sol genomics network

Writing Perl Test

sol genomics network

Boyce Thompson Institute for Plant Research
Tower Road
Ithaca, New York 14853-1801
U.S.A.

by
Aureliano Bombarely Gomez

BTI Boyce Thompson Institute for Plant Research

Writing Perl Test:

1. Why should Perl Code have test?

2. How to write a simple test.

3. Using Test::Simple.

4. Expanding horizonts: Test::More and Test::Exception

Writing Perl Test:

1. Why should Perl Code have test?
2. How to write a simple test.
3. Using Test::Simple.
4. Expanding horizonts: Test::More and Test::Exception

1. Why should Perl Code have test?

3 Good Reasons to Write Test:

- **Funcionality**:
  Code does the things that should do.

- **Integrability**:
  Code use/interact with other modules in the way that they should.

- **Maintenance**:
  If you change something, you need to know that it is working.

1. Why should Perl Code have test?

sol genomics network

Documentation:

- **CPAN**:
  http://search.cpan.org/dist/Test-Simple/lib/Test/Tutorial.pod.
- **Perldocs**:
  perldoc Test::Simple, Test::More and Test::Exception
- **Books:**
  Perl Testing: A Developer's Notebook.

1. Why should Perl Code have test?

Testing, a code writing mode.

- When you should a test for a piece of code ?

  ALWAYS

- How many test should have a piece of code ?

  A MINIMUM OF ONE PER FUNCTION.
  A MAXIMUM AS A MANY AS YOU FEEL CONFORTABLE.

- How you write the test script ?

  IN PARALLEL WITH YOUR CODE.

1. Why should Perl Code have test?

sol genomics network

A scientific point of view for CODE TESTING:

"Test are for code the same than positive and negative controls for experiments, you never will be sure of your results without them".

1.Why should Perl Code have test?

sol genomics network

Writing Test, two approaches:

1) Integrated with the script.
   1.1) Test files and output comparisson.
   1.2) Option that enables the Test variables.

2) A separated test script: MyModuleName.t

Writing Perl Test:

1. Why should Perl Code have test?
2. How to write a simple test.
3. Using Test::Simple.
4. Expanding horizonts: Test::More and Test::Exception

2. How to write a simple test.

User Case: Sam format flags are in bitwise format

http://samtools.sourceforge.net/samtools.shtml

SAM FORMAT

Sequence Alignment/Map (SAM) format is TAB-delimited. Apart from the header lines, which are started with the '@' symbol, each alignment line consists of:

Col	Field	Description
1	QNAME	Query template/pair NAME
2	FLAG	bitwise FLAG
3	RNAME	Reference sequence NAME
4	POS	1-based leftmost POSition/coordinate of clipped sequence
5	MAPQ	MAPping Quality (Phred-scaled)
6	CIAGR	extended CIGAR string
7	MRNM	Mate Reference sequence NaMe ('=' if same as RNAME)
8	MPOS	1-based Mate POSistion
9	TLEN	inferred Template LENgth (insert size)
10	SEQ	query SEQuence on the same strand as the reference
11	QUAL	query QUALity (ASCII-33 gives the Phred base quality)
12+	OPT	variable OPTional fields in the format TAG:VTYPE:VALUE

2. How to write a simple test.

sol genomics network

User Case: Sam format flags are in bitwise format

http://samtools.sourceforge.net/samtools.shtml

Each bit in the FLAG field is defined as:

Flag	Chr	Description
0x0001	p	the read is paired in sequencing
0x0002	P	the read is mapped in a proper pair
0x0004	u	the query sequence itself is unmapped
0x0008	U	the mate is unmapped
0x0010	r	strand of the query (1 for reverse)
0x0020	R	strand of the mate
0x0040	1	the read is the first read in a pair
0x0080	2	the read is the second read in a pair
0x0100	s	the alignment is not primary
0x0200	f	the read fails platform/vendor quality checks
0x0400	d	the read is either a PCR or an optical duplicate

where the second column gives the string representation of the FLAG field.

2. How to write a simple test.

User Case: Sam format flags are in bitwise format

http://samtools.sourceforge.net/samtools.shtml

Bitwise:

00000000001	=> 0x0001 => 2^0	= 1	=> PAIRED
00000000010	=> 0x0002 => 2^1	= 2	=> PAIR MAPPED
00000000100	=> 0x0004 => 2^2	= 4	=> READ UNMAPPED
00000001000	=> 0x0008 => 2^3	= 8	=> MATE UNMAPPED
00000010000	=> 0x0001 => 2^4	= 16	=> READ REVERSE
00000100000	=> 0x0002 => 2^5	= 32	=> MATE REVERSE
00001000000	=> 0x0004 => 2^6	= 64	=> FIRST IN PAIR
00010000000	=> 0x0008 => 2^7	= 128	=> SECOND IN PAIR
00100000000	=> 0x0001 => 2^8	= 256	=> ALIGN NOT PRIM.
01000000000	=> 0x0002 => 2^9	= 512	=> QUALITY FAILS
10000000000	=> 0x0004 => 2^10	= 1024	=> PCR DUPLICATE

2. How to write a simple test.

Example:

99 = 64 + 32 + 2 + 1

Bitwise:

00000000001	=> 0x0001 => 2^0	= 1	=> PAIRED
00000000010	=> 0x0002 => 2^1	= 2	=> PAIR MAPPED
00000000100	=> 0x0004 => 2^2	= 4	=> READ UNMAPPED
00000001000	=> 0x0008 => 2^3	= 8	=> MATE UNMAPPED
00000010000	=> 0x0001 => 2^4	= 16	=> READ REVERSE
00000100000	=> 0x0002 => 2^5	= 32	=> MATE REVERSE
00001000000	=> 0x0004 => 2^6	= 64	=> FIRST IN PAIR
00010000000	=> 0x0008 => 2^7	= 128	=> SECOND IN PAIR
00100000000	=> 0x0001 => 2^8	= 256	=> ALIGN NOT PRIM.
01000000000	=> 0x0002 => 2^9	= 512	=> QUALITY FAILS
10000000000	=> 0x0004 => 2^10	= 1024	=> PCR DUPLICATE

2. How to write a simple test.

Example:

145 = 128 + 16 + 1

Bitwise:

00000000001	=> 0x0001 => 2^0	= 1	=> PAIRED
00000000010	=> 0x0002 => 2^1	= 2	=> PAIR MAPPED
00000000100	=> 0x0004 => 2^2	= 4	=> READ UNMAPPED
00000001000	=> 0x0008 => 2^3	= 8	=> MATE UNMAPPED
00000010000	=> 0x0001 => 2^4	= 16	=> READ REVERSE
00000100000	=> 0x0002 => 2^5	= 32	=> MATE REVERSE
00001000000	=> 0x0004 => 2^6	= 64	=> FIRST IN PAIR
00010000000	=> 0x0008 => 2^7	= 128	=> SECOND IN PAIR
00100000000	=> 0x0001 => 2^8	= 256	=> ALIGN NOT PRIM.
01000000000	=> 0x0002 => 2^9	= 512	=> QUALITY FAILS
10000000000	=> 0x0004 => 2^10	= 1024	=> PCR DUPLICATE

2. How to write a simple test.

sol genomics network

```
package Sam::Flags;

use strict;
use warnings;

## CONSTRUCTOR

sub new {
        my $class = shift;
        my $flags = shift;

        my $self = bless( { } , $class);

        $self->set_flags($flags);
        return $self;
}
```

```
## ACCESSORS
sub set_flags {
        my $self = shift;
        my $flags = shift;

        if (defined $flags) {
                if ($flags !~ m/^\d+$/) {
                  die("ERROR: $flags isnt an integer.");
                }
                if ($flags > 2047) {
                  die("ERROR: Max. flag int.= 2047");
                }
        }
        $self->{flags} = $flags;
}
sub get_flags {
        my $self = shift;
        return $self->{flags};
}
```

2. How to write a simple test.

sol genomics network

```
## SYNOPSIS

## use Sam::Flags;

## my $samflag = Sam::Flags->new();

## $samflag->set_flag($flag);
## my $flag = $samflag->get_flag();
```

2. How to write a simple test.

sol genomics network

```
## TEST: sam_flags.t
use strict;
use warnigs;
use Sam::Flags;

my $samflag = Sam::Flags->new();

## Define the variable to test
my $flag = 64;

## Use the functions
$samflag->set_flags($flag);
my $get_flag = $samflag->get_flags();

## Check the result
if ($flag == $get_flag) {
      print STDERR “TEST OK.\n”;
}
else {
      print STDERR “TEST FAIL.\n”; }
```

Writing Perl Test:

1. Why should Perl Code have test?
2. How to write a simple test.
3. Using Test::Simple.
4. Expanding horizonts: Test::More and Test::Exception

3. Using Test::Simple

Test::Simple: A perl module to write test.

http://search.cpan.org/~mschwern/Test-Simple-0.98/lib/Test/Simple.pm

NAME

Test::Simple - Basic utilities for writing tests.

SYNOPSIS

```
use Test::Simple tests => 1;

ok( $foo eq $bar, 'foo is bar' );
```

Plan:
Specify the test number

ok function:
Check a variable

3. Using Test::Simple

sol genomics network

```
## TEST: sam_flags.t
use strict;
use warnings;
use Sam::Flags;
use Test::Simple tests => 1;

my $samflag = Sam::Flags->new();

## Define the variable to test
my $flag = 64;

## Use the functions
$samflag->set_flags($flag);
my $get_flag = $samflag->get_flags();

## Check the result
ok($flag == $get_flag, “ACCESSOR TEST OKAY”);
```

3. Using Test::Simple


```
## FUNCTION

sub flag2descriptions {
        my $self = shift;

        my %flags = (
                1           =>   'PAIRED',
                2           =>   'PAIR MAPPED',
                4           =>   'READ UNMAPPED',
                8           =>   'MATE UNMAPPED',
                16          =>   'READ REVERSE',
                32          =>   'MATE REVERSE',
                64          =>   'FIRST IN PAIR',
                128         =>   'SECOND IN PAIR',
                256         =>   'ALIGN NOT PRIM',
                512         =>   'QUALITY FAILS',
                1024        =>   'PCR DUPLICATE',
        );
```

```
        my $flag = $self->get_flags();

        my @descriptions = ();

        my @sfl = sort {$b <=> $a}  keys %flags;

        foreach my $fl (@sfl) {
                if ($fl <= $flag $$ $flag > 0) {
                        push @descriptions, $flags{$fl};
                        $flag -= $fl;
                }
        }

        return @descriptions;
}
```

3. Using Test::Simple

sol genomics network

```
## TEST: sam_flags.t

use strict;
use warnings;
use Sam::Flags;
use Test::Simple tests => 2;

My $flag = 64;
my $samflag = Sam::Flags->new($flag);

ok($flag == $samflag->get_flags,  “ACCESSOR TEST OKAY”);

my @expdesc1 = ('FIRST IN PAIR');

ok (join(sort($samflag->flag2descriptions())) eq join(sort(@expdec1)),
     “FLAG2DESCRIPTION TEST OKAY”);
```

3. Using Test::Simple

sol genomics network

```
## OUTPUT

1..2
ok 1 - ACCESSOR TEST OKAY
ok 2 - FLAG2DESCRIPTION TEST OKAY
```

Writing Perl Test:

1. Why should Perl Code have test?
2. How to write a simple test.
3. Using Test::Simple.
4. Expanding horizonts: Test::More and Test::Exception

4. Expanding horizonts: Test::More

Test::More: Another perl module to write test.

http://search.cpan.org/~mschwern/Test-Simple-0.98/lib/Test/More.pm

NAME

Test::More - yet another framework for writing test scripts

SYNOPSIS

```
use Test::More tests => 23;
# or
use Test::More skip_all => $reason;
# or
use Test::More;   # see done_testing()

BEGIN { use_ok( 'Some::Module' ); }
require_ok( 'Some::Module' );

# Various ways to say "ok"
ok($got eq $expected, $test_name);

is  ($got, $expected, $test_name);
isnt($got, $expected, $test_name);

# Rather than print STDERR "# here's what went wrong\n"
diag("here's what went wrong");

like  ($got, qr/expected/, $test_name);
unlike($got, qr/expected/, $test_name);

cmp_ok($got, '==', $expected, $test_name);
```

Plan:
Specify the test number

require_ok
Test modules

ok function:
is function
like function

4. Expanding horizonts: Test::More

Test::More: Another perl module to write test.

Functions:

is($got, $expected, $message)
=> *compare variables with 'eq' function*

like($got, '/expected/i', $message)
=> *compare using a regex*

unlike($got, '/expected/', $message)
=> *compare using !~ and a regex*

cmp_ok($got, $operator, $expected, $message)
=> *compare variables using an specific op.*

4. Expanding horizonts: Test::More

sol genomics network

```
## TEST: sam_flags.t

use strict;
use warnings;
use Sam::Flags;
use Test::More tests => 2;

My $flag = 64;
my $samflag = Sam::Flags->new($flag);

is($flag, $samflag->get_flags,  “ACCESSOR TEST OKAY”);

my @expdesc1 = ('FIRST IN PAIR');

like(join(sort($samflag->flag2descriptions())),  '/FIRST IN PAIR/i'),
     “FLAG2DESCRIPTION TEST OKAY”);
```

4. Expanding horizonts: Test::More

sol genomics network

Test::More: Another perl module to write test.

Functions:

can_ok($module, @methods)
=> *check that a module can use some methods*

isa_ok($object, $class, $object_name)
=> *check if an object is the right class*

use_ok($module_name)
=> *check that the module can be loaded*

diag(*$diagnostic_message*)
=> *print a diagnostic message*

4. Expanding horizonts: Test::More

sol genomics network

```
## TEST: sam_flags.t

use strict;
use warnings;
use Test::More tests => 5;

BEGIN { use_ok('Sam::Flags'); };

can_ok('Sam::Flags', qw/new get_flags set_flags flag2descriptions/);

my $flag = 64;
my $samflag = Sam::Flags->new($flag);
isa_ok($samflag, 'Sam::Flags');

is($flag, $samflag->get_flags,  “ACCESSOR TEST OKAY”)
            or diag(“Accessor test failed”);

my @expdesc1 = ('FIRST IN PAIR');

like(join(sort($samflag->flag2descriptions())),  '/FIRST IN PAIR/i'),
     “FLAG2DESCRIPTION TEST OKAY”) or diag(“Flag2description failed”);
```

4. Expanding horizonts: Test::More

sol genomics network

Test::Exception: Test exception based code.

http://search.cpan.org/~adie/Test-Exception-0.31/lib/Test/Exception.pm

NAME

Test::Exception - Test exception based code

SYNOPSIS

```
use Test::More tests => 5;
use Test::Exception;

# or if you don't need Test::More

use Test::Exception tests => 5;

# then...

# Check that the stringified exception matches given regex
throws_ok { $foo->method } qr/division by zero/, 'zero caught okay';

# Check an exception of the given class (or subclass) is thrown
throws_ok { $foo->method } 'Error::Simple', 'simple error thrown';
```

Plan:
Specify the test number

throws_ok function:

4. Expanding horizonts: Test::More

sol genomics network

Test::Exception: Test exception based code.

Functions:

throws_ok(BLOCK REGEX/CLASS, $description)
=> *check if a block is throw with a regex or a class.*

dies_ok(BLOCK, $description)
=> *check if a block dies*

lives_ok(BLOCK, $description)
=> *check if a block does not die*

4. Expanding horizonts: Test::More

sol genomics network

```
## TEST: sam_flags.t

use strict;
use warnings;
use Test::More tests => 5;
Use Test::Exception;

BEGIN { use_ok('Sam::Flags'); };

can_ok('Sam::Flags', qw/new get_flags set_flags flag2descriptions/);

my $flag = 64;
my $samflag = Sam::Flags->new($flag);
isa_ok($samflag, 'Sam::Flags');

is($flag, $samflag->get_flags,  “ACCESSOR TEST OKAY”)
            or diag(“Accessor test failed”);

throws_ok { $samflag->set_flags('NoInteger')} qr/ERROR: NoInteger/,
      “SET_FLAGS with wrong argument dies” ;
```

4. Expanding horizonts: Test::More

sol genomics network

```
## TEST: sam_flags.t


dies_ok { $samflag->set_flags(30000)},
      “SET_FLAGS with wrong argument dies (high integer)” ;
```