* 2a⟩≡
```
  my $PROGRAM = 'fullgffcoords.pl';
  my $VERSION = '0.1';
```

2b ⟨*Prog USAGE* 2b⟩≡
```
  $PROGRAM [options] < input_files > output_files
```

2c ⟨*Prog DESC* 2c⟩≡
```
  Retrieving CDS and protein coords from GFF mapped on genomic coords.
```

2d ⟨*Program Description* 2d⟩≡
```
  # #-----------------------------------------#
  # #                         fullgffcoords                        #
  # #-----------------------------------------#
  #
  #  fullgffcoords [options] < input_files > output_files
  #
  #  Retrieving CDS and protein coords from GFF mapped on genomic coords.
  #
  #
  #     Copyright (C) 2001 - Josep Francesc ABRIL FERRANDO
```

2.1 Program outline

2e ⟨*fullgffcoords* 2e⟩≡
```
  ⟨PERL shebang 12a⟩
  #
  # MODULES
  #
  ⟨Use Modules 2f⟩
  #
  # VARIABLES
  #
  ⟨Global Vars 2g⟩
  #
  # MAIN LOOP
  #
  ⟨Main Loop 3a⟩
  #
  # FUNCTIONS
  #
  ⟨Functions 3c⟩
```

2f ⟨*Use Modules* 2f⟩≡
```
  ⟨Use Modules - Benchmark 12c⟩
  ⟨Use Modules - Getopt 3d⟩
```

2g ⟨*Global Vars* 2g⟩≡
```
  ⟨Boolean 12b⟩
  ⟨Counter vars 13e⟩
  ⟨Stderr subs vars 14a⟩
  my ($id,$seq) = ("","");
  my %CmdLineVar = (
                  ⟨cmdline defaults - format 6d⟩
                  ⟨command-line defaults - GFF 7b⟩
                  );
```

3a ⟨*Main Loop* 3a⟩≡
```
  &main();

  exit(0);
```

3b ⟨*messages - program running* 3b⟩≡
```
  'PROG-START'  => "$line$s\n$s Running $PROGRAM\n$s\n".
                   "$s HOST: $host\n".
                   "$s USER: $USER\n".
                   "$s DATE: $DATE\n$s\n$line$s\n" ,
  'PROG-FINISH' => "$s\n$line$s\n$s $PROGRAM FINISHED\n$s\n".
                   "$s TOTAL TIME: \%s\n$line" ,
```

3c ⟨*Functions* 3c⟩≡
```
  ⟨Parsing command line options 3f⟩
  ⟨Main program functions 7d⟩
  ⟨Common PERL subs - Text fill 13c⟩
  ⟨Common PERL subs - Counter 13d⟩
  ⟨Common PERL subs - Min Max 13b⟩
  ⟨Common PERL subs - Benchmark 12e⟩
  ⟨Common PERL subs - STDERR 13f⟩
```

3 Program functions

3.1 Processing command-line options

3d ⟨*Use Modules - Getopt* 3d⟩≡
```
  use Getopt::Long;
  Getopt::Long::Configure qw/ bundling /;
```

3e ⟨*perl requires - Getopt* 3e⟩≡
```
  "Getopt::Long" - processing command-line options.
```

See '`man Getopt::Long`' for further info about this package.

3f ⟨*Parsing command line options* 3f⟩≡
```
  sub parse_cmdline() {
      ⟨looking for STDIN 4a⟩

      $SIG{__WARN__} = sub { &warn('UNKNOWN_CL_OPTION',$T,$_[0]) };
      GetOptions(
                 ⟨command-line options - format 6c⟩
                 ⟨command-line options - GFF 7a⟩
                 ⟨command-line options with exit 5b⟩
                 ) || (&warn('CMD_LINE_ERROR',$T), exit(1));
      $SIG{__WARN__} = 'DEFAULT';

      &report("PROG-START");
      @data_files = ();
      &set_input_file($cmdln_stdin);
      @ARGV = (); # ensuring that command-line ARGVs array is empty

  } # parse_cmdline
```

3g ⟨*warnings - command-line* 3g⟩≡
```
  'UNKNOWN_CL_OPTION' =>
    $Warn."Error trapped while processing command-line:\n".(" "x16)."\%s\n",
  'CMD_LINE_ERROR' =>
    $spl.$spw." Please, check your command-line options!!!\n".$Error."\n".
    $spw." ".("."x12)." Type \"$PROGRAM -h\" for help.\n".$spl,
```

To avoid errors reported when using ’`-`’ as ’`STDIN`’ mark and `GetOptions` to parse command-line parameters, we capture the single dash when present in the command-line arguments list. ’`$cmdln_stdin`’ will be used by ’`&set_input_file`’ function to include the ’`STDIN`’ in the correct ordering.

4a ⟨*looking for STDIN* 4a⟩≡
```
my $cmdln_stdin = undef;
for (my $a = 0; $a <= $#ARGV; $a++) {
    next unless $ARGV[$a] =~ /^-$/o;
    $cmdln_stdin = $a - $#ARGV;
    splice(@ARGV,$a,1);
};
```

3.1.1 Testing command-line input filenames

TO DO

- Not implemented yet an option to switch on/off reporting execution to STDERR (now by default send logs there but cannot be dissabled).

4b ⟨*Parsing command line options* 3f⟩+≡
```
sub set_input_file() {
    my $stdin_flg = $F;
    ⟨STDIN backwards compatibility 5a⟩
    &report("CHECKING_FILENAMES");
  FILECHK: foreach my $test_file (@ARGV) {
        $test_file ne ’-’ && do {
            -e $test_file || do {
                &warn(’FILE_NO_OPEN’,$T,$test_file);
                next FILECHK;
            };
            &report(’READING_FILE’,$test_file);
            push @data_files, $test_file;
            next FILECHK;
        };
        $stdin_flg = $T;
        push @data_files, ’-’;
    }; # foreach
    scalar(@data_files) == 0 && do {
        push @data_files, ’-’;
        $stdin_flg = $T;
    };
    $stdin_flg && &report(’READING_STDIN’);
} # set_input_file
```

4c ⟨*Global Vars* 2g⟩+≡
```
my @data_files = ();
```

4d ⟨*warnings - input/output* 4d⟩≡
```
FILE_NO_OPEN =>
  $spl.$Warn."Cannot Open Current file \"\%s\" . Not used !!!\n".$spl,
```

4e ⟨*messages - input/output* 4e⟩≡
```
CHECKING_FILENAMES =>
  $sp."### Validating INPUT FILENAMES\n".$sp,
READING_FILE =>
  "###--> \"\%s\" exists, including as Input File.\n",
READING_STDIN =>
  "###--> Including GFF records from standard input.\n",
```

Here is the fix for the explained in section 3.1 on page 4 (⟨*looking for STDIN* 4a⟩ code).

5a ⟨*STDIN backwards compatibility* 5a⟩≡
```
my $chk_stdin = shift @_;
my $t = scalar(@ARGV);
defined($chk_stdin) && do {
    abs($chk_stdin) > $t && ($chk_stdin = -$t);
    $chk_stdin > 0  && ($chk_stdin = 0 );
    $t += $chk_stdin;
    splice(@ARGV,$t,0,'-');
};
```

3.1.2 Defining help and auxiliary code chunks

The following command-line checkings look for those options exiting the program: '`help`' and '`version`'. Both need to output to screen without any other message/warning being displayed at the same time.

5b ⟨*command-line options with exit* 5b⟩≡
```
"version"   => \&prt_version,
"h|help|?"  => \&prt_help,
```

5c ⟨*command-line help - help* 5c⟩≡
```
-h, -help            Shows this help.
-version             Shows current version and exits.
```

5d ⟨*Parsing command line options* 3f⟩+≡
```
sub prt_version() {
    &report('SHOW_VERSION',$PROGRAM,$VERSION);
    exit(1);
} # prt_version
```

5e ⟨*messages - parsing command-line* 5e⟩≡
```
'SHOW_VERSION' => $sp."### \%s - Version: \%s\n".$sp,
```

Printing command-line help to `STDERR`:

5f ⟨*Parsing command line options* 3f⟩+≡
```
sub prt_help() {
    print STDERR «"+++EndOfHelp+++";
PROGRAM:
                        $PROGRAM - $VERSION

    ⟨Prog DESC 2c⟩

USAGE:      ⟨Prog USAGE 2b⟩


DESCRIPTION:

    ⟨Prog DESC 2c⟩


REQUIRES:

    ⟨perl requires help 6a⟩


COMMAND-LINE OPTIONS:

    ⟨command-line help 6b⟩
BUGS:    Report any problem to 'jabril\@imim.es'.

AUTHOR:  $PROGRAM is under GNU-GPL (C) 2000 - Josep F. Abril

+++EndOfHelp+++
    exit(1);
} # prt_help
```

6a ⟨*perl requires help* 6a⟩≡

```
$PROGRAM needs the following Perl modules
installed in your system, we used those available
from the standard Perl distribution. Those that
are not in the standard distribution are marked
with an '(*)', in such cases make sure that you
already have downloaded them from CPAN
(http://www.perl.com/CPAN) and installed.
```

⟨*perl requires - Getopt* 3e⟩
⟨*perl requires - Benchmark* 12f⟩

6b ⟨*command-line help* 6b⟩≡

```
A double dash on itself "-" signals end of the options
and start of file names (if present). After double dash,
you can use a single dash "-" as STDIN placeholder.
Available options and a short description are listed here:

+ General options:
```

⟨*command-line help - help* 5c⟩

⟨*command-line help - format* 6e⟩

```
+ GFF output:
```

⟨*command-line help - GFF* 7c⟩

3.1.3 Defining command-line options

6c ⟨*command-line options - format* 6c⟩≡

```
"g|genomic-coords" => sub { $CmdLineVar{GFFCOORDS} = 1 },
"c|cds-coords"     => sub { $CmdLineVar{GFFCOORDS} = 2 },
"p|protein-coords" => sub { $CmdLineVar{GFFCOORDS} = 3 },
"i|include-groups" => \$CmdLineVar{SHOWGROUPS},
```

6d ⟨*cmdline defaults - format* 6d⟩≡

```
GFFCOORDS  => 1,
SHOWGROUPS => 0,
```

6e ⟨*command-line help - format* 6e⟩≡

```
-g, -genomic-coords   Output GFF coords fields (columns 4th and 5th)
                       are set as genomic coords (by default).
-c, -cds-coords       Output GFF coords fields (columns 4th and 5th)
                       are set to CDS coords.
-p, -protein-coords   Output GFF coords fields (columns 4th and 5th)
                       are set to protein coords (using partial codon
                       notation as explained in ... ).
-i, -include-groups   Output a GFF record with the start/end coords
                       for each group in the input GFF file.
```

7a ⟨*command-line options - GFF* 7a⟩≡
```
"1|gff-version1"    => sub { $CmdLineVar{GFFVERSION} = 1 },
"2|gff-version2"    => sub { $CmdLineVar{GFFVERSION} = 2 },
```

7b ⟨*command-line defaults - GFF* 7b⟩≡
```
GFFVERSION => 1,
```

7c ⟨*command-line help - GFF* 7c⟩≡
```
-2, -gff-version2
-1, -gff-version1     Output GFF version, default is 1, where version 1
                       means simple grouping field and version 2 forces
                       grouping to be tag-value pair grouping field,
                       putting value between double quotes.
```

3.2 Main program functions

7d ⟨*Main program functions* 7d⟩≡
```
sub main() {
    &parse_cmdline(); # PROG-START
    &set_output_subs();
    foreach my $lfile (@data_files) {
        &parse_input_files($lfile);
    }; # foreach $lfile
    &sort_by_acceptor();
    &output_extended_GFF();
    &report('PROG-FINISH',&timing($T));
} # main
```

3.2.1 Defining referenced functions

Here we describe how we set the variables that call by reference a funcion. They point to different subroutines depending on a given command-line options set.

Groups, when shown in the output, share the same format as the other GFF fields, so that, we do not have to define a new variable containing the output format for both data types.

7e ⟨*Main program functions* 7d⟩+≡
```
sub set_output_subs() {
    &report("SETTING_VARS");
    my $base = "\%s\t" x 8 ;
    if ($CmdLineVar{GFFVERSION} == 1) {
        $GFF = $base."\%s  \%s  \%s\n" ; # base + group coordsA coordsB
    } else {                       # base + tag "group"; coordsA; coordsB
        $GFF = $base.'gene_id "%s"; %s; %s'."\n" ;
    };
    CHECKCOORDS: {
        $CmdLineVar{GFFCOORDS} == 3 && do {
            $printGFF = \&gff_protein;
            last CHECKCOORDS;
        };
        $CmdLineVar{GFFCOORDS} == 2 && do {
            $printGFF = \&gff_cds;
            last CHECKCOORDS;
        };
        $printGFF = \&gff_genomic;
    };
} # set_output_subs
```

7f ⟨*Global Vars* 2g⟩+≡
```
my ($GFF,$printGFF);
```

8a ⟨*messages - program running* 3b⟩+≡
```
SETTING_VARS => $sp."### Variable Definition Finished...\n".$sp,
```

The array passed to the `&$printGFF` function has the same fields structure as it has the `&gff_genomic` function. The other two functions just take a different ordering of such array.

8b ⟨*Main program functions* 7d⟩+≡
```
sub gff_genomic() {
    my @data = @_;
    printf STDOUT $GFF, @data[0..8],
                        "CDS @data[9..10]", "AA @data[11..12]";
} # gff_genomic
sub gff_cds() {
    my @data = @_;
    printf STDOUT $GFF, @data[0..2,9..10,5..8],
                        "BASE @data[3..4]", "AA @data[11..12]";
} # gff_cds
sub gff_protein() {
    my @data = @_;
    printf STDOUT $GFF, @data[0..2,11..12,5..8],
                        "BASE @data[3..4]", "CDS @data[9..10]";
} # gff_protein
```

3.2.2 Parsing GFF records

8c ⟨*Main program functions* 7d⟩+≡
```
sub parse_input_files() {
    my $ifile = $_[0];
    &report("PARSING_GFF",$ifile);
    open(IFILE,"< $ifile") || do {
        &warn('FILE_NO_OPEN',$T,$ifile);
        return;
    };
    ($n,$c) = (0,undef);
    while (<IFILE>) {
        my (@f,$start,$end,$group,$strand);
        $c = '*';
        ⟨Skip comments and empty records 13a⟩
        @f = split /\s+/o, $_, 9;
        ($group,$c) = &checkgroup($f[8]);
        ($start,$end,$strand) = (@f[3,4],"$f[6]");
        ($start > $end) && do {
            &warn('SWAPPINGCOORDS',$T,$start,$end);
            ($start,$end) = ($end,$start);
        };
        defined($GeneList{$group}{STRAND}) ||
            ($GeneList{$group}{STRAND} = $strand);
        ($GeneList{$group}{STRAND} ne $strand) && do {
            &warn('NOT_EQ_STRAND',$T,$GeneList{$group}{STRAND},$strand);
            # does nothing, just warns.
        };
        push @{ $GeneList{$group}{RECORDS} }, [ @f[0..7] ];
        push @{ $GeneList{$group}{MIN} }, $start;
        push @{ $GeneList{$group}{MAX} }, $end;
        defined($GeneList{$group}{LENGTH}) || ($GeneList{$group}{LENGTH} = 0);
        $GeneList{$group}{LENGTH} += ($end - $start + 1);
    } continue {
        &counter(++$n,$c);
    }; # while
    &counter_end($n,$c);
```

```
        close(IFILE);
    } # parse_input_files
```

9a ⟨*Global Vars* 2g⟩+≡
```
    my %GeneList = ();
```

9b ⟨*messages - program running* 3b⟩+≡
```
    PARSING_GFF => $sp."### PARSING GFF FILE: \%s\n".$sp,
```

9c ⟨*warnings - input/output* 4d⟩+≡
```
    SWAPPINGCOORDS =>
      $Warn."START greater than END (\%s > \%s). SWAPPING COORDS !!!\n",
    NOT_EQ_STRAND =>
      $Warn."GROUP STRAND (\%s) DOES NOT MATCH GFF-RECORD (\%s). GROUP RULES !!!\n",
```

9d ⟨*Main program functions* 7d⟩+≡
```
    sub checkgroup() {
        my ($gpA,$gpB);
        my ($gpstr) = @_;
        $gpstr =~ /^([^\s]+)(?:\s+"(.+?)")?(?:.*)?/o && do {
            $gpB = defined($2) ? $2 : ”; # for GFF version 2
            $gpA = defined($1) ? $1 : $gpB;  # for GFF version 1
        };
        ($gpB ne ”) && ( return $gpB, ’:’ );
        return $gpA, ’.’;
    } # checkgroups
```

3.2.3 Sorting GFF records

9e ⟨*Main program functions* 7d⟩+≡
```
    sub sort_by_acceptor() {
        &report("SORTING_FEATURES");
        $c = 0;
        foreach my $gname (keys %GeneList) {
            my $ref = \@{ $GeneList{$gname}{RECORDS} };
            print STDERR "$gname...(".($#{$ref} + 1)."ft)".
                          (((++$c % 4) == 0) ? "\n" : "\t");
            # sorting all features by acceptor
            @{ $ref } = map  { $_->[2] }
                        sort { &sort_forward }
                        map  { [ $_->[3], $_->[4], $_ ] } @{ $ref };
            # getting group boundaries
            $GeneList{$gname}{MIN} = min(@{ $GeneList{$gname}{MIN} });
            $GeneList{$gname}{MAX} = max(@{ $GeneList{$gname}{MAX} });
            push @SortedGenes,
                 [ $gname, $GeneList{$gname}{MIN}, $GeneList{$gname}{MAX},
                   $GeneList{$gname}{STRAND}, $GeneList{$gname}{LENGTH} ];
        }; # foreach $gname
        ((++$c % 4) != 0) && print STDERR "\n";
        &report("SORTING_GROUPS");
        @SortedGenes = map  { $_->[2] }
                       sort { &sort_forward }
                       map  { [ $_->[1], $_->[2], $_ ] } @SortedGenes;
    } # sort_by_acceptor
```

9f ⟨*Global Vars* 2g⟩+≡
```
    my @SortedGenes = ();
```

9g ⟨*messages - program running* 3b⟩+≡
```
    SORTING_FEATURES => $sp."### Sorting GFF Group Features\n".$sp,
    SORTING_GROUPS   => $sp."### Sorting GFF Groups\n".$sp,
```

10a ⟨*Main program functions* 7d⟩+≡
```
  sub sort_forward {
      $a->[0] <=> $b->[0]  # sorting by start
               or
      $b->[1] <=> $a->[1]; # reverse sorting by end if same start
  } # sort_forward
```

3.2.4 Thowing "expanded output"

10b ⟨*Main program functions* 7d⟩+≡
```
  sub output_extended_GFF() {
      my $rpt = '%-15s%12s%12s%7s%10s'."\n";
      &report("WRITING_FEATURES");
      printf STDERR $rpt,"# GeneName","MinCoord","MaxCoord","Strand","CDSlength";
      for (my $g = 0; $g <= $#SortedGenes; $g++) {
          my ($ref,$gene,$ori,$end,$str,$len,$f,$do_it,$base);
          ($gene,$ori,$end,$str,$len) = @{ $SortedGenes[$g] };
          printf STDERR $rpt,"  $gene",$ori,$end,$str,$len;
          $ref = \@{ $GeneList{$gene}{RECORDS} };
          if ($str ne '-') {
              $do_it = \&do_it_forward;
              $base  = 0;
          } else {
              $do_it = \&do_it_reverse;
              $base  = $len + 1;
          };
          for ($f = 0; $f <= $#{$ref}; $f++) {
              my ($o_cds,$e_cds,$o_aa,$e_aa);
              # ($o_gn,$e_gn) = ($ref->[],$ref->[]);
              ($o_cds,$e_cds) = &$do_it($base,$ref->[$f][3],$ref->[$f][4]);
              $base = $e_cds;
              ($o_aa,$e_aa) = &to_protein($o_cds,$e_cds);
              ($str eq '-') && do { # just to display GFF-like (start<end)
                  ($o_cds,$e_cds,$o_aa,$e_aa) = ($e_cds,$o_cds,$e_aa,$o_aa);
              };
              &$printGFF(@{ $ref->[$f] },$gene,$o_cds,$e_cds,$o_aa,$e_aa);
          }; # for $f
      }; # for $g
  } # output_extended_GFF
```

10c ⟨*messages - program running* 3b⟩+≡
```
  WRITING_FEATURES => $sp."### Writing NEW GFF Records to STDOUT...\n".$sp,
```

10d ⟨*Main program functions* 7d⟩+≡
```
  sub do_it_forward() {
      my ($bori,$gori,$gend) = @_;
      my ($cori,$cend);
      $cori = $bori + 1;
      $cend = $bori + ($gend - $gori + 1);
      return ($cori,$cend);
  } # do_it_forward
```

The following function, together with `$GeneList{$gene_name}{LENGTH}`, is only needed if we are working with all the groups sorted by acceptor, an easier solution would be sorting reverse-strand groups by donor instead by acceptor (but then we produce GFF-records that are not sorted by acceptor in genomic coords, which is the main coords system by default —just take into account Enrique's programs. . .).

10e ⟨*Main program functions* 7d⟩+≡

```
sub do_it_reverse() {
    my ($bori,$gori,$gend) = @_;
    my ($cori,$cend);
    $cori = $bori - 1;
    $cend = $cori - ($gend - $gori);
    return ($cori,$cend);
} # do_it_reverse
```

Here we need to convert to protein coords.

11 ⟨*Main program functions* 7d⟩+≡

```
sub to_protein() {
    my ($cori,$cend) = @_;
    my ($pori,$pend);
    $pori = &toprot($cori);
    $pend = &toprot($cend);
    return ($pori,$pend);
} # to_protein
sub toprot() {
    my ($val) = @_;
    my ($a,$b);
    $a = $val % 3;                # nucleotide order within codon
    ($a == 0) && ($a = 3);
    $b = int(($val - 1) / 3) + 1; # codon number
    return "$b.$a";
} # toprot
```

A Common code blocks

A.1 PERL scripts

12a ⟨*PERL shebang* 12a⟩≡
```
#!/usr/bin/perl -w
# This is perl, version 5.005_03 built for i386-linux
#
⟨Program Description 2d⟩
#
⟨GNU License 14f⟩
#
⟨Version Control Id Tag 14e⟩
#
use strict;
#
⟨Program Info 2a⟩
my $DATE = localtime;
my $USER = defined($ENV{USER}) ? $ENV{USER} : 'Child Process';
my $host = `hostname`;
chomp($host);
#
```

12b ⟨*Boolean* 12b⟩≡
```
my ($T,$F) = (1,0); # for 'T'rue and 'F'alse
```

A.1.1 Timing our scripts

The '`Benchmark`' module encapsulates a number of routines to help to figure out how long it takes to execute a piece of code and the whole script.

12c ⟨*Use Modules - Benchmark* 12c⟩≡
```
use Benchmark;
   ⟨Timer ON 12d⟩
```

See '`man Benchmark`' for further info about this package. We set an array to keep record of timing for each section.

12d ⟨*Timer ON* 12d⟩≡
```
my @Timer = (new Benchmark);
```

12e ⟨*Common PERL subs - Benchmark* 12e⟩≡
```
sub timing() {
    push @Timer, (new Benchmark);
    # partial time
    $_[0] ||
        (return timestr(timediff($Timer[$#Timer],$Timer[($#Timer - 1)])));
    # total time
    return timestr(timediff($Timer[$#Timer],$Timer[0]));
} # timing
```

12f ⟨*perl requires - Benchmark* 12f⟩≡
```
"Benchmark" - checking and comparing running times of code.
```

A.1.2 Printing complex Data Structures

With '`Data::Dumper`' we are able to pretty print complex data structures for debugging them.

12g ⟨*Use Modules - Dumper* 12g⟩≡
```
use Data::Dumper;
local $Data::Dumper::Purity = 0;
local $Data::Dumper::Deepcopy = 1;
```

A.1.3 Common functions

13a ⟨*Skip comments and empty records* 13a⟩≡
```
next if /^\#/o;
next if /^\s*$/o;
chomp;
```

13b ⟨*Common PERL subs - Min Max* 13b⟩≡
```
#
sub max() {
    my $z = shift @_;
    foreach my $l (@_) { $z = $l if $l > $z };
    return $z;
} # max
sub min() {
    my $z = shift @_;
    foreach my $l (@_) { $z = $l if $l < $z };
    return $z;
} # min
```

13c ⟨*Common PERL subs - Text fill* 13c⟩≡
```
#
sub fill_right() { $_[0].($_[2] x ($_[1] - length($_[0]))) }
sub fill_left()  { ($_[2] x ($_[1] - length($_[0]))).$_[0] }
sub fill_mid()   {
    my $l = length($_[0]);
    my $k = int(($_[1] - $l)/2);
    ($_[2] x $k).$_[0].($_[2] x ($_[1] - ($l+$k)));
} # fill_mid
```

These functions are used to report to STDERR a single char for each record processed (useful for reporting parsed records).

13d ⟨*Common PERL subs - Counter* 13d⟩≡
```
#
sub counter { # $_[0]~current_pos++ $_[1]~char
    print STDERR "$_[1]";
    (($_[0] % 50) == 0) && (print STDERR "[".&fill_left($_[0],6,"0")."]\n");
} # counter
#
sub counter_end { # $_[0]~current_pos   $_[1]~char
    (($_[0] % 50) != 0) && (print STDERR "[".&fill_left($_[0],6,"0")."]\n");
} # counter_end
```

13e ⟨*Counter vars* 13e⟩≡
```
my ($n,$c); # counter and char (for &counter function)
```

A.1.4 Common functions for reporting program processes

Function ’`report`’ requires that a hash variable ’`%Messages`’ has been set, such hash contains the strings for each report message we will need. The first parameter for ’`report`’ is a key for that hash, in order to retrieve the message string, the other parameters passed are processed by the `sprintf` function on that string.

13f ⟨*Common PERL subs - STDERR* 13f⟩≡
```
sub report() { print STDERR sprintf($Messages{ shift @_ },@_) }
```

The same happens to ’`warn`’ function which also uses the hash variable ’`%Messages`’ containing the error messages.

13g ⟨*Common PERL subs - STDERR* 13f⟩+≡
```
sub warn() { print STDERR sprintf($Messages{ shift @_ }, @_) }
```

Those are accessory variables for the messages strings:

14a ⟨*Stderr subs vars* 14a⟩≡
```
my $line = ('#' x 80)."\n";
my $s = '### ';
my $sp = "###\n";
my $Error = "\<\<\<  ERROR  \>\>\> ";
my $Warn  = "\<\<\< WARNING \>\>\> ";
my $spl   = "\<\<\<\-\-\-\-\-\-\-\-\-\>\>\>\n";
my $spw   = "\<\<\<         \>\>\> ";
```

And here the main messages hash:

14b ⟨*Stderr subs vars* 14a⟩+≡
```
my %Messages = (
    # ERROR MESSAGES
    ⟨warnings - command-line 3g⟩
    ⟨warnings - input/output 4d⟩
    # WORKING MESSAGES
    ⟨messages - program running 3b⟩
    ⟨messages - parsing command-line 5e⟩
    ⟨messages - input/output 4e⟩
   ); # %Messages
```

A.2 BASH scripts

14c ⟨*BASH shebang* 14c⟩≡
```
#!/usr/bin/bash
# GNU bash, version 2.03.6(1)-release (i386-redhat-linux-gnu)
⟨Version Control Id Tag 14e⟩
#
SECONDS=0 # Reset Timing
# Which script are we running...
L="####################"
{ echo "$L$L$L$L";
  echo "### RUNNING [$0]";
  echo "### Current date:`date`";
  echo "###"; } 1>&2;
```

14d ⟨*BASH script end* 14d⟩≡
```
{ echo "###"; echo "### Execution time for [$0] : $SECONDS secs";
  echo "$L$L$L$L";
  echo ""; } 1>&2;
#
exit 0
```

A.3 Version control tags

This document is under Revision Control System (RCS). The version you are currently reading is the following:

14e ⟨*Version Control Id Tag* 14e⟩≡
```
# $Id: deploy.nw,v 1.12 2001/10/05 18:30:33 jabril Exp $
```

A.4 GNU General Public License

14f ⟨*GNU License* 14f⟩≡
```
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
```

```
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#
# #-------------------------------------------#
```

B Extracting code blocks from this document

From this file we can obtain both the code and the documentation. The following instructions are needed:

B.1 Extracts Script code chunks from the NOWEB file

Remember when tangling that '-L' option allows you to include program line-numbering relative to original NOWEB file. Then the first line of the executable files is a comment, not a shebang, and must be removed to make scripts runnable.

16a ⟨*tangling* 16a⟩≡
```
# showing line numbering comments in program
notangle -L -R"fullgffcoords" $WORK/$nwfile.nw | \
    perl -ne '$.>1 && print' | cpif $BIN/fullgffcoords.pl ;
chmod a+x $BIN/fullgffcoords.pl ;
```

16b ⟨*tangling* 16a⟩+≡
```
# reformating program with perltidy
notangle -R"fullgffcoords" $WORK/$nwfile.nw | \
    perltidy - | cpif $SRC/fullgffcoords ;
# html pretty-printing program with perltidy
notangle -R"fullgffcoords" $WORK/$nwfile.nw | \
    perltidy -html - | cpif $DOCS/html/fullgffcoords.html ;
#
```

B.2 Extracting different Config Files

16c ⟨*tangling* 16a⟩+≡
```
notangle -R"root" $WORK/$nwfile.nw | \
         cpif $DATA/root_config ;
```

B.3 Extracting documentation and LaTeX'ing it

16d ⟨*tangling* 16a⟩+≡
```
notangle -Rweaving  $WORK/$nwfile.nw | cpif $WORK/nw2tex ;
notangle -RLaTeXing $WORK/$nwfile.nw | cpif $WORK/ltx ;
chmod a+x $WORK/nw2tex $WORK/ltx;
```

16e ⟨*tangling complementary LaTeX files* 16e⟩≡
```
notangle -R"HIDE: LaTeX new definitions" $WORK/$nwfile.nw | cpif $DOCS/defs.tex ;
notangle -R"HIDE: TODO" $WORK/$nwfile.nw | cpif $DOCS/todo.tex ;
```

16f ⟨*weaving* 16f⟩≡
⟨*BASH shebang* 14c⟩
```
# weaving and LaTeXing
```
⟨*BASH Environment Variables* 17b⟩
⟨*tangling complementary LaTeX files* 16e⟩
```
noweave -v -t4 -delay -x -filter 'elide "HIDE: *"' \
        $WORK/$nwfile.nw | cpif $DOCS/$nwfile.tex ;
# noweave -t4 -delay -index $WORK/$nwfile.nw > $DOCS/$nwfile.tex
pushd $DOCS/ ;
#
latex $nwfile.tex ;
dvips $nwfile.dvi -o $nwfile.ps -t a4 ;
#
popd;
```
⟨*BASH script end* 14d⟩

17a ⟨*LaTeXing* 17a⟩≡

```
⟨BASH shebang 14c⟩
# only LaTeXing
⟨BASH Environment Variables 17b⟩
pushd $DOCS/ ;
#
echo "### RUNNING LaTeX on $nwfile.tex" 1>&2 ;
latex $nwfile.tex ;
latex $nwfile.tex ;
latex $nwfile.tex ;
dvips $nwfile.dvi -o $nwfile.ps -t a4 ;
#
# pdflatex $nwfile.tex ;
echo "### CONVERTING PS to PDF: $nwfile" 1>&2 ;
ps2pdf $nwfile.ps $nwfile.pdf ;
#
popd ;
⟨BASH script end 14d⟩
```

B.4 Defining working shell variables for the current project

17b ⟨*BASH Environment Variables* 17b⟩≡

```
#
# Setting Global Variables
WORK="/home/ug/jabril/development/softjabril/fullgffcoords" ;
BIN="$WORK/bin" ;
PARAM="$BIN/param" ;
SRC="$WORK/src" ; # where to put the distributable files
DOCS="$WORK/docs" ;
DATA="$WORK/data" ;
TEST="$WORK/tests" ;
nwfile="fullgffcoords" ;
export WORK BIN PARAM DOCS DATA nwfile ;
#
```

17c ⟨*tangling* 16a⟩+≡

```
#
# BASH Environment Variables
notangle -R'BASH Environment Variables' $WORK/$nwfile.nw | \
         cpif $WORK/.bash_VARS ;
source $WORK/.bash_VARS ;
#
```