PROGRAM NAME: # blat2gff.pl

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

LAST UPDATE: **October 5, 2001**

DESCRIPTION: Converting BLAT output to GFF

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: perlscript.nw,v 1.5 2001/10/03 11:21:02 jabril Exp >

1 Introduction

1.1 Program description

1.2 Input

1 ⟨*BLAT output example* 1⟩≡

```
39  9   0   0   0   0   0   0   ++  Hsap_BTK    78931   61871   61919   ~/Research/Homolo
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   64087   64135   1   48, 61871,  64
36  6   0   0   0   0   0   0   ++  Hsap_BTK    78931   61867   61909   ~/Research/Homolo
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   64087   64129   1   42, 61867,  64
1940    181 0   0   4   87  5   96  -+  Hsap_BTK    78931   66352   68560   ~/Research/Ho
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   16106   18323   7   185,103,50,41,
212 26  0   0   0   0   0   0   -+  Hsap_BTK    78931   11174   11412   ~/Research/Homolo
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   72680   72918   1   238,    67519,
406 68  0   0   1   207 1   202 +-  Hsap_BTK    78931   62134   62815   ~/Research/Homolo
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   21775   22451   2   165,309,    62
238 28  0   0   0   0   0   0   +-  Hsap_BTK    78931   10446   10712   ~/Research/Homolo
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   73544   73810   1   266,    10446,
33  0   0   0   0   0   0   0   -  Hsap_BTK    78931   22065   22098   ~/Research/Homolog
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   75119   75152   1   33, 56833,  13
33  0   0   0   0   0   0   0   -  Hsap_BTK    78931   22081   22114   ~/Research/Homolog
blat/tests/homologydb/blat/raw/Mmus_BTK.nib    88871   75111   75144   1   33, 56817,  13
```

1.3 Output

1.4 To Do

- ▷ [*Section* 2.1, *page* 4]
  This is a first draft of the blat2gff.pl.

- ▷ [*Section* 2.1, *page* 4]
  Include GetOPTs to check comand-line options.

2 Implementation

2a ⟨*Program Info* 2a⟩≡
```
my $PROGRAM = 'blat2gff.pl';
my $VERSION = '0.1_alpha';
```

2b ⟨*Program Description* 2b⟩≡
```
# #-----------------------------------------#
# #                          blat2gff.pl                          #
# #-----------------------------------------#
#
#    blat2gff.pl [options] <files>
#
#    Remember to put a short description of your script here...
#
#     Copyright (C) 2001 - Josep Francesc ABRIL FERRANDO
```

2.1 Program outline

2c ⟨*blat2gff* 2c⟩≡
```
⟨PERL shebang 6a⟩
#
# MODULES
#
⟨Use Modules 2d⟩
#
# VARIABLES
#
⟨Global Vars 2e⟩
#
# MAIN LOOP
#
⟨Main Loop 3a⟩
#
# FUNCTIONS
#
⟨Functions 3b⟩
```

2d ⟨*Use Modules* 2d⟩≡

2e ⟨*Global Vars* 2e⟩≡
```
my @frame = (3,1,2);
my %blat = (
    STRAND =>  8,
    Q_NAME =>  9,
    Q_LEN  => 10,
    Q_ORI  => 11,
    Q_END  => 12,
    T_NAME => 13,
    T_LEN  => 14,
    T_ORI  => 15,
    T_END  => 16,
    B_NUM  => 17,
    B_LEN  => 18,
    B_QPOS => 19,
    B_TPOS => 20,
    );
my %gff = ();
my $maxfields = 21;
```

```
my %group = ();
my ($group_cnt, $rec_cnt) = (0) x 2;
```

3a ⟨*Main Loop* 3a⟩≡

```
while (<STDIN>) {
    my @f = ();
    next unless /^\d/o;
    chomp;
    @f = split /\s+/og, $_;
    do { # ensure that there are 21 fields
        print STDERR "### NOT enough fields for this record:\n##### @f \n";
        next;
    } unless scalar(@f) == $maxfields;
    %gff = ();
    ++$rec_cnt;
    $f[$blat{STRAND}] =~ /(.)(.)/og &&
        ($gff{Q_STRAND} = $1, $gff{T_STRAND} = $2);
    do { # ensure that strands are +/-
        print STDERR "### CANNOT find strand for this record:\n##### @f \n";
        next;
    } unless ($gff{Q_STRAND} =~ /[+-]/o && $gff{T_STRAND} =~ /[+-]/o);
    $f[$blat{Q_NAME}] =~ m{/?([^/]+)$}og && ($gff{Q_NAME} = $1) ;
    $f[$blat{T_NAME}] =~ m{/?([^/]+)$}og && ($gff{T_NAME} = $1) ;
    defined($group{"$gff{T_NAME}.$gff{T_STRAND}"}) ||
        ($group{"$gff{T_NAME}.$gff{T_STRAND}"} = ++$group_cnt);
    $gff{GROUP} = $group{"$gff{T_NAME}.$gff{T_STRAND}"};
    $gff{Q_LEN} = $f[$blat{Q_LEN}]; # seq lengths are OK
    $gff{T_LEN} = $f[$blat{T_LEN}];
    $gff{Q_ORI} = $f[$blat{Q_ORI}] + 1; # HSP coords start at 0, not at 1
    $gff{Q_END} = $f[$blat{Q_END}] + 1;
    $gff{T_ORI} = $f[$blat{T_ORI}] + 1;
    $gff{T_END} = $f[$blat{T_END}] + 1;
    $gff{Q_FRAME} =
        &get_frame($gff{Q_STRAND}, $gff{Q_LEN}, $gff{Q_ORI}, $gff{Q_END});
    $gff{T_FRAME} =
        &get_frame($gff{T_STRAND}, $gff{T_LEN}, $gff{T_ORI}, $gff{T_END});
    $gff{B_NUM} = $f[$blat{B_NUM}];
    @{ $gff{B_LEN} }  = split /,/og, $f[$blat{B_LEN}];
    @{ $gff{B_QPOS} } = split /,/og, $f[$blat{B_QPOS}];
    @{ $gff{B_TPOS} } = split /,/og, $f[$blat{B_TPOS}];
    printf STDOUT "# ".$GFFstring,
        $gff{Q_NAME}, "BLAT","$gff{Q_LEN}:$gff{T_LEN}",
            $gff{Q_ORI}, $gff{Q_END}, 0, $gff{Q_STRAND}, $gff{Q_FRAME},
        "$gff{T_NAME}.$gff{GROUP}.$rec_cnt",
            $gff{T_ORI}, $gff{T_END}, $gff{T_STRAND}, $gff{T_FRAME};
    &loop_HSPs($rec_cnt);
}; # while read input

exit(0);
```

3b ⟨*Functions* 3b⟩≡

```
sub get_frame() {
    my ($strand,$len,$ori,$end) = @_;
    if ($strand eq '-') {
       return $frame[(($len - $end + 1) % 3)];
    } else {
       return $frame[($ori % 3)];
    }
} # get_frame
sub get_hsp() {
```

```
    my ($s,$L,$l,$n) = @_;
    my ($o,$e);
    if ($s eq '-') {
        $e = $L - $n;
        $o = $e - $l
    } else {
        $o = $n;
        $e = $o + $l;
    };
    $o++; $e++; # HSP coords start at 0, not at 1
    return ($o, $e, &get_frame($s,$L,$o,$e));
} # get_hsp
```

4a ⟨*Global Vars* 2e⟩+≡

```
# GetSRsAln.pl like:
#$srs->{QUERY}\t$blst_prg\tsr\t$srs->{START_Q}\t$srs->{END_Q}
#\t$srs->{SCORE}\t$srs->{STRAND_Q}\t$srs->{FRAME_Q}
#\tTarget \"$srs->{SUBJECT}\"\t$srs->{START_S}\t$srs->{END_S}
#\tE_value $srs->{E_VALUE}\tStrand $srs->{STRAND_S}
#\tFrame $srs->{FRAME_S}\t\#Projection $srs->{PROJECTION}
my $GFFstring = ("\%s\t" x 8).'Target "%s"'.
       "\t\%s\t\%s\tE_Value .\tStrand \%s\tFrame \%s\n";
```

4b ⟨*Functions* 3b⟩+≡

```
sub loop_HSPs() {
    my ($num) = @_;
    my ($c,$l,$q,$t,$qo,$qe,$qf,$to,$te,$tf);
    for ($c = 0; $c < $gff{B_NUM}; $c++) {
        $l = $gff{B_LEN}[$c];
        $q = $gff{B_QPOS}[$c];
        $t = $gff{B_TPOS}[$c];
        ($qo,$qe,$qf) = &get_hsp($gff{Q_STRAND},$gff{Q_LEN},$l,$q);
        ($to,$te,$tf) = &get_hsp($gff{T_STRAND},$gff{T_LEN},$l,$t);
        printf STDOUT $GFFstring,
            $gff{Q_NAME}, "blat", "hsp", $qo, $qe, 0, $gff{Q_STRAND}, $qf,
            "$gff{T_NAME}.$gff{GROUP}.$num", $to, $te, $gff{T_STRAND}, $tf;
    }; # for
} # loop_HSPs
```

TO DO

- This is a first draft of the blat2gff.pl.
- Include GetOPTs to check comand-line options.

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
#
⟨Program Description 2b⟩
#
⟨GNU License 8d⟩
#
⟨Version Control Id Tag 8c⟩
#
use strict;
#
⟨Program Info 2a⟩
my $DATE = localtime;
my $USER = defined($ENV{USER}) ? $ENV{USER} : 'Child Process';
my $host = `hostname`;
chomp($host);
#
```

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

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.

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

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

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

6e ⟨*Common PERL subs - Benchmark* 6e⟩≡
```
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.

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

B.1.3 Common functions

7a ⟨*Skip comments and empty records* 7a⟩≡
```
next if /^\#/o;
next if /^\s*$/o;
chomp;
```

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

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

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

7e ⟨*Global Vars - Counter* 7e⟩≡
```
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.

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

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

7g ⟨*Common PERL subs - STDERR* 7f⟩+≡
```
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: perlscript.nw,v 1.5 2001/10/03 11:21:02 jabril Exp $
```

B.4 GNU General Public License

8d ⟨*GNU License* 8d⟩≡
```
# 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"blat2gff" $WORK/$nwfile.nw | \
    perl -ne '$.>1 && print' | cpif $BIN/blat2gff.pl ;
chmod a+x $BIN/blat2gff.pl ;
```

9b ⟨*tangling* 9a⟩+≡
```
# reformating program with perltidy
notangle -R"blat2gff" $WORK/$nwfile.nw | \
    perltidy - | cpif $BIN/blat2gff.pl ;
# html pretty-printing program with perltidy
notangle -R"blat2gff" $WORK/$nwfile.nw | \
    perltidy -html - | cpif $DOCS/html/blat2gff.html ;
#
```

C.2 Extracting different Config Files

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

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/blat2gff" ;
BIN="$WORK/bin" ;
PARAM="$BIN/param" ;
DOCS="$WORK/docs" ;
DATA="$WORK/data" ;
nwfile="blat2gff" ;
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 ;
#
```