humangenome

Authors List Here

— October 2, 2001—

Abstract

Short description of your program here !!!

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

† e-mail: author@imim.es

Contents

List of Tables

List of Figures

< Id: deploy.nw,v 1.9 2001/09/25 17:45:07 jabril Exp >

1 Introduction

2 Working on data

2.1 Command-line

1 ⟨*BASH commands* 1⟩≡
```
#
#
```

A empty appendix section

A.1 empty appendix subsection

B Common code blocks

B.1 PERL scripts

3a ⟨*PERL shebang* 3a⟩≡
```
#!/usr/bin/perl -w
# This is perl, version 5.005_03 built for i386-linux
#
⟨Program Description (never defined)⟩
#
⟨GNU License (never defined)⟩
#
⟨Version Control Id Tag 5d⟩
#
use strict;
#
⟨Program Info (never defined)⟩
my $DATE = localtime;
my $USER = defined($ENV{USER}) ? $ENV{USER} : 'Child Process';
my $host = `hostname`;
chomp($host);
#
```

3b ⟨*Global Constants - Boolean* 3b⟩≡
```
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.

3c ⟨*Use Modules - Benchmark* 3c⟩≡
```
use Benchmark;
  ⟨Timer ON 3d⟩
```

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

3d ⟨*Timer ON* 3d⟩≡
```
my @Timer = (new Benchmark);
```

3e ⟨*Common PERL subs - Benchmark* 3e⟩≡
```
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.

3f ⟨*Use Modules - Dumper* 3f⟩≡
```
use Data::Dumper;
local $Data::Dumper::Purity = 0;
local $Data::Dumper::Deepcopy = 1;
```

B.1.3 Common functions

4a ⟨*Skip comments and empty records* 4a⟩≡
```
next if /^\#/o;
next if /^\s*$/o;
chomp;
```

4b ⟨*Common PERL subs - Min Max* 4b⟩≡
```
#
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
```

4c ⟨*Common PERL subs - Text fill* 4c⟩≡
```
#
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).

4d ⟨*Common PERL subs - Counter* 4d⟩≡
```
#
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
```

4e ⟨*Global Vars - Counter* 4e⟩≡
```
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.

4f ⟨*Common PERL subs - STDERR* 4f⟩≡
```
sub report() { print STDERR sprintf($MessageList{ shift @_ },@_) }
```

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

4g ⟨*Common PERL subs - STDERR* 4f⟩+≡
```
sub warn() { print STDERR sprintf($ErrorList{ shift @_ }, @_) }
```

B.2 AWK scripts

5a ⟨*GAWK shebang* 5a⟩≡
```
#!/usr/bin/gawk -f
# GNU Awk 3.0.4
⟨Version Control Id Tag 5d⟩
```

B.3 BASH scripts

5b ⟨*BASH shebang* 5b⟩≡
```
#!/usr/bin/bash
# GNU bash, version 2.03.6(1)-release (i386-redhat-linux-gnu)
⟨Version Control Id Tag 5d⟩
#
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;
```

5c ⟨*BASH script end* 5c⟩≡
```
{ echo "###"; echo "### Execution time for [$0] : $SECONDS secs";
  echo "$L$L$L$L";
  echo ""; } 1>&2;
#
exit 0
```

B.4 Version control tags

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

5d ⟨*Version Control Id Tag* 5d⟩≡
```
# $Id: deploy.nw,v 1.9 2001/09/25 17:45:07 jabril Exp $
```

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.

6a ⟨*tangling* 6a⟩≡
```
# showing line numbering comments in program
notangle -L -R"humangenome" $WORK/$nwfile.nw | \
    perl -ne '$.>1 && print' | cpif $BIN/humangenome ;
chmod a+x $BIN/humangenome ;
```

6b ⟨*tangling* 6a⟩+≡
```
# reformating program with perltidy
notangle -R"humangenome" $WORK/$nwfile.nw | \
    perltidy - | cpif $BIN/humangenome ;
# html pretty-printing program with perltidy
notangle -R"humangenome" $WORK/$nwfile.nw | \
    perltidy -html - | cpif $DOCS/html/humangenome.html ;
#
```

C.2 Extracting different Config Files

6c ⟨*tangling* 6a⟩+≡
```
notangle -R"root" $WORK/$nwfile.nw | \
         cpif $DATA/root_config ;
```

C.3 Extracting documentation and LaTeX'ing it

6d ⟨*tangling* 6a⟩+≡
```
notangle -Rweaving  $WORK/$nwfile.nw | cpif $WORK/nw2tex ;
notangle -RLaTeXing $WORK/$nwfile.nw | cpif $WORK/ltx ;
chmod a+x $WORK/nw2tex $WORK/ltx;
```

6e ⟨*tangling complementary LaTeX files* 6e⟩≡
```
notangle -R"HIDE: LaTeX new definitions" $WORK/$nwfile.nw | cpif $DOCS/defs.tex ;
notangle -R"HIDE: TODO" $WORK/$nwfile.nw | cpif $DOCS/todo.tex ;
```

6f ⟨*weaving* 6f⟩≡
```
⟨BASH shebang 5b⟩
# weaving and LaTeXing
⟨BASH Environment Variables 7b⟩
⟨tangling complementary LaTeX files 6e⟩
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 5c⟩
```

7a ⟨*LaTeXing* 7a⟩≡
```
⟨BASH shebang 5b⟩
# only LaTeXing
⟨BASH Environment Variables 7b⟩
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 5c⟩
```

C.4 Defining working shell variables for the current project

7b ⟨*BASH Environment Variables* 7b⟩≡
```
#
# Setting Global Variables
WORK="/home/ug/jabril/development/projects/sgp/humangenome" ;
BIN="$WORK/bin" ;
PARAM="$BIN/param" ;
DOCS="$WORK/docs" ;
DATA="$WORK/data" ;
nwfile="humangenome" ;
export WORK BIN PARAM DOCS DATA nwfile ;
#
```

7c ⟨*tangling* 6a⟩+≡
```
#
# BASH Environment Variables
notangle -R'BASH Environment Variables' $WORK/$nwfile.nw | \
         cpif $WORK/.bash_VARS ;
source $WORK/.bash_VARS ;
#
```