PROGRAM NAME: # splitfasta

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

LAST UPDATE: **September 4, 2001**

DESCRIPTION: This program breaks large fasta files into smaller fasta sequences, given a certain subsequence length and a fixed overlap.

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 1.7 2001/09/03 18:23:46 jabril Exp` >

1 Introduction

1.1 Program description

1.2 Input

1.3 Output

1.4 To Do

- This is a first draft of the splitfasta. . . . . [*Section* 2.1, *page* 2]

2 Implementation

2.1 Program outline

2a ⟨*splitfasta* 2a⟩≡
```
⟨PERL shebang 6a⟩
#
# MODULES
#
⟨Use Modules 2b⟩
#
# VARIABLES
#
⟨Global Vars 2c⟩
#
# MAIN LOOP
#
⟨Main Loop 2d⟩
#
# FUNCTIONS
#
⟨Functions 2e⟩
#
#
⟨OLD IMPLEMENTATION 2f⟩
```

2b ⟨*Use Modules* 2b⟩≡

2c ⟨*Global Vars* 2c⟩≡

2d ⟨*Main Loop* 2d⟩≡
```

exit(0);
```

2e ⟨*Functions* 2e⟩≡
```
sub {
} #
```

TO DO

- This is a first draft of the splitfasta.

2f ⟨*OLD IMPLEMENTATION* 2f⟩≡
```
#
# splitfastaseq.pl \
#     seqlength overlap < fastafile > output
#
#     Breaking large fasta sequences to build
#     databases for running tblastx faster
#
use lib qw( /usr/lib/perl5/site_perl/5.005/ );
use Bio::Seq;
use Bio::SeqIO;
use Benchmark;
my ($T,$F) = (1,0);
my @Timer = (new Benchmark);
my $PROGRAM = 'splitfastaseq.pl';
my $DATE = localtime;
my $USER = defined($ENV{USER}) ? $ENV{USER} : '??????';
my $host = `hostname`;
```

```
chomp($host);
my $line = ('#' x 80)."\n";
my $s = '### ';
#
my ($id,$ln,$sq) = ('',0,'');
my ($total_time,$seq);
my ($maxlen,$overlap) = @ARGV;

print STDERR « "+++EOR+++";
$line$s\n$s Running $PROGRAM\n$s
$s HOST: $host
$s USER: $USER
$s DATE: $DATE\n$s\n$line$s
+++EOR+++

&getseq();
&splitseq();

$total_time = &timing($T);
print STDERR « "+++EOR+++";
$s\n$line$s\n$s $PROGRAM FINISHED\n$s
$s TOTAL TIME: $total_time\n$line
+++EOR+++

exit(0);

sub getseq() { # assuming here single sequence input fasta files
    print STDERR "$s Processing fasta file.\n";
    my $seqin = Bio::SeqIO->new(-format => 'FASTA', -fh => \*STDIN);
    while (my $iseq = $seqin->next_seq()) {
        $id = $iseq->display_id();
        $ln = $iseq->length();
        $sq = $iseq->seq();
        last; # to make sure that we only catch a single fasta sequence
    }; # while next_seq
    $seq = Bio::Seq->new( -seq => $sq , -id => $id );
    print STDERR "$s Processing DONE: ".(&timing($F))."\n$s\n";
} # getseq
#
sub splitseq() {
    my ($e,$sid,$ssq,$nseq,$wseq);
    my ($t,$sqlen) = (1,($maxlen + $overlap - 1));
    print STDERR "$s Creating splitted-sequence fasta file ($ln nt).\n";
    my $seqout = Bio::SeqIO->new(-format => 'FASTA', -fh => \*STDOUT);
    while ($t < $ln) {
         $e = $t + $sqlen;
         ($e > $ln) && ($e = $ln);
         $sid = "$id\_$t\_$e";
         print STDERR "$s -> $id : from $t to $e (".($e - $t + 1)." nt)\n";
         $ssq = $seq->subseq($t,$e);
         $t += $maxlen;
         #
         $wseq = Bio::Seq->new( -seq => $ssq , -id => $sid );
         $seqout->write_seq($wseq);
    }; # while
    print STDERR "$s Splitting DONE: ".(&timing($F))."\n$s\n";
} # splitseq
#
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
```

A empty appendix section

A.1 empty appendix subsection

B Common code blocks

B.1 PERL scripts

6a ⟨*PERL shebang* 6a⟩≡
```
#!/usr/bin/perl -w
# This is perl, version 5.005_03 built for i386-linux
⟨GNU License 8d⟩
⟨Version Control Id Tag 8c⟩
#
use strict;
```

6b ⟨*Global Constants - Boolean* 6b⟩≡
```
my ($T,$F) = (1,0); # for 'T'rue and 'F'alse
```

We also set here the date when the script is running and who is the user running it.

6c ⟨*Global Vars - User and Date* 6c⟩≡
```
my $DATE = localtime;
my $USER = $ENV{USER};
```

B.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.

6d ⟨*Use Modules - Benchmark* 6d⟩≡
```
use Benchmark;
    ⟨Timer ON 6e⟩
```

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

6e ⟨*Timer ON* 6e⟩≡
```
my @Timer = (new Benchmark);
```

6f ⟨*Common PERL subs - Benchmark* 6f⟩≡
```
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
```

B.1.2 Printing complex Data Structures

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

6g ⟨*Use Modules - Dumper* 6g⟩≡
```
use Data::Dumper;
local $Data::Dumper::Purity = 0;
local $Data::Dumper::Deepcopy = 1;
```

B.1.3 Common functions

6h ⟨*Skip comments and empty records* 6h⟩≡
```
next if /^\#/o;
next if /^\s*$/o;
chomp;
```

7a ⟨*Common PERL subs - Min Max* 7a⟩≡
```
#
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
```

7b ⟨*Common PERL subs - Text fill* 7b⟩≡
```
#
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).

7c ⟨*Common PERL subs - Counter* 7c⟩≡
```
#
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
```

7d ⟨*Global Vars - Counter* 7d⟩≡
```
my ($n,$c); # counter and char (for &counter function)
```

B.1.4 Common functions for reporting program processes

Function ’`report`’ requires that a hash variable ’`%MessageList`’ 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.

7e ⟨*Common PERL subs - STDERR* 7e⟩≡
```
sub report() { print STDERR sprintf($MessageList{ shift @_ },@_) }
```

The same happens to ’`warn`’ function which also requires a hash variable ’`%ErrorList`’ containing the error messages.

7f ⟨*Common PERL subs - STDERR* 7e⟩+≡
```
sub warn() { print STDERR sprintf($ErrorList{ shift @_ }, @_) }
```

B.2 BASH scripts

8a ⟨*BASH shebang* 8a⟩≡

```
#!/usr/bin/bash
# GNU bash, version 2.03.6(1)-release (i386-redhat-linux-gnu)
⟨Version Control Id Tag 8c⟩
#
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;
```

8b ⟨*BASH script end* 8b⟩≡

```
{ echo "###"; echo "### Execution time for [$0] : $SECONDS secs";
  echo "$L$L$L$L";
  echo ""; } 1>&2;
#
exit 0
```

B.3 Version control tags

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

8c ⟨*Version Control Id Tag* 8c⟩≡

```
# $Id: deploy.nw,v 1.7 2001/09/03 18:23:46 jabril Exp $
```

B.4 GNU General Public License

8d ⟨*GNU License* 8d⟩≡

```
# #------------------------------------------#
# #                         splitfasta                        #
# #------------------------------------------#
#
#    Remember to put a short description of your script here...
#
#     Copyright (C) 2001 - Josep Francesc ABRIL FERRANDO
#
# 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.
#
# #------------------------------------------#
```

C Extracting code blocks from this document

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

C.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.

9a ⟨*tangling* 9a⟩≡
```
# showing line numbering comments in program
notangle -L -R"splitfasta" $WORK/$nwfile.nw | \
    perl -ne '$.>1 && print' | cpif $BIN/splitfasta.pl ;
chmod a+x $BIN/splitfasta.pl ;
```

9b ⟨*tangling* 9a⟩+≡
```
# reformating program with perltidy
notangle -R"splitfasta" $WORK/$nwfile.nw | \
    perltidy - | cpif $BIN/splitfasta.pl ;
# html pretty-printing program with perltidy
notangle -R"splitfasta" $WORK/$nwfile.nw | \
    perltidy -html - | cpif $DOCS/html/splitfasta.html ;
#
```

C.2 Extracting different Config Files

9c ⟨*tangling* 9a⟩+≡
```
notangle -R"root" $WORK/$nwfile.nw | \
         cpif $DATA/root_config ;
```

C.3 Extracting documentation and LaTeX'ing it

9d ⟨*tangling* 9a⟩+≡
```
notangle -Rweaving  $WORK/$nwfile.nw | cpif $WORK/nw2tex ;
notangle -RLaTeXing $WORK/$nwfile.nw | cpif $WORK/ltx ;
chmod a+x $WORK/nw2tex $WORK/ltx;
```

9e ⟨*tangling complementary LaTeX files* 9e⟩≡
```
notangle -R"HIDE: LaTeX new definitions" $WORK/$nwfile.nw | cpif $DOCS/defs.tex ;
notangle -R"HIDE: TODO" $WORK/$nwfile.nw | cpif $DOCS/todo.tex ;
```

9f ⟨*weaving* 9f⟩≡
```
⟨BASH shebang 8a⟩
# weaving and LaTeXing
⟨BASH Environment Variables 10b⟩
⟨tangling complementary LaTeX files 9e⟩
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 8b⟩
```

10a ⟨*LaTeXing* 10a⟩≡
```
⟨BASH shebang 8a⟩
# only LaTeXing
⟨BASH Environment Variables 10b⟩
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 8b⟩
```

C.4 Defining working shell variables for the current project

10b ⟨*BASH Environment Variables* 10b⟩≡
```
#
# Setting Global Variables
WORK="/home/ug/jabril/development/softjabril/splitfasta" ;
BIN="$WORK/bin" ;
PARAM="$BIN/param" ;
DOCS="$WORK/docs" ;
DATA="$WORK/data" ;
nwfile="splitfasta" ;
export WORK BIN PARAM DOCS DATA nwfile ;
#
```

10c ⟨*tangling* 9a⟩+≡
```
#
# BASH Environment Variables
notangle -R'BASH Environment Variables' $WORK/$nwfile.nw | \
         cpif $WORK/.bash_VARS ;
source $WORK/.bash_VARS ;
#
```